

По договору между издательством **"Вильямс"** и Интернет-Магазином "Books.Ru - Книги России" единственный легальный способ получения данного файла с книгой **“PHP 5 для начинающих ” (ISBN 5-8459-1039-0)** – покупка в Интернет-магазине "Books.Ru - Книги России".

Если вы получили данный файл каким-либо другим образом, вы нарушили законодательство об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству **"Вильямс"** где именно вы получили данный файл.

ББК 32.973.26-018.2.75
М52
УДК 681.3,07

Компьютерное издательство “Диалектика”

Зав. редакцией *С.Н. Тригуб*

Перевод с английского *В.А. Швеца*

По общим вопросам обращайтесь в издательство “Диалектика” по адресу:
info@diagnostika.com, http://www.diagnostika.com
115419, Москва, а/я 783; 031150, Киев, а/я 152

Мерсер, Дэйв У., **Кент**, Аллан, **Новицки**, Стивен, **Мерсер**, Дэвид,
Скуайер, Дэн, **Чой**, Ван Кью.

М52 РНР 5 для начинающих. : Пер. с англ. — М. : ООО И.Д. “Вильямс”, 2006. — 848 с. :
ил. — Парал. тит. англ.

ISBN 5-8459-1039-0 (рус.)

Эта книга представляет собой подробное учебное пособие для желающих освоить современную версию РНР. В книге описывается установка и конфигурирование РНР, основные понятия программирования, такие как переменные, циклы, условные операторы и массивы, а также основы объектно-ориентированного программирования и возможности его применения в РНР 5. Здесь также рассматриваются такие темы, как работа из РНР с HTTP-данными, использование XML, СУБД (MySQL и SQLite), работа с изображениями и создание РНР-сценариев командной строки. Кроме того, в книге описано проектирование приложений с помощью UML, PEAR-пакеты и методика повторного использования РНР-кода, а также обработка ошибок, тестирование и отладка приложений.

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства JOHN WILEY & Sons, Inc.

Copyright © 2006 by Diagnostika Computer Publishing.

Original English language edition Copyright © 2004 by Wiley Publishing, Inc., Indianapolis, Indiana.

All rights reserved including the right of reproduction in whole or in part in any form. This translation published by arrangement with Wiley Publishing, Inc.

Wiley, the Wiley Publishing logo, Wrox, the Wrox logo, and Programmer to Programmer are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

ISBN 5-8459-1039-0 (рус.)

ISBN 0-7645-5783-1 (англ.)

© Компьютерное издво “Диалектика”, 2006
перевод, оформление, макетирование

© by Wiley Publishing, Inc., 2004

Оглавление

Введение	24
Глава 1. Установка и использование PHP	34
Глава 2. Написание простых программ	65
Глава 3. PHP, HTML и состояние сеанса	99
Глава 4. Логические операторы, циклы и массивы	159
Глава 5. Надежный и понятный код	220
Глава 6. Создание высококачественного кода	264
Глава 7. Файлы и каталоги	292
Глава 8. XML	343
Глава 9. Введение в базы данных и SQL	372
Глава 10. Получение данных от MySQL с помощью PHP	418
Глава 11. Использование PHP для управления информацией в базах данных MySQL	442
Глава 12. Введение в объектно-ориентированное программирование	485
Глава 13. Работа с UML и классами	518
Глава 14. PEAR	552
Глава 15. PHP5 и электронная почта	589
Глава 16. Генерирование графики	606
Глава 17. Учебный пример: диспетчер протоколирования на PHP	633
Приложение А. Ответы	688
Приложение Б. Справочник по PHP-функциям	710
Приложение В. Использование SQLite	774
Приложение Г. ODBC	790
Приложение Д. CLI-интерфейс PHP	804
Приложение Е. Конфигурация PHP5	813
Предметный указатель	837

Содержание

Об авторах	22
Введение	24
Почему именно PHP?	26
Для кого предназначена эта книга	26
Темы, которые рассматриваются в данной книге	26
Как организована эта книга	27
Что требуется для использования данной книги	29
Использование командной строки	30
Соглашения	31
Исходный код	31
Ошибки	32
p2p.wrox.com	32
Глава 1. Установка и использование PHP	34
Истоки PHP	35
Установка, конфигурирование и запуск PHP	36
Системные требования	36
Конфигурационный файл PHP, php.ini	37
Настройка тестовой машины	37
Сетевые соединения	38
С чего начать?	38
Использование PHP 5	38
Установка PHP на Linux и Apache	39
Выбор метода установки	40
RPM-установка PHP 4	40
Компиляция PHP 5 из исходного кода	43
Настройка Apache для работы с PHP	50
Конфигурирование Apache для работы с PHP 5	51
Запуск или перезапуск Apache	53
Установка PHP 5 на Windows 2000/Internet Information Server 5	53
Загрузка дистрибутива PHP 5	55
php.ini и расширения	57
Тестирование и устранение неисправностей	61
Конфигурирование PHP	62
Файл php.ini	62
PHP-расширения	63
Кэширование	63
Резюме	63
Упражнения	64

Глава 2. Написание простых программ	65
Создание PHP-программы	66
Некоторые детали	67
Как работает PHP-код	68
Web-страница (пользовательский интерфейс)	68
Расширения файлов	69
PHP-разделители	69
Корректный PHP-код	70
Общие маркеры в коде	70
Как работают PHP-программы в Web-среде	70
Web-соединения: Internet-протоколы и HTTP	71
TCP/IP	72
HTTP-протокол	72
HTTP-запрос	73
HTTP-ответ	74
Тело ответа	75
Запуск PHP-сценариев посредством HTTP-запроса	76
Web-сервер	76
PHP-процессор	76
Использование переменных в PHP	76
Создание переменных	77
Именованые переменных	77
Типы данных	78
Область видимости переменной	80
Ключевое слово <code>global</code>	80
Статические переменные	80
Определение констант	81
Операторы и выражения	81
PHP-операторы	81
PHP-выражения	82
Типы операторов	83
Строковые операторы и функции	83
Как это работает	87
Арифметические операции в PHP	88
Как это работает	93
Массивы	94
Индексы массивов	94
Использование строк в качестве индексов массивов	95
Инициализация массивов	95
Работа с массивами	96
Сортировка массивов с помощью функций <code>sort()</code> и <code>asort()</code>	97
Резюме	98
Упражнения	98
Глава 3. PHP, HTML и состояние сеанса	99
Основы HTML	99
HTML DTD	101
Дескрипторы <code>form</code> и <code>input</code>	101

Доступ к PHP- и HTTP-данным	103
Предопределенные переменные	104
Переменные в HTTP-запросах и ответах	104
Как это работает	106
Суперглобальные массивы	106
Гиперссылки	109
Строки запросов	110
HTML-формы (или Web-формы)	111
Дескрипторы HTML-форм	111
Дескриптор <form>	111
Атрибуты дескриптора <form>	112
URL-кодирование	114
PHP и поля (элементы управления) HTML-форм	116
Текстовые поля (текстовые окна)	116
Как это работает	118
Почему этот пример может не работать	119
Текстовая область	119
Как это работает	120
Флажки	122
Как это работает	124
Использование нескольких флажков	124
Как это работает	126
Переключатели	127
Как это работает	129
Списки	130
Как это работает	131
Скрытые поля форм	134
Как это работает	136
Поля ввода паролей	137
Кнопки submit и reset	138
Использование в PHP-сценариях значений, возвращаемых	
формами	138
Как это работает	140
Возможные усовершенствования приложения	143
Понятие состояния	144
Контроль состояния	145
Скрытые поля форм	145
Строки запроса	146
Базы данных	146
Cookie-файлы	146
Как это работает	151
Собственные сеансы в PHP	153
Как это работает	155
Резюме	157
Упражнение	157
Глава 4. Логические операторы, циклы и массивы	159
Проектирование логики PHP-программы	159
Постановка задачи	160

Написание псевдокода	160
Булева логика	161
Булевы термы	161
Булевы значения	161
Использование булевых термов и значений	162
Условные операторы или операторы ветвления	164
Пример ветвления	164
Оператор if	165
Использование булевых операторов в структурах управляющей логики	166
Операторы > и <	166
Как это работает	168
Операторы == и ===	169
Операторы != и <>	170
Как это работает	172
Логические операторы (AND, OR, !)	172
Как это работает	174
Операторы switch	176
Как это работает	179
Циклы и массивы	181
Циклы	181
Цикл while	181
Как это работает	184
Бесконечные циклы	187
Циклы do while	188
Как это работает	189
Циклы for	191
Как это работает	193
Массивы	196
Инициализация массивов	197
Итерации в массиве	198
Как это работает	200
Усовершенствование программы	202
Итерации в неупорядоченных массивах	203
Функции current() и key()	203
Функции next() и prev()	204
Функции list() и each()	205
Итерации в ассоциативных массивах	206
Сортировка массивов	207
Функция sort()	207
Функция asort()	207
Функции rsort() и arsort()	208
Функция ksort()	209
Многомерные массивы	209
Практическое использование массивов	210
Как это работает	212
Функция array_multisort()	214

Циклы foreach	215
Как это работает	217
Резюме	217
Упражнения	218
Глава 5. Надежный и понятный код	220
Тестирование и отладка	221
Значения, нарушающие работу кода	221
Основные типы ошибок	222
Отладка PHP-сценария	223
Сообщения об ошибках PHP	223
Конфигурирование PHP для обработки ошибок	223
Типы ошибок в PHP	224
Синтаксические ошибки	224
Логические ошибки	227
Ошибки времени выполнения	227
Деление на ноль	228
Бесконечные циклы	230
Логические ошибки вывода	231
Присвоение значений вместо сравнения значений	232
Отладка и обработка ошибок в PHP5	232
Предотвращение отображения конфиденциальной информации	232
Создание собственных инструментов для отладки	232
Использование оператора echo()	233
Ошибки внутри HTML-кода	233
Проверка данных форм	234
Использование оператора exit	234
Как это работает	236
Предотвращение ввода пользователем HTML-кода: функция htmlspecialchars()	238
Проверка строк и регулярные выражения	239
Проверка строк	239
Регулярные выражения	241
Использование функции ereg()	242
Специальные символы	243
Некоторые распространенные образцы и их варианты	244
Проверка ввода данных	247
Как это работает	249
Использование регулярных выражений для проверки URL-указателей	250
Как это работает	251
Использование регулярных выражений для проверки параметров файловых путей	252
Как это работает	253
Изысканная обработка ошибок	254
Конфигурирование обработки ошибок в PHP	254
Подавление сообщений об ошибках	254
Проверка журнала ошибок	255

Try/Catch — нововведения в PHP5	255
Как это работает	259
Резюме	262
Упражнение	263
Глава 6. Создание высококачественного кода	264
Планирование разработки	265
Формальный процесс разработки программного обеспечения	265
Написание спецификации	265
Процесс написания кода	266
Тестирование, отладка и сопровождение	267
Оптимизация кода	267
Использование стандартов написания кода	268
Написание пользовательских функций в PHP	270
Структура функций	271
Определение и вызов функций	272
Как это работает	276
Функции переключения	277
Как значения попадают в функции	278
Передача по значению	279
Передача по ссылке	279
Установка значений по умолчанию	280
Важность порядка аргументов	280
Область видимости переменных	280
Глобальные и локальные переменные	281
Создание статических переменных в функциях	282
Как это работает	283
Вложенность функций	285
Рекурсия	285
Как это работает	287
Операторы include и require	288
Как это работает	290
О чем следует позаботиться при использовании подключаемых файлов	290
Резюме	291
Упражнение	291
Глава 7. Файлы и каталоги	292
Обработка файлов и каталогов	293
Работа с файлами	294
Открытие и закрытие файлов	294
Функция fopen()	294
Функция fclose()	297
Получение информации о файле	297
Чтение и запись файлов	298
Функция fread()	298
Функция fwrite()	298
Как это работает	300
Чтение и запись символов в файлы	302

Чтение файлов целиком	305
Произвольный доступ к данным файла	306
Как это работает	309
Получение информации о файлах	310
Временные свойства файлов	311
Как это работает	313
Принадлежность и права доступа к файлам	314
Функции <code>is_dir()</code> и <code>is_file()</code>	316
Как это работает	318
Пользовательские функции для работы с файлами	319
Разделение имени файла и пути	319
Копирование, переименование и удаление файлов	320
Работа с каталогами	322
Как это работает	323
Другие функции для обработки каталогов	324
Обход дерева каталогов	325
Сценарий для навигации по каталогам	327
Создание текстового редактора	332
Загрузка файлов на сервер	337
Как это работает	341
Резюме	342
Упражнение	342
Глава 8. XML	343
Что такое XML	344
Структура XML-документа	345
Основные части XML-документа	346
Правильно сформированные XML-документы	346
Использование XML-элементов и атрибутов	347
Корректные XML-документы: DTD-определения и XML-схемы	348
DTD-определение для XHTML	349
Ссылки на DTD-определения и XML-схемы	350
Внешние XML-схемы	351
Написание XML-документов с помощью XHTML	352
Web-службы	352
PHP и XML	353
XML-функции в PHP4	353
Как это работает	358
XML-анализаторы	359
Как это работает	362
Объектная модель документа	363
DOM-расширение	363
Использование функций DOM-расширения PHP	363
XML-функции PHP5	364
Расширение SimpleXML	364
Использование функции <code>simplexml_load_string()</code>	365
Использование функции <code>simplxml_load_file()</code>	366

Изменение значений с помощью simpleXML	368
Как это работает	370
Резюме	371
Упражнение	371
Глава 9. Введение в базы данных и SQL	372
Хранение данных	373
Базы данных	374
Архитектуры баз данных	374
Встроенные базы данных	374
Клиент/серверные базы данных	375
Выбор базы данных	375
Установка MySQL	376
Установка на Windows	377
Установка MySQL на Linux	377
Установка MySQL из RPM-пакетов	378
Установка MySQL из исходного кода	379
Конфигурирование MySQL	380
Реляционные базы данных	381
Нормализация	382
Первая нормальная форма (1NF)	382
Вторая нормальная форма (2NF)	384
Третья нормальная форма (3NF)	384
Другие нормальные формы	385
Обращение к базам данных с помощью SQL	385
Типы данных в SQL	385
Индексы и ключи	387
Запросы	388
Практическое применение MySQL	390
Запуск клиентской программы mysql	390
Выбор используемой базы данных	391
Просмотр таблиц в базе данных	392
Использование SQL для просмотра данных	393
Манипуляции данными	394
Использование команд GRANT и REVOKE	395
GRANT	395
REVOKE	396
Подключение к базам данных MySQL из PHP-программ	397
Связь PHP и MySQL	397
Основные функции соединения	398
Как это работает	400
Обработка серверных ошибок	401
Как это работает	404
Создание баз данных и таблиц с помощью MySQL-клиента	405
Создание демонстрационной базы данных и таблиц с помощью PHP	409
Как это работает	411
Изменение структуры таблицы	412
Вставка данных в таблицу	414

Экранирование кавычек	415
Заполнение таблиц данными	416
Резюме	417
Глава 10. Получение данных от MySQL с помощью PHP	418
Получение данных с помощью PHP	418
SQL-операторы для выборки данных	421
Серверные функции	421
Выбираемые поля	422
Ограничение количества возвращаемых результатов	423
Упорядочение результатов	425
Сравнение с образцом	426
Получение итоговых данных	427
Более сложные выборки	427
Практический пример сценария	431
Содержимое файла common_db.inc	432
Глобальные переменные	432
Функция html_header()	432
Функция html_footer()	433
Функция error_message()	433
Содержимое файла userviewer.php	433
Функция list_records()	434
Функция view_record()	437
Выбор действия	440
Использование сценария	440
Резюме	440
Глава 11. Использование PHP для управления информацией в базах данных MySQL	442
Вставка записей с помощью PHP	442
Специальные символы	443
Функция htmlspecialchars()	444
Обновление и удаление записей в таблицах	445
Работа с полями даты и времени	447
Получение информации о таблицах в базе данных	450
Как это работает	451
Как это работает	454
Параметры ENUM и стандартные значения полей	456
Как это работает	458
Создание сценария для регистрации пользователей	459
Сценарий register.php	460
Функция in_use()	460
Функция register_form()	461
Функция create_account()	462
Выбор следующего действия	464
Создание сценария для протоколирования посещений страниц	466
Файл auth_user.php	467
Файл access_logger.php	469

Создание сценария для управления пользователями	473
Файл userman.php	473
Функция user_message()	474
Функция list_records()	474
Функция delete_record()	474
Функция edit_record()	475
Функция edit_log_record()	476
Функция view_record()	477
Выбор действия	480
Резюме	482
Упражнения	483
 Глава 12. Введение в объектно-ориентированное программирование	 485
Что такое объектно-ориентированное программирование?	485
Основные понятия ОО-программирования	487
Классы	488
Объекты	489
Создание класса	489
Добавление методов	490
Добавление свойства	491
Ограничение доступа к переменным экземпляра	492
Как это работает	493
Использование функций __get и __set	494
Как это работает	495
Инициализация объектов	496
Как это работает	499
Уничтожение объекта	499
Как это работает	501
Наследование	502
Как это работает	506
Переопределение методов	507
Сохранение функциональности родительского класса	510
Как это работает	512
Интерфейсы	512
Как это работает	513
Инкапсуляция	514
Изменения объектно-ориентированных возможностей PHP5	516
Резюме	517
Упражнения	517
 Глава 13. Работа с UML и классами	 518
Унифицированный язык моделирования	518
Зачем использовать UML?	519
Инструменты для создания UML-диаграмм	519
Диаграммы классов	520
Создание диспетчера контактов	521

UML-диаграммы диспетчера контактов	521
Другие полезные UML-диаграммы	526
Диаграммы активности	526
Диаграммы ситуаций	527
Диаграмма последовательностей	527
Создание класса Entity	529
Сборка классов	535
Класс PropertyObject	535
Классы, представляющие типы контактной информации	537
Класс DataManager	541
Классы Entity, Individual и Organization	542
Использование системы	549
Резюме	551
Глава 14. PEAR	552
Что такое PEAR?	553
Структура PEAR	553
Базовые классы PHP-кода	554
Общественная библиотека PHP-расширений	554
Диспетчер пакетов PEAR	555
Обзор PEAR-стандартов	555
Управляющие структуры, комментарии и отступы	555
Вызовы и определения функций	556
Соглашения по именованию	557
Установка PEAR-пакетов	557
Поиск PEAR-пакетов на сайте pear.php.net	557
Изучение PEAR-классов и приложений	558
Установка и использование диспетчера пакетов PEAR	558
Установка диспетчера PEAR-пакетов для Windows	559
Использование диспетчера пакетов	559
Использование PEAR-пакетов	569
Как это работает	572
Что делать, если найдена проблема?	574
Создание приложения с использованием двух PEAR-компонентов	574
Приложение	574
Архитектура	575
Генерация XML-кода	575
Сценарий radiogeneratexml.php	577
Сценарий radiorequest.php	578
Как это работает: radiogeneratexml.php	581
Как это работает: сценарий radiorequest.php	585
Резюме	587
Глава 15. PHP5 и электронная почта	589
Основы e-mail	589
Почтовые протоколы Internet	590
Структура e-mail-сообщения	590
Отправка e-mail с помощью PHP	592

Использование функции mail()	592
Как это работает	593
Многоцелевые расширения почты в Internet (MIME)	595
Поля MIME-заголовков	595
Многоэлементный MIME-формат	595
Библиотеки почтовых функций в PEAR	597
Создание простого RНР-приложения для работы с e-mail	597
Резюме	604
Упражнение	605
Глава 16. Генерирование графики	606
Основы компьютерной графики	607
Теория цвета	607
Системы координат	607
Типы изображений	608
Работа с растровыми изображениями	609
Создание нового изображения	609
Распределение цветов	609
Основные функции рисования	610
Рисование отдельных пикселей	610
Рисование линий	611
Как это работает	612
Рисование прямоугольников	612
Рисование окружностей и эллипсов	613
Рисование дуг	613
Рисование многоугольников	614
Как это работает	616
Изменение растровых изображений	618
Открытие существующего изображения	618
Как это работает	620
Внедрение водяных знаков	620
Копирование copyright-знака в изображение	620
Использование прозрачности	622
Использование непрозрачности	624
Создание пиктограмм	624
Использование текста в изображениях	627
Добавление стандартного текста	628
Как это работает	628
Использование шрифтов True Type	629
Как это работает	631
Резюме	631
Упражнения	632
Глава 17. Учебный пример: диспетчер протоколирования на RНР	633
Почему именно диспетчер протоколирования?	634
Smarty	635

Установка Smarty	635
Как это работает	637
PHPUnit	640
Работа с PHPUnit	640
Проектирование диспетчера протоколирования	641
База данных sitelogs.db	642
Использование UML для планирования диспетчера протоколирования	644
Планирование классов обработки данных	644
Планирование классов обработки исключений	647
Диаграмма последовательностей диспетчера протоколирования	648
Код приложения	648
Вспомогательные сценарии	648
settings.php	648
common.php	649
setup.php	649
initialize.php	649
Сценарии обработки данных	651
class.LogUtils.php	651
class.PersistableLog.php	655
class.UserLog.php	657
class.LogContainer.php	661
class.UserDemographic.php	662
Сценарии проверки данных и обработки ошибок	664
Сценарии уровня представления и шаблоны	669
index.php	669
report.php	670
report.tpl	671
report-html.tpl	672
Тестирование приложения	672
Работа с диспетчером протоколирования	683
userlog.php	683
Просмотр интерфейса диспетчера протоколирования	683
Резюме	687
Приложение А. Ответы	688
Глава 1	688
Упражнение	688
Решение	689
Глава 2	689
Упражнение 1	689
Решение	689
Упражнение 2	690
Решение	690
Глава 3	690
Упражнение	690
Решение	690

Глава 4	691
Упражнение	691
Решение	692
Глава 5	694
Упражнение	694
Решение	694
Глава 6	694
Упражнение	694
Решение	695
Глава 7	696
Упражнение	696
Решение	696
Глава 8	696
Упражнение	696
Решение	696
Глава 11	697
Упражнение 1	697
Решение	697
Упражнение 2	697
Решение	698
Упражнение 3	698
Решение	698
Упражнение 4	699
Решение	699
Упражнение 5	699
Решение	699
Упражнение 6	699
Решение	699
Глава 12	703
Упражнение 1	703
Решение	703
Упражнение 2	703
Решение	703
Упражнение 3	704
Решение	704
Глава 15	704
Упражнение	704
Решение	704
Глава 16	705
Упражнение 1	705
Решение	705
Упражнение 2	706
Решение	706
Приложение Б. Справочник по PHP-функциям	710
Web-сервер Apache	710
Массивы	711
BCMath	716

BZip2	717
Календарь	717
Классы и объекты	718
Типы символов	719
Curl	720
Дата и время	720
Каталоги	722
Обработка ошибок	722
Файловая система	723
FTP-функции	726
Вызов функций	728
HTTP-функции	729
Библиотека iconv	730
Функции для работы с изображениями	730
IMAP-функции	736
Mail-функции	740
Math-функции	741
MIME-функции	743
Разные функции	743
Функции MS SQL	744
Функции MySQL	746
Сетевые функции	748
ODBC-функции	749
Буферизация вывода	752
Функции PCRE	752
Параметры PHP и информация о PHP	753
Выполнение программ	755
Регулярные выражения	756
Сеансы	757
Simple XML	758
Сокеты	758
Функции SQLite	760
Расширение Streams	762
Строки	764
URL-функции	770
Функции переменных	770
XML-функции	772
ZLib	773
Приложение В. Использование SQLite	774
Что такое SQLite?	774
Как получить SQLite?	775
Почему стоит (или не стоит) использовать SQLite?	775
Использование SQLite в PHP	776
Создание и поддержка соединений	776
Манипуляция данными	777
Сведения об извлекаемых данных	778
Разные функции	779

Практическое использование SQLite	780
Приложение “Персональная библиотека”	780
Создание базы данных и таблиц	780
Форма ввода данных	782
Главная страница и листинг книг	785
Редактирование записи	786
Приложение Г. ODBC	790
Общие ODBC-функции	790
ODBC-функции в PHP	791
Другие ODBC-функции	792
Использование ODBC в Windows и Linux	792
ODBC-параметры в конфигурации PHP	793
Пример использования ODBC и SQL Server в PHP-приложении для Windows	793
Создание базы данных Microsoft SQL Server	793
Создание системного DSN	795
Использование ODBC-функций PHP	799
Приложение Д. CLI-интерфейс PHP	804
Начало	804
Некоторые важные моменты	805
Обработка параметров командной строки	807
Аргументы в PHP CLI	807
Запуск shell-команды	808
Автоматизация PHP CLI	809
Интерактивность средствами PHP CLI	811
Заключение	812
Приложение Е. Конфигурация PHP5	813
Описание файла php.ini-dist	813
Настройки языка	815
Ограничения ресурсов	818
Обработка и протоколирование ошибок	819
Обработка данных	821
Пути и каталоги	823
Загрузка файлов на сервер	824
Обработчики функции fopen	824
Динамически загружаемые расширения	825
Настройки расширений	826
Предметный указатель	837

Об авторах

Дэйв У. Мерсер

Дэйв У. Мерсер (Dave W. Mercer) имеет пятнадцатилетний опыт организации производства и системного анализа. Руководитель технического отдела по обслуживанию корпоративных клиентов, отвечает за разработку и внедрение автоматизированных сетевых бизнес-служб. Весь его сайт, хостинг-сервер, а также приложения, которые он разрабатывает для своих клиентов, написаны на PHP с использованием серверов баз данных PostgreSQL и MySQL.

Аллан Кент

Аллан Кент (Allan Kent) — PHP-программист, владеющий собственной компанией, соавтор книги *Beginning PHP 4*. Серьезно занимается программированием. Не считая официального курса обучения программированию на Cobol, в результате которого Аллан получил диплом, он научился всему полностью самостоятельно.

Стивен Д. Новицки

Стивен Д. Новицки (Steven D. Nowicki) — директор отдела разработки программного обеспечения в калифорнийской консалтинговой фирме в Санта-Монике *The Content Project*. Эта фирма разрабатывает массивную систему планирования ресурсов предприятия и учета клиентов, которая состоит более чем из 300 000 строк объектно-ориентированного PHP-кода. Стивен имеет десятилетний опыт работы в качестве разработчика крупномасштабных программ и системного архитектора для всех ведущих платформ.

Дэвид Мерсер

Дэвид Мерсер (David Mercer) — PHP-программист, внесший свой вклад в книгу *Beginning PHP 4*. Он сохранил свое увлечение открытым исходным кодом с тех пор как ему удалось собрать работающий Beowulf-кластер из старых компьютерных деталей. Мерсер — автор нескольких книг издательства Wrox о продуктах с открытым исходным кодом PHP, Perl и Linux.

Дэн Скуайер

Дэн Скуайер (Dan Squier) — PHP-программист, активный участник сообщества Wrox.

Ван Кью Чой

Ван Кью Чой (Wankyu Choi) — профессиональный PHP-программист и ведущий автор книги *Beginning PHP 4*. Дипломированный переводчик корейского языка.

Хью Айде-Гудман

Хью Айде-Гудман (Heow Eide-Goodman) — член сообществ NYPHP и LispNYC, использующий в своей повседневной работе PHP для создания Web-сайтов, служб и офисных продуктов на базе SQL Server, Interbase/Firebird и MySQL.

Эдвард Лекки-Томпсон

Эдвард Лекки-Томпсон (Edward Lecky-Thompson) — основатель и директор Ashridge New Media, компании, оказывающей консультационные услуги по новым медиа-технологиям. Эд более шести лет разрабатывает коммерческое программное обеспечение и системные архитектуры уровня предприятия для множества платформ, в частности для PHP и Apache на Linux-платформах.

Кларк Морган

Кларк Морган (Clark Morgan) — опытный программист, который разрабатывает и администрирует базы данных, а также Web-сайты с использованием PHP и MySQL для компании Fusion Computing and Media.

Эта книга является плодом усилий многих людей. Нынешняя команда авторов посвящает эту книгу авторам предыдущего издания за их великолепное введение в ранние версии PHP, а также редакторам и менеджерам, чей тяжелый труд помог нам создать введение в новую версию PHP, в частности Дебре Уильямс Каули (Debra Williams Cauley), Мэриэнн Стейнхарт (Maryann Steinhart) и Дэвиду Мерсеру. Кроме того, авторы выражают благодарность разработчикам PHP, Zend Engine и всем, кто внес свой вклад в разработку программного обеспечения с открытым исходным кодом и примеров кода, которые делают PHP идеальным языком сценариев для Web-приложений.

Издательский дом "Вильямс" благодарит Кущенко Сергея за большой вклад в подготовку издания книги.

Введение

PHP 5 — последнее воплощение PHP (PHP: Hypertext Preprocessor) — языка программирования, который был первоначально создан в 1994 году Расмусом Лердорфом (Rasmus Lerdorf) для разработки динамических, интерактивных Web-сайтов. С тех пор PHP благодаря усилиям многих разработчиков постепенно становится полноценным языком программирования.

Верными признаками того, что PHP созревает как технология, являются полностью исправленная и обновленная поддержка принципов объектно-ориентированного программирования, а также усовершенствованная поддержка XML. Процессор Zend Engine (программа, которая интерпретирует и выполняет PHP-код) в настоящее время позволяет PHP-разработчикам среди прочего реализовать изящную обработку ошибок в масштабах приложения.

Все новые возможности и функциональность, предоставляемая PHP 5, требуют “обновления” знаний программистов с тем, чтобы они могли наилучшим образом использовать этот мощный язык Web-сценариев. Именно поэтому для читателя очень важно уделить время изучению самых последних и самых лучших возможностей, предоставляемых PHP 5.

Итак, что же такое PHP?

Известно, что PHP — язык для написания компьютерных программ, поэтому на самом деле интересен другой вопрос: *какие* программы можно писать с помощью PHP? В технических терминах PHP находит свое главное применение в качестве кросс-платформенного, HTML-совместимого, серверного языка Web-сценариев. Рассмотрим вкратце эти понятия.

- ❑ **Кросс-платформенность:** почти весь PHP-код можно без изменения использовать на компьютерах, работающих под управлением различных операционных систем. Например, PHP-сценарий, работающий на Linux-машине, как правило, так же хорошо будет работать на компьютере под управлением Windows.
- ❑ **HTML-совместимость:** PHP-код может быть написан в файлах, содержащих смесь PHP-инструкций и HTML-кода.
- ❑ **Серверный язык:** PHP-программы работают на сервере, а именно — на Web-сервере.
- ❑ **Язык написания Web-сценариев:** PHP-программы запускаются посредством Web-браузеров.

Это означает, что на PHP пишутся программы, работающие на Web-сервере, в которых PHP-код смешивается с HTML-кодом. Доступ к этим программам реализуется посредством Web-браузера, отображающего результат PHP-обработки, возвращенный Web-сервером в виде HTML-кода. Иными словами, можно создавать программы, доступные для всеобщего просмотра через Web, просто размещая их на общедоступном Web-сервере.

Читатель, вероятно, уже знаком с HTML (HyperText Markup Language — язык гипертекстовой разметки) — основным языком, используемым для создания Web-стра-

ниц, в котором обычный текст комбинируется со специальными тегами, определяющими интерпретацию этого текста браузерами. Язык HTML используется для описания того, как должны отображаться различные элементы Web-страницы, как страницы должны быть связаны друг с другом, где размещаются изображения и т.д.

Чистые HTML-документы, несмотря на всю свою универсальность, — на самом деле не просто красиво представленная статическая организация текста и рисунков. Большинство Web-сайтов являются не статическими, а динамическими и даже интерактивными. Они способны показывать список статей, содержащих определенное слово, интересующее посетителя, или последние новости, или даже приветствовать посетителя по имени при регистрации в системе. Такие сайты предоставляют пользователю возможность взаимодействовать с ними и предлагают различную информацию в соответствии с пользовательским выбором.

Подобные сайты невозможно создать, используя чистый HTML-код. Здесь на помощь приходит язык PHP, позволяющий программировать сайты, которые:

- ❑ представляют данные из многих различных источников, таких как базы данных, файлы или даже другие Web-страницы;
- ❑ включают в себя интерактивные элементы, например, средства поиска, обмен сообщениями и опросы общественного мнения;
- ❑ дают пользователю возможность выполнять некоторые действия, например, отправлять email-сообщения или делать покупки.

Иначе говоря, PHP можно использовать для написания таких сайтов, с которыми регулярные пользователи Web сталкиваются ежедневно. Начиная от поисковых машин и информационных порталов и заканчивая сайтами электронной коммерции, большинство Web-сайтов включают в себя некоторые или все эти виды программирования. Среди прочего в данной книге рассказывается, как с помощью PHP создать:

- ❑ простой текстовый редактор на Web-странице;
- ❑ Web-приложение для отправки с email-сообщений;
- ❑ объектно-ориентированное приложение для управления контактной информацией;
- ❑ объектно-ориентированную программу для протоколирования.

Язык PHP 5 можно использовать для создания широкого диапазона приложений: от простых утилит, таких как текстовый редактор, до мощных Web-приложений, таких как диспетчер протоколирования. Эта книга научит читателя создавать с помощью PHP 5 любые Web-сайты. Читатель познакомится с несколькими полезными методиками и, возможно, почерпнет идеи, которые затем можно будет внедрить в реальные Web-сайты и приложения.

Web-сценарии, безусловно, — основа успеха PHP, однако они не являются единственным способом применения данного языка. Сценарии командной строки, т.е. использование CLI-интерфейса (Command Line Interface — интерфейс командной строки), которые появились в PHP 4, — один из многих популярных видов применения PHP. (CLI-интерфейс рассматривается в приложении в конце данной книги.) Другим видом является разработка графических пользовательских интерфейсов клиентской стороны с помощью библиотеки GTK (Gnome ToolKit).

Почему именно PHP?

Одно из преимуществ PHP состоит в том, что его поддерживают многие провайдеры Internet-услуг (ISP) и Web-хостинговые компании. В настоящее время PHP используют сотни тысяч разработчиков, и это не удивительно, если учесть, что PHP установлен на нескольких миллионах узлов.

Читателю, скорее всего, уже известно, что PHP является кросс-платформенной технологией и что как только Web-страница написана, ее можно очень легко заставить работать на собственном Web-сервере. Однако, как сравнить PHP с другими технологиями? Сравнить PHP с Perl достаточно сложно, потому что эти языки предназначены для решения разных задач. Язык PHP, в отличие от Perl, был специально разработан для быстрого создания динамических Web-страниц. В результате Perl может оказаться слишком сложным и дорогим для пользователей, которые хотят создавать Web-страницы. Целесообразнее сравнивать PHP с ASP, однако за ASP необходимо платить, к тому же этот язык должным образом не работает на различных платформах — его приходится использовать только на коммерческих платформах, за что также необходимо платить.

Возникает вопрос: а есть ли у PHP недостатки? В прошлом PHP часто критиковали за то, как в нем организована обработка данных — например, одним из главных препятствий для PHP был его способ реализации поддержки объектов. Разработчики PHP 5 внимательно рассмотрели все недостатки его предшественников и, там где это понадобилось, полностью переписали реализацию его функциональности. В настоящее время PHP — серьезный претендент на крупномасштабную промышленную разработку, он располагает значительной, объединенной базой небольших и средних приложений.

Для кого предназначена эта книга

Как очевидно из названия, данная книга предназначена для тех, кто начинает работать с PHP 5 — это не только начинающие программисты, создающие программы впервые, но и опытные, издавшие виды разработчики, которые хотят выяснить возможности последней версии PHP.

Авторы пытались по возможности сделать материал этой книги в равной степени полезным для всех читателей, и все же очевидно, что некоторые разделы будут востребованы одной группой и менее интересны для другой. Например, пользователи, которые уже устанавливали и запускали PHP5 на своих машинах, могут спокойно пропустить главу 1, целиком посвященную именно этим вопросам.

Темы, которые рассматриваются в данной книге

Главная цель данной книги заключается в том, чтобы читатель получил широкое представление о PHP 5, а также о связанных технологиях и темах. В этой книге рассматривается очень много вопросов, поэтому для решения каких-либо конкретных проблем читателю следует обратиться к более специализированным книгам. Например, если вас главным образом интересует использование PHP 5 совместно с MySQL, то, скорее всего, лучше будет изучить материал, где этот аспект PHP рассматривается подробно. Если же вы уже хорошо знакомы со всеми аспектами PHP-программирова-

ния, то ознакомиться с более сложными темами вам поможет книга *PHP 5 для профессионалов* (ИД “Вильямс”, 2006г).

Ниже приведен перечень некоторых наиболее важных тем, обсуждаемых в книге.

- ☐ Установка и настройка.
- ☐ Основы — переменные, циклы, условные операторы и массивы.
- ☐ Методики и практические приемы программирования — создание и поддержка эффективного и надежного кода.
- ☐ Работа с данными.
- ☐ Обработка файлов и каталогов.
- ☐ XML.
- ☐ PHP 5 и базы данных (в частности, MySQL и SQLite).
- ☐ Объектно-ориентированное программирование.
- ☐ PEAR.
- ☐ E-mail.
- ☐ Графика.
- ☐ CLI-интерфейс.

Естественно, здесь рассматриваются и многие другие вопросы, однако этот список должен дать читателю представление о том, чего можно ожидать от книги *на PHP 5 для начинающих*.

Как организована эта книга

Вся информация в книге организована логичным и последовательным способом. Это означает, что без особой необходимости новые понятия или темы не рассматриваются без предварительного ознакомления с ними. Например, до того как узнать о создании метода класса, вы уже познакомитесь с понятием функции, и поэтому вам не придется тратить время, “перепрыгивая” с одного раздела на другой, чтобы понять обсуждаемую тему.

С этой целью в нескольких первых главах представлены основы PHP — как его получить, как заставить его работать и как с его помощью выполнять некоторые простейшие операции. Впоследствии, когда читатель уже будет обладать определенным уровнем знаний, будут рассмотрены более сложные темы.

Далее представлены краткие аннотации к главам книги, которые помогут читателю определить, следует ли читать ее от начала до конца или лучше просто выбирать определенные главы. Читатели, уже знакомые с PHP, естественно, захотят углубиться в некоторые особенно интересные их главы.

Глава 1 — начало работы. Вслед за кратким обсуждением PHP будут представлены пошаговые инструкции по установке PHP на Linux- и Windows-машины. Кроме того, в данной главе описана установка и конфигурирование Web-сервера (IIS или Apache), с тем чтобы можно было просматривать PHP-страницы в Web-браузере. На случай, если в процессе установки и настройки возникнут какие-либо проблемы, в конце этой главы имеется раздел с полезной информацией по устранению неисправностей и отладке.

В главе 2 представлен небольшой PHP-сценарий в действии. Здесь рассматриваются Internet-протоколы и HTTP, а также связь PHP с ними. Примеры сценариев

позволят читателю понять, как PHP вписывается в общую картину Web. Затем начинается подробное изучение основных понятий языка: читатель познакомится с переменными, типами данных и выражениями.

В главе 3 показано, как можно использовать информацию, передаваемую как часть HTTP-запросов и ответов, для сбора полезных данных о различных аспектах программной среды. В частности, в главе рассматривается серверная информация, методы GET и POST, cookie-файлы и т.д. Множество практических примеров проиллюстрируют, как можно задействовать HTML-формы и их элементы для сбора информации, используемой в PHP-сценариях. Здесь также рассматриваются такие темы, как протокол HTTP, сеансы и состояние соединений.

В главе 4 рассматривается одна из наиболее важных тем в изучении любого языка: программная логика. Описываются операторы сравнения, условные операторы, а также различные выражения, например, `if` и `switch`, в которых они используются. Не менее важны циклы, которые позволяют сравнительно просто выполнять повторяющиеся действия. Хорошее понимание этих основ позволит вам разобраться с несколько более сложной концепцией массивов и изучить использование циклов, в том числе и для манипулирования данными в массивах.

Глава 5 предостерегает читателя от плохой практики программирования. В главе рассматривается отладка и тестирование, а также использование PHP 5 для изящной обработки ошибок. В данной главе представлены полезные примеры по поддержанию эффективного, надежного и устойчивого кода, здесь также показано, как можно использовать проверку форм с тем чтобы предохранить программы от ложных или бессмысленных данных.

Глава 6 по своему названию и содержанию связана с предыдущей главой; здесь представлены хорошие практические приемы программирования. Читатель познакомится со всеми аспектами создания высококачественного кода: от проектирования, оптимизации и представления до разбиения на модули (при написании пользовательских функций), а также с важными связанными понятиями, такими как область действия.

В главе 7 всесторонне рассматривается возможность работы PHP с файлами и каталогами. Открытие, закрытие, чтение и запись в файлы являются основными операциями многих PHP-программ. В данной главе подробно описываются эти операции, а также другие важные вопросы, такие как права доступа к файлам и владение файлами. В качестве примера описанных в данной главе функций рассматривается создание редактора текстовых файлов.

В главе 8 иллюстрируется интереснейшая технология XML — популярный метод передачи структурированной информации. PHP 5 оснащен новыми функциями, которые позволяют быстрее и эффективнее работать с XML. Множество примеров данной главы раскроют читателю все преимущества языка XML.

В главах 9–11 представлено хорошее руководство по сохранению, извлечению и обработке данных с помощью PHP и СУРБД (систем управления реляционными базами данных), а именно — MySQL. В указанных главах рассматривается установка и использование MySQL для изучения основ архитектуры реляционных баз данных. Читатель познакомится с основными PHP-функциями, необходимыми для выполнения многих базовых операций, таких как создание баз данных и подключение к ним, а также вставка, извлечение, обновление и изменение информации. Эти знания используются для разработки связанных программ, которые фигурируют в качестве представленных примеров.

В главах 12 и 13 обсуждается одно из наиболее важных понятий в современном программировании: объекты. Глава 12 знакомит читателя с теоретическими основами объектно-ориентированного программирования и охватывает такие важные темы,

как наследование, инкапсуляция и абстракция, а также представляет примеры, демонстрирующие поддержку объектно-ориентированного программирования в PHP 5. В главе 13 рассматривается практическое использование ОО-программирования на примере создания объектно-ориентированного приложения, спроектированного с помощью UML (Unified Modeling Language — унифицированный язык моделирования), который рассматривается в начале данной главы.

Глава 14 знакомит читателя с библиотекой PEAR (PHP Extension and Application Repository — репозиторий PHP-расширений и приложений). Здесь рассматривается методика поиска и применения PEAR-пакетов для добавления функциональности в разрабатываемые приложения. В качестве примеров разрабатывается несколько приложений, использующих PEAR-пакеты для решения различных задач.

В главе 15 подробно изучается тема e-mail. Изложены основы электронной почты и связанных с ней протоколов и технологий, описывается PHP5-поддержка e-mail-функциональности, в главе также обсуждаются MIME-сообщения (Multipurpose Internet Mail Extensions — многоцелевые почтовые расширения Internet) и рассматривается пример приложения, демонстрирующего на практике присоединение к e-mail-сообщению файла с помощью MIME.

В главе 16 показано, как использовать библиотеку GD для создания графики в Web-страницах. Рассматриваются примеры создания и использования изображений, рисование линий и фигур, а также работа с текстом в GD.

Глава 17 поможет читателю объединить полученные знания и разработать более крупное приложение. В данной главе на примере разработки агента протоколирования описывается весь процесс создания сложного программного решения.

В приложении А даны ответы к упражнениям, представленным почти в каждой главе.

Приложение Б представляет собой справочник по функциям, который, несомненно, пригодится разработчику, учитывая большое количество собственных PHP-функций.

В приложении В обсуждается база данных SQLite, поставляемая с PHP 5. В этом приложении имеются практические примеры SQLite в действии.

В приложении Г дано краткое описание ODBC (Open Database Connectivity — интерфейс открытого взаимодействия с базами данных). Рассматривается сущность и функции этого интерфейса, а также возможность его применения для подключения к различным базам данных SQL. В частности, описывается установка ODBC с Microsoft SQL Server и использование ODBC-функций PHP 5 для подключения к SQL Server.

В приложении Д поясняется использование интерпретатора командной строки в PHP для решения не Web-задач с помощью PHP 5. CLI-интерфейс предоставляет пользователям PHP 5 возможность применять средства командной строки, которые раньше были доступны пользователям Perl, BASH и DOS.

В приложении Е подробно изучается файл `php.ini`, содержащий все конфигурационные параметры PHP 5. Данное приложение представляет собой полезный справочник по всем сложным настройкам (не рассмотренным в основной части книги), необходимость изменения которых может возникнуть во время программирования на PHP 5.

Что требуется для использования данной книги

Как уже было сказано, PHP может работать на многих различных операционных системах, включая Windows, Linux, Mac OS X и др. Следовательно, первое, что необходимо сделать читателю, это получить версию PHP, которая подходит для исполь-

зуемой операционной системы. (Полные инструкции по загрузке и установке корректной версии PHP 5 представлены в главе 1.)

Для создания и редактирования сценариев потребуется текстовый редактор.

Кроме того, потребуется Web-сервер. Хорошим выбором всегда будет Apache, особенно на Unix-машинах (хотя он также хорошо работает и в Windows). Apache включается в состав большинства дистрибутивов Linux, а также в Mac OS X. Загрузить последнюю версию этого Web-сервера можно с сайта www.apache.org. Кроме всех прочих своих достоинств, Apache бесплатен.

Некоторые Windows-пользователи предпочитают работать с собственным Web-сервером Microsoft, IIS (Internet Information Server), который включается в Windows 2000 и в XP-версии (кроме Home). Однако он не обязательно устанавливается по умолчанию. В случае Windows 98 можно воспользоваться усеченной версией IIS, которая называется PWS (Personal Web Server), она идеально подходит для небольших проектов. Однако если предполагается создавать большие проекты, то Apache будет, вероятно, наилучшим выбором в долгосрочной перспективе, а в случае Windows ME или XP Home (ни та ни другая версия не поддерживают ни PWS ни IIS), Apache будет единственно возможным решением.

Использование командной строки

Большинство пользователей Windows или Macintosh не знакомы с интерфейсом командной строки, или *shell*, — мощным инструментом для взаимодействия с системой. Для многих примеров в данной книге он является основным.

До того как графические среды получили широкое распространение, а об интерфейсах drag-and-drop фактически еще никто не слышал, единственным способом взаимодействия с компьютерами был ввод команд с клавиатуры по одной строке за раз. Пиктограммы отсутствовали, поэтому для запуска какой-либо программы приходилось вводить ее имя.

Интерфейс командной строки до сих пор используется во многих программах. Почему? С одной стороны, так проще писать программы, и даже многие современные мощные утилиты и приложения написаны исключительно для использования посредством командной строки. С другой стороны, многие пользователи до сих пор предпочитают взаимодействовать с командной строкой, чем с оконной средой и мышью. (Например, в главе 9 при изучении баз данных рассматривается диспетчер MySQL, который конфигурируется путем ввода инструкций в командной строке.)

Чтобы получить доступ к командной строке, необходимо:

- ❑ в Windows нажать кнопку Пуск и найти в меню (обычно в категории Программы⇒Стандартные) пункт Командная строка или Сеанс MS-DOS или, воспользовавшись комбинацией клавиш <Windows+R>, вызвать диалоговое окно Выполнить, ввести в нем команду cmd и нажать ОК;
- ❑ в Unix (включая такие варианты, как Linux и Mac OS X) найти программу с именем console, terminal, konsole, xterm, eterm или kterm; все они являются широко используемыми shell-программами, которые можно найти на многих Unix-системах.

После того как интерфейс командной строки вызван, на экране появится короткий текст, варианты которого показаны ниже:


```
$
%
C: />
#
bash$
```

Это *приглашение командной строки*, информирующее пользователя о том, что интерфейс готов к получению инструкций — фактически это приглашение для ввода команд. То, как выглядит приглашение, не имеет значения. В данной книге приглашение командной строки обозначается так:

>

Необходимые для ввода в командную строку инструкции показаны сразу после приглашения (>), главным образом в первой строке. Вывод, сгенерированный компьютером, печатается ниже:

```
> mysqlshow
+-----+
| Databases |
+-----+
| mysql    |
| test     |
+-----+
```

Соглашения

В книге использовано несколько соглашений по оформлению текста.

В блоках, подобных этому, содержится важная, относящаяся к рассматриваемому материалу информация, которую следует запомнить.

Примечания, советы и отступления от темы печатаются с отступом и выделяются курсивом.

Что касается стилей текста:

- ☐ важные слова *выделяются курсивом* при первом упоминании;
- ☐ клавиатурные комбинации показываются так: <Ctrl+A>;
- ☐ имена файлов, URL-адреса и код в тексте выделяются так: `persistence.properties`;
- ☐ код выделяется двумя различными способами:

В примерах новый и важный код выделяется серым фоном.

Серый фон не используется для кода, который менее важен в текущем контексте или уже был показан ранее.

Исходный код

В ходе работы с представленными в книге примерами код можно вводить вручную или использовать файлы исходного кода, сопровождающие книгу. Весь использованный в книге исходный код доступен для загрузки с сайта www.wrox.com (русифицированные примеры доступны на сайте www.williamspublishing.com).

На этом сайте найдите название книги (воспользовавшись поисковой формой либо используя списки названий) и щелкните на ссылке Download Code, которая расположена на странице с описанием книги.

Поскольку многие книги имеют похожие названия, проще всего искать книги по ISBN-номерам; английский ISBN данной книги — 0-7645-5783-1.

После загрузки кода его необходимо разархивировать с помощью какого-либо архиватора. Кроме того, можно посетить страницу загрузки кода на сайте Wrox www.wrox.com/dynamic/books/download.aspx и найти доступный код для данной книги, а также для других книг Wrox.

Ошибки

Сообщество Wrox прилагает все усилия, чтобы полностью исключить ошибки в тексте или коде. И все же ошибки случаются. Поэтому авторы будут очень признательны за сообщения об ошибках, например, опечатках или неверных блоках кода. Отправляя описание ошибки, вы сэкономите время многих других читателей и в то же время поможете издателям предоставить более качественную информацию.

Чтобы найти перечень ошибок по данной книге, необходимо зайти на сайт www.wrox.com и с помощью поиска или по списку названий найти эту книгу. Затем на странице описания книги следует щелкнуть на ссылке "Errata". На этой странице можно просмотреть перечень ошибок в данной книге, которые были опубликованы редакторами Wrox. Полный список книг, включая ссылки на страницы ошибок для каждой книги, доступен на странице www.wrox.com/misc-pages/booklist.shtml.

Если найденная вами ошибка отсутствует в соответствующем перечне, то ее можно отправить, заполнив форму на странице www.wrox.com/contact/techsupport.shtml. Редакторы проверяют эту информацию и в случае необходимости опубликуют соответствующее сообщение на странице ошибок, а также исправят проблему в последующих изданиях книги.

p2p.wrox.com

Для обсуждения книг можно зарегистрироваться в P2P-формах на сайте p2p.wrox.com. Форумы представляют собой Web-систему для опубликования сообщений, касающихся книг Wrox и связанных технологий. На форумах предлагается подписка на e-mail-рассылку по интересующим вас темам, здесь же вы можете пообщаться с авторами книг Wrox, редакторами, экспертами, а также другими читателями.

На странице <http://p2p.wrox.com> можно найти несколько различных форумов, которые способны помочь посетителям не только как читателям, но и как разработчикам приложений. Чтобы присоединиться к форуму, необходимо выполнить несколько шагов.

1. Зайти на сайт p2p.wrox.com и щелкнуть на ссылке "Register Now" (Зарегистрироваться).
2. Прочитать условия использования и щелкнуть на кнопке "Agree" (Согласен).

3. Заполнить форму необходимой для регистрации информацией, а также дополнительными сведениями, которые вы желаете предоставить, и нажать кнопку "Submit" (Отправить).
4. В ответ будет отправлено письмо с указаниями, как проверить регистрационную запись и завершить процесс регистрации.

Читать сообщения можно, не регистрируясь в форуме; регистрация требуется только для публикации сообщений.

После регистрации можно отправлять новые сообщения и отвечать на сообщения других пользователей. Читать сообщения в Web можно в любое время. Чтобы получать новые сообщения с определенного форума на e-mail-адрес, необходимо щелкнуть на пиктограмме "Subscribe to this Forum" (Подключиться к форуму).

Более подробную информацию об использовании форумов Wrox P2P, о работе программного обеспечения форума, а также ответы на многие другие вопросы вы найдете в списке часто задаваемых вопросов (ссылка FAQ на любой из P2P-страниц).

От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо, либо просто посетить наш Web-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг. Наши координаты:

E-mail: info@dialektika.com

WWW: <http://www.dialektika.com>

Адреса для писем из:

России: 115419, Москва, а/я 783

Украины: 03150, Киев, а/я 152

1

Установка и использование PHP

PHP, или HyperText Preprocessor, широко используется для создания программируемых функций web-сайтов, поскольку этот язык прост в изучении, а также потому, что его синтаксис скопирован с других широко используемых языков и понятен многим программистам. В данной главе вкратце представлена история развития PHP, а затем обсуждается природа этого языка применительно к web-среде.

Прежде чем углубляться в подробности программирования на PHP 5, необходимо четко представлять себе принципы работы PHP-программ в Web, а это в свою очередь подразумевает знание Web-протокола, который называется *протокол передачи гипертекста* (HyperText Transfer Protocol — HTTP). HTTP представляет собой язык или формат для обмена данными между браузером и Web-сервером и, следовательно, является существенным для многих аспектов работы PHP. В данной главе представлен краткий обзор HTTP; подробнее этот протокол рассматривается в главе 2.

В данной главе также описана правильная установка PHP на Linux- и на Windows-сервер. PHP-программы работают в сопряжении с Web-страницами, предоставляемыми программным обеспечением Web-сервера (например, Apache или IIS), который в свою очередь работает в операционной системе (такой как Linux или Windows). Для того чтобы создавать хорошие PHP-программы, совсем необязательно знать все о сетевых операционных системах. Однако многие аспекты PHP контролируются или находятся под влиянием Web-сервера. Если читатель недостаточно знаком с серверными компьютерами и программным обеспечением Web-серверов, то волноваться не стоит, поскольку ниже рассматривается их работа, а также технические требования и процесс установки базового программного обеспечения Web-сервера.

В данной главе рассматривается установка PHP на машину под управлением Red Hat Linux с Web-сервером Apache, а также установка PHP на компьютер с Windows 2000 и Web-сервером IIS.

Кроме того, в данной главе рассматривается содержимое конфигурационного файла PHP, `php.ini`, а также показано, как проверить правильность установки PHP. Итак, начнем.

Истоки PHP

Язык программирования PHP предназначен для работы с HTML-кодом, но в отличие от HTML, PHP обладает возможностями обработки данных. Читатели, знакомые с HTML, знают, что данный язык в действительности является не языком программирования, а скорее языком визуализации, т.е. HTML позволяет писать Web-страницы с помощью кода, создающего в окне браузера приятное (будем надеяться) отображение текста, графики и гиперссылок. И хотя в HTML имеется несколько полезных функций (таких как возможность инициировать передачу форм), в основном HTML не позволяет программировать. Например, не существует HTML-команд, которые позволяют сложить два числа или получить доступ к базе данных.

Те читатели, которые помнят зарождение Web в начале 90-х годов, могут также вспомнить, что ранние Web-страницы состояли из HTML-кода, написанного в виде простых текстовых файлов. Когда клиентский браузер подключался к Web-сайту, программное обеспечение Web-сервера отправляло браузеру эти HTML-файлы, состоящие из обычного текста, для обработки и отображения. Браузер фактически выполнял визуализацию (то же самое он делает и теперь), но при выборе пункта меню “Вид⇒Просмотр HTML-кода” можно было просмотреть чистый HTML-код.

JavaScript и несколько других почти неизвестных языков программирования улучшили ситуацию для Web-дизайнеров, обеспечивая программируемую функциональность внутри Web-страниц. Однако в JavaScript эта функциональность обеспечивалась только на пользовательском компьютере, а не на Web-сервере, где действительно выполняется важнейшая обработка информации и доступ к базам данных. Язык PERL (Practical Extraction and Reporting Language — практичный язык для извлечения текстов и генерации отчетов) был одним из первых широко применяемых языков для программирования серверной части Web-сайтов, но имел собственные ограничения; например, его невозможно внедрить в HTML-код с целью простого программирования отдельных страниц.

Язык PHP (вначале он назывался PHP/FI) был создан в 1995 году Расмусом Лердорфом (Rasmus Lerdorf) из нескольких Perl-сценариев, которые он разработал для отслеживания посещаемости своего резюме, опубликованного на Web-странице. Со временем Расмус написал реализацию данного языка на C и опубликовал исходный код для широкой общественности, а в начале 1998 года появилась версия PHP 3.0 (ее написали Расмус Лердорф, Энди Гутманс (Andi Gutmans) и Зив Сураски (Zeev Suraski)). Это была первая версия, очень похожая на нынешние выпуски PHP.

Главная цель PHP заключается в том, чтобы позволить программисту легко создавать динамические Web-страницы. От статических Web-страниц динамические отличаются тем, что содержимое и структура последних может изменяться при каждом доступе к ним (именно для этого и требуется серверное программирование), тогда как содержимое и структура статических Web-страниц фиксировано и не изменяется до тех пор, пока дизайнер не изменит их вручную.

В отличие от многих других языков PHP-код можно внедрять непосредственно в HTML-код, таким образом значительно облегчая добавление в Web-страницы серверной, программируемой функциональности. Именно эта возможность является одной из

важнейших составляющих гибкости языка PHP и как следствие этого его популярности. Вместе с тем нет никаких сомнений, что PHP развивается в гораздо более полнофункциональный язык, который выходит далеко за рамки первоначальных замыслов его авторов. PHP стремится стать основным языком для широкого множества сетевых и сетевых приложений, пятая версия PHP демонстрирует все признаки этого развития.

Не стоит забывать об эффективной работе PHP с HTTP (протоколом передачи гипертекста), давно согласованным протоколом (или форматом) обмена данными в Web-среде. Всякий раз, когда пользователь щелкает по гиперссылке или вводит Web-адрес в браузере, запрос в HTTP-формате отправляется Web-серверу, который отправляет в ответ Web-страницу. Если затребованной страницы не существует, пользователь получает ответ "404 Not Found" (файл не найден). Отправка корректного ответа или сообщения об ошибке в случае, если страница не найдена, — все это функции протокола HTTP. В главе 2 протокол HTTP рассматривается очень подробно, поскольку от него зависит несколько важнейших аспектов работы PHP.

Установка, конфигурирование и запуск PHP

Чтобы написать PHP-приложение, работающее с Web-страницами, необходимо установить и сконфигурировать PHP. Так как в данной книге рассматривается создание Web-приложений, для полного понимания изложенного в ней материала читателю, безусловно, потребуется Web-сервер и несколько Web-страниц (краткое введение в HTML представлено в главе 3, однако здесь предполагается, что читатель уже знает, как создавать простейшие Web-страницы). Также потребуется загрузить, установить и настроить PHP, в последующих разделах представлены исчерпывающие инструкции по этим вопросам. Следует отметить, что некоторые конфигурационные параметры PHP связаны с требованиями весьма специфических приложений (о них не стоит беспокоиться без реальной на то причины), поэтому многие параметры описываются в соответствующих главах данной книги.

Системные требования

Для выполнения кода, представленного на страницах этой книги, потребуется как минимум следующее программное обеспечение:

- ☐ серверная операционная система, например, Windows 2000 или Linux;
- ☐ PHP-совместимый Web-сервер (например, Apache или Internet Information Server (IIS));
- ☐ PHP 5 (его можно загрузить с сайта www.php.net);
- ☐ система управления реляционными базами данных (начиная с главы 9 в данной книге рассматривается MySQL или SQLite);
- ☐ Web-браузер (например, IE, Mozilla и т.п.);
- ☐ текстовый редактор, такой как Notepad, Emacs, vi, BBEdit и др.

Если используется не очень старая и не перегруженная система, то беспокоиться о свободном месте на жестком диске или оперативной памяти не стоит. PHP нетребователен к системным ресурсам и работает весьма эффективно.

В целях разработки PHP-сценариев все перечисленное здесь программное обеспечение можно установить на один компьютер. Если же есть доступ к нескольким компьютерам, подключенным к сети, то можно установить все серверное программное обеспечение на один из них (обычно либо на Unix-машину, либо на компьютер с Windows NT/2000), а в качестве клиентской машины использовать другой компьютер. В данной книге, как правило, предполагается, что все программное обеспечение работает на одном компьютере, поскольку такая конфигурация используется большинством Web-разработчиков.

Конфигурационный файл PHP, `php.ini`

В дистрибутиве PHP имеется два примера конфигурационного файла: `php.ini-dist` и `php.ini-recommended`. После загрузки и установки PHP в системе будет находиться один файл с именем `php.ini`, и при каждом запуске PHP будет считывать этот файл и настраиваться согласно заданным в нем параметрам. Файл `php.ini` можно написать вручную, но, естественно, большинство пользователей PHP просто модифицируют под свои нужды либо `dist`-, либо `recommended`-файл, а затем копируют его в соответствующий каталог и переименовывают.

Однако необходимо обратить внимание на следующие строки `dist`-файла:

```
; Это стандартный файл для новых инсталляций PHP.  
; По умолчанию PHP устанавливается с конфигурацией подходящей для  
; целей разработки, а *НЕ* для использования на работающих сайтах.
```

Почти для всех примеров в данной книге используются настройки `dist`-файла, а все изменения настроек в случае необходимости оговариваются особо. После завершения разработки приложения следует использовать `recommended`-файл и скопировать его на реальный сервер. Необходимо помнить, что может понадобиться некоторое изменение кода для обеспечения корректной работы созданного приложения с настройками `recommended`-файла. Этот момент далее освещается более подробно.

Настройка тестовой машины

В данной главе рассматривается установка PHP 5 на машину под управлением Red Hat Linux с Web-сервером Apache, а также на машину с Windows 2000 и Web-сервером IIS. PHP 5 способен работать на многих других операционных системах и Web-серверах, поэтому в документации на PHP имеются сведения о его установке и конфигурировании на других платформах. Кроме того, имеется множество возможных методов установки. Например, для Windows-версии существует автоматический инсталлятор, тогда как в Linux (в некоторых версиях) можно использовать RPM-пакеты, а также при желании можно загрузить оригинальный исходный код и скомпилировать PHP из него. В любом случае установка не вызывает трудностей, если четко придерживаться необходимой процедуры установки. Примеры, представленные здесь, могут послужить хорошей отправной точкой для многих вариантов установки.

Существует несколько инсталляторов от сторонних производителей (часто с открытым исходным кодом и бесплатных). Например, можно попробовать поискать в Google `PHPTriad` или `Foxserv`.

Сетевые соединения

Для того чтобы Web-сервер работал, компьютер, на котором он установлен, не обязательно должен быть подключен к Internet или к локальной сети. К Web-серверу, установленному на компьютере, всегда можно получить доступ посредством Web-браузера, работающего на той же самой машине, даже если она не имеет сетевой платы или модема. Конечно, для загрузки и установки необходимого программного обеспечения требуется доступ к Internet-соединению. Однако чтобы Web-сервер работал, наличие такого соединения совсем необязательно.

После установки и запуска Web-сервера можно установить PHP 5. Существует несколько конфигурационных параметров Web-сервера, которые определяют, как в нем будут работать PHP-программы; настройка этих параметров рассматривается далее. В большинстве дистрибутивов PHP имеются автоматические инсталляторы; в данной книге главным образом рассматривается установка и настройка вручную — это позволяет полнее проиллюстрировать все происходящие в ходе процесса установки события.

Что делать, если что-то не так? В большинстве дистрибутивов PHP присутствуют файлы README и INSTALL. В данных файлах, а также в PHP-руководстве на странице www.php.net/manual/ представлена исчерпывающая информация, которая может быть более актуальной, чем представленная здесь (в книге рассматривается версия 5.0.2 PHP).

С чего начать?

Ниже описаны два основных варианта установки, каждый из которых зависит от используемой операционной системы:

- ☐ установка PHP с Web-сервером Apache на Linux (рассматривается Red Hat Fedora Linux);
- ☐ установка PHP 5 с Microsoft Internet Information Server на Windows (в данном случае используется Windows 2000).

PHP 5 можно установить на многие другие комбинации Web-сервер/операционная система, включая, например, Apache и Windows. Рассматриваемые в данной книге операционные системы являются самыми простыми для того, чтобы приступить к работе. Если ни одна из них не подходит, то, конечно, можно установить PHP на любую другую платформу — все примеры в книге должны работать на всех системах. Более полные инструкции по установке представлены в руководстве по PHP 5.

Использование PHP 5

Во время инсталляции PHP 5 на Web-сервер необходимо решить, как именно будет работать PHP: либо как CGI-программа, либо как отдельный статический или динамический модуль. CGI (Common Gateway Interface — интерфейс общего шлюза) представляется весьма полезным способом работы таких интерпретаторов, как PHP 5. Из-за риска, связанного с безопасностью сценариев (см. раздел “Использование PHP как CGI-программы” далее в настоящей главе), в большинстве случаев рекомендуется компиляция PHP 5 как статического или динамического модуля. В данной книге рассматривается установка (на Linux и Windows) PHP как отдельного SAPI-модуля (Server

Application Programming Interface — интерфейс разработки серверных приложений). На Windows для запуска PHP как SAPI-модуля используется ISAPI-фильтр.

Хотя чаще всего PHP работает совместно с Web-сервером, с тем чтобы Web-страницы, представленные файлами с расширением .php, перед отправкой браузеру обрабатывались PHP-интерпретатором, существует также утилита командной строки, которая позволяет запускать PHP-код из командной строки. Она доступна в любой рассматриваемой здесь инсталляции. Большой объем документации по этой теме представлен на сайте PHP (www.php.net).

Удовлетворительное создание и выполнение Web-приложений предполагает наличие доступа к Web-серверу, на котором установлен (или может быть установлен) PHP, к тому же инсталляция должна быть протестирована и работать должным образом. Кроме того, предполагается, что PHP сконфигурирован (или может быть сконфигурирован) для поддержки различных потребностей PHP-программ. Эти требования удовлетворены в двух описанных ниже ситуациях.

- ☐ Разработчик использует настольный компьютер или серверную машину, операционную систему и Web-сервер, совместимый с PHP, а PHP установлен и сконфигурирован.
- ☐ Разработчик использует настольную или серверную машину, подключенную к Internet, с доступом к учетной записи на Web-хостинговом сервере, на котором установлен и сконфигурирован PHP.

На подавляющем большинстве настольных компьютеров работает операционная система Windows версий 98, NT, 2000, 2003 или XP. Во многих случаях можно получить бесплатную копию персонального Web-сервера (Personal Web Server — PWS) и установить его на машине с одной из этих операционных систем. PHP совместим с PWS, поэтому можно устанавливать и настраивать PHP на настольной машине, работающей под управлением простейшей операционной системы, например, Windows 98. Серверные операционные системы, такие как Windows NT, 2000 и 2003, поставляются с Web-сервером IIS. PHP совместим с ним и может устанавливаться и конфигурироваться на таких машинах. В рассматриваемой здесь инсталляции Windows 2000 PHP 5 использует в качестве Web-сервера IIS.

Большинство Web-хостинговых компьютеров работают с одной из версий Linux, например, Debian или RedHat, с FreeBSD либо с какими-нибудь другими клонами Unix. На таких машинах предпочтительным Web-сервером является Apache. PHP совместим с Linux и Apache, поэтому PHP можно устанавливать и конфигурировать на таких машинах, однако если разработчик не владеет Web-хостинговым компьютером (часто так оно и бывает), то, скорее всего, контролировать установку и настройку PHP он не сможет. В такой ситуации (например, при работе над существующим Web-сайтом, размещенным на чужом сервере) для устранения проблем, возникающих при разработке PHP-программ, можно просто получить информацию об операционной системе, программном обеспечении Web-сервера и версии PHP.

Установка PHP на Linux и Apache

Во время написания этой книги была доступна самая первая версия PHP 5, которая и рассматривается здесь. При необходимости читатель может посетить сайт PHP и получить более свежую версию, а также узнать о внесенных изменениях.

Комбинация Linux, Apache, MySQL и PHP — наиболее распространенная промышленная среда для работы PHP-совместимых Web-серверов. Часто эту комбинацию программного обеспечения с открытым исходным кодом обозначают аббревиатурой LAMP. Применение LAMP-связки позволяет воспользоваться опытом многих других разработчиков, использующих эту схему.

Разработчики PHP работают в очень тесном сотрудничестве с коллективами разработки Apache и MySQL, это гарантирует, что передовые функции в одной из трех данных серверных систем полностью поддерживаются двумя другими компонентами. Однако к моменту написания книги PHP 5 распространялся с поддержкой SQLite, а не MySQL. Это связано с тем, что судьба MySQL как продукта с открытым исходным кодом в то время была неясной. Поэтому при разработке PHP-сценариев на эту проблему все же стоит обратить внимание.

Выбор метода установки

Как и в случае других программных продуктов с открытым исходным кодом, пользователь может загрузить исходный код PHP и Apache (в обоих случаях это код, написанный на языке программирования C) и скомпилировать данные программы самостоятельно. Можно поступить и по-другому: получить заранее скомпилированные версии в одной из двух форм. Первая форма — бинарные файлы, представляющие собой заранее скомпилированные версии программного обеспечения, обычно поставляемые с установочными сценариями, которые помещают все необходимые компоненты в соответствующие каталоги файловой системы. Вторая форма — простейшие в установке бинарные пакеты, доступные для систем, имеющих средство управления пакетами, такое как, например, RPM (Red Hat Package Manager — менеджер пакетов Red Hat) для Linux.

Далее приведен краткий обзор всех трех методов.

<i>Метод установки</i>	<i>Преимущества</i>	<i>Недостатки</i>
Исходный код	Наиболее гибкое решение для нестандартной установки. В дистрибутив включаются дополнительные тесты и примеры	Необходимость компиляции, что несколько труднее, чем использование двух других вариантов. Установленные таким образом программы сложнее удалять
Бинарные (скомпилированные) файлы	Нет необходимости разбираться с компиляцией. Требуют меньше времени для установки	Меньшая гибкость, чем в случае установки из исходных кодов
Бинарные RPM-пакеты	Самый простой и быстрый метод установки, упрощающий деинсталляцию и последующее обновление программ	Необходимо использовать RPM-совместимый дистрибутив Linux, например, Red Hat. Наименее гибкий метод инсталляции

RPM-установка PHP 4

Версия Red Hat, которая использовалась авторами данной книги, фактически называется Fedora, так как компания Red Hat разделила разработку на две ветви: Fedora и Enterprise Red Hat Linux Server. Во время написания книги сайт Fedora не имел RPM-пакетов для PHP 5, поэтому в этом разделе представлены инструкции по получению и установке RPM-пакетов для PHP 4, а также по загрузке и компиляции PHP 5 для Fedora. К моменту публикации книги, скорее всего, появится RPM-пакет PHP 5

для многих дистрибутивов Linux, поэтому описанная здесь RPM-инсталляция может послужить хорошим руководством по установке PHP 5 RPM-методом.

Менеджер пакетов Red Hat используется во многих популярных Linux-дистрибутивах, включая Red Hat, SuSE, Mandrake, Definite, TurboLinux, Caldera и Yellow Dog. Если читатель использует другую систему управления пакетами, например, пакеты Debian, то инструкции по установке следует искать в руководстве по используемому дистрибутиву.

Получение RPM-пакетов

Лучше всего воспользоваться RPM-пакетами с инсталляционных дисков используемого дистрибутива Linux. Например, в дистрибутивы Red Hat 7 и SuSE 7 включен PHP 4 (хотя он не устанавливается по умолчанию) — к тому моменту, когда вы будете читать эту книгу, PHP, скорее всего, будет включен в самые свежие версии этих дистрибутивов.

Если в состав используемого дистрибутива PHP 4 не включен или в этом дистрибутиве отсутствуют необходимые функции или поддержка RPM-пакетов, то следующим источником RPM-пакетов можно считать Web-сайт производителя данного дистрибутива. На таком сайте должен присутствовать раздел загрузки или ссылка на FTP-узел, с которого можно загрузить самые свежие RPM-пакеты.

Наконец, на сайте www.rpmfind.net имеется мощная служба поиска RPM-пакетов. Прежде чем загружать RPM-пакеты, следует убедиться, что они совместимы с используемым дистрибутивом Linux и аппаратным обеспечением компьютера, на который они будут устанавливаться. Различные дистрибутивы помещают важные файлы в разные каталоги, и это может привести к тому, что RPM-пакеты одного производителя не будут работать на системах, собранных другими производителями дистрибутивов. Большинство RPM-пакетов скомпилированы так, чтобы работать на различном аппаратном обеспечении, поддерживаемом Linux. В приведенной ниже таблице показаны наиболее распространенные аббревиатуры, используемые в именах RPM-пакетов (аббревиатуры потребуются для поиска на сайте [rpmfind](http://rpmfind.net)).

Аббревиатура	Совместимость
i386	PC-компьютеры на основе процессоров Intel или на основе полностью совместимых с ними процессоров: Intel 80386, 486, Pentium, Pentium II, Pentium III и Celeron; AMD 5x86, K-серия, а также Athlon; Cyrix 6x86
i586	PC-компьютеры на основе процессоров Intel или на основе полностью совместимых с ними процессоров: Intel Pentium II, III и Celeron; AMD K-серии и Athlon; Cyrix 6x86
PPC	Компьютеры на базе микросхем Motorola PowerPC (и совместимых с ними), например, Power Mac корпорации Apple, G3, G4 и iMac. Однако использовать RPM можно только на аппаратуре Macintosh с установленной операционной Linux
alpha	Серверы и рабочие станции, на основе 64-битовых процессоров Compaq Digital Alpha
sparc	Серверы и рабочие станции с процессорами 64-битовой архитектуры SPARC, такие как UltraSPARC корпорации Sun Microsystems
m68k	Компьютеры, основанные на старых процессорах Motorola серии 68000, такие как Amiga и ранние Apple Macintosh, для которых существуют различные версии Linux

Тем пользователям, которые хотят использовать графический инсталлятор, поставляемый с их дистрибутивом, рекомендуется обратиться к руководству этого дистрибутива. Так как графические средства инсталляции весьма различаются в разных дистрибутивах, здесь эти средства не рассматриваются. Вместе с тем, управлять любой RPM-системой можно с помощью командной строки; далее показано, как устанавливать необходимые компоненты, используя интерфейс командной строки.

Какие RPM-пакеты потребуются?

Для установки PHP понадобятся следующие RPM-пакеты:

- ☐ zlib
- ☐ libpng
- ☐ libjpeg
- ☐ gd
- ☐ gd-devel
- ☐ apache
- ☐ mod_php4

Чтобы выяснить, какие из них уже установлены в системе, введите в командной строке следующую команду, последовательно подставляя необходимые имена:

```
> rpm -q zlib
zlib-1.1.3-6-i386
> rpm -q libpng
Package libpng is not installed
(Пакет libpng не установлен)
```

Если пакет установлен, то система выведет определенную строку, в противном случае будет напечатано соответствующее сообщение об ошибке. В строке фактически содержится версия программы, установленной с помощью этого пакета (в данном случае 1.1.3), номер выпуска самого пакета (шестой общедоступный выпуск), а также тип архитектуры, для которой был скомпилирован RPM-пакет (в данном случае совместимая с Intel 386, поскольку использовался компьютер Pentium III).

Обратите внимание на уже имеющиеся пакеты и версии соответствующих программ (номер версии более важен, чем номер выпуска).

Подойдет Apache версии 1.3.29, если желательно использовать старые версии библиотеки GD; для использования последней версии данной библиотеки хорошим выбором будет Apache версии 2.0.48. Библиотека GD теперь поставляется вместе с PHP 5 и работает с Apache 2.0.17.

Затем следует найти подходящие новые версии всех остальных пакетов, которых еще нет в системе, или использовать их старые версии. Как уже отмечалось, пакеты следует искать на установочных дисках дистрибутива, Web-сайте производителя, а также на сайте www.rpmfind.net.

После того как текущие версии всех пакетов найдены, можно устанавливать эти пакеты. Для обновления имеющейся инсталляции используется такая же команда, как и для новой установки. Необходимо перейти в каталог с RPM-пакетами (на CD или жестком диске) и ввести следующую команду от имени root:

```
> rpm -Uh libpng-1.0.5-3-i386.rpm
#####
```

Для каждого пакета, который необходимо обновить или установить, следует подставить имя файла в данной команде. Строка знаков # отражает на экране процесс установки.

Если установка производится так, как описано выше, то все требуемые файлы, скорее всего, установлены в необходимом порядке.

Компиляция PHP 5 из исходного кода

В данном случае в качестве метода установки PHP на Red Hat Fedora с Apache рассматривается компиляция исходного кода PHP. Для этого можно использовать командную строку и некоторые визуальные инструменты (например, Konqueror), входящие в состав дистрибутива Red Hat Linux. В случае использования какого-либо графического пользовательского интерфейса Linux (например, KDE) активизировать интерфейс командной строки можно, нажав на кнопку **Red Hat**, а затем выбрав пункты меню **System Tools**⇒**Terminal**. Окно терминала показано на рис. 1.1.

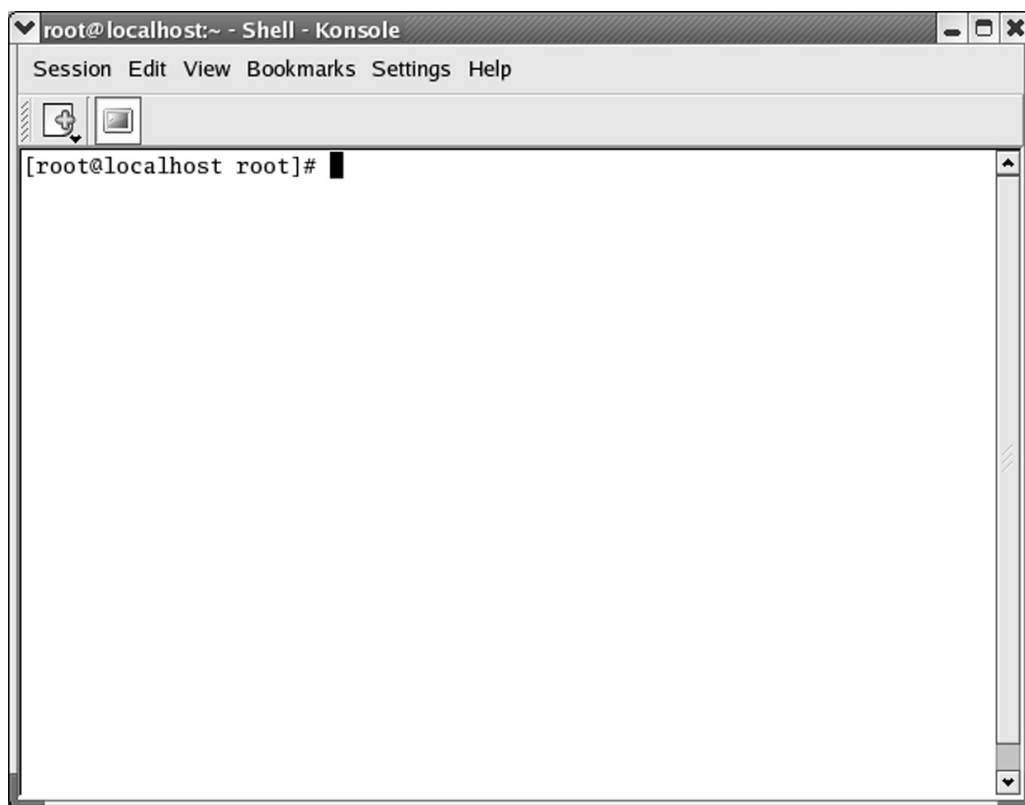


Рис. 1.1.

Для инсталляции PHP понадобится установленный компилятор (компилятор ANSI C). Иногда такой компилятор устанавливается как часть инсталляции Linux, однако если в системе нет такого компилятора, то на сайте www.gnu.org можно найти и получить

хороший (и бесплатный) компилятор, который называется `gnugcc`. На рис. 1.2 показан Web-сайт проекта GNU, а на рис. 1.3 — фрагмент документации для GCC.

После того как компилятор установлен, можно загружать исходный файл с сайта `www.php.net`. Данный файл представляет собой `tar`-архив, сжатый `gzip`, поэтому его необходимо распаковать. Также можно загрузить файл `.bz2`, однако потребуется только один из этих файлов — либо `gzip`, либо `.bz2`.

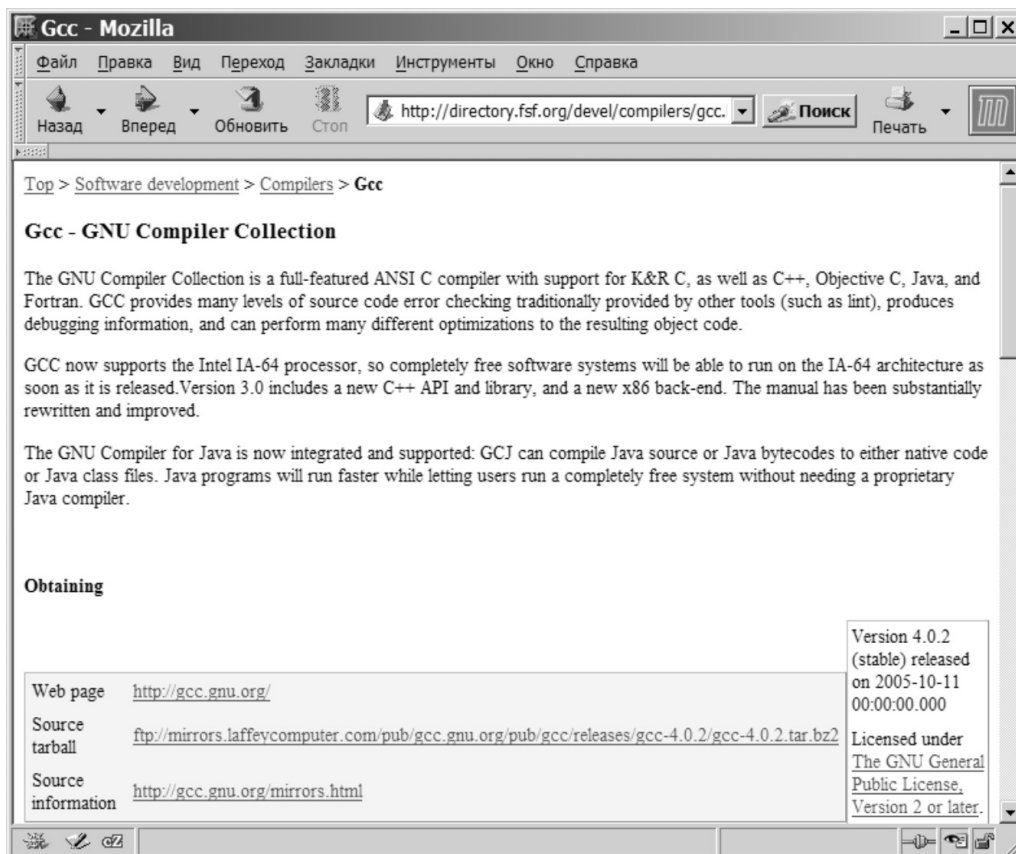


Рис. 1.2.

Чтобы просмотреть содержимое сжатого файла, можно использовать файловый менеджер Konquerer. На рис. 1.4 показано содержимое `tar`-файла.

Кроме того, Konquerer можно использовать для непосредственного копирования всех файлов, входящих в сжатый архив, в другой каталог, однако это впоследствии приведет к тому, что процесс компиляции по непонятным причинам аварийно завершится (будут появляться странные сообщения, которые не помогут выявить, в чем проблема). Вместо этого для разархивирования файлов лучше использовать следующую команду в терминале (рис. 1.5):

```
tar -xvzf php-5.0(остальная часть номера версии).tar.gz
```

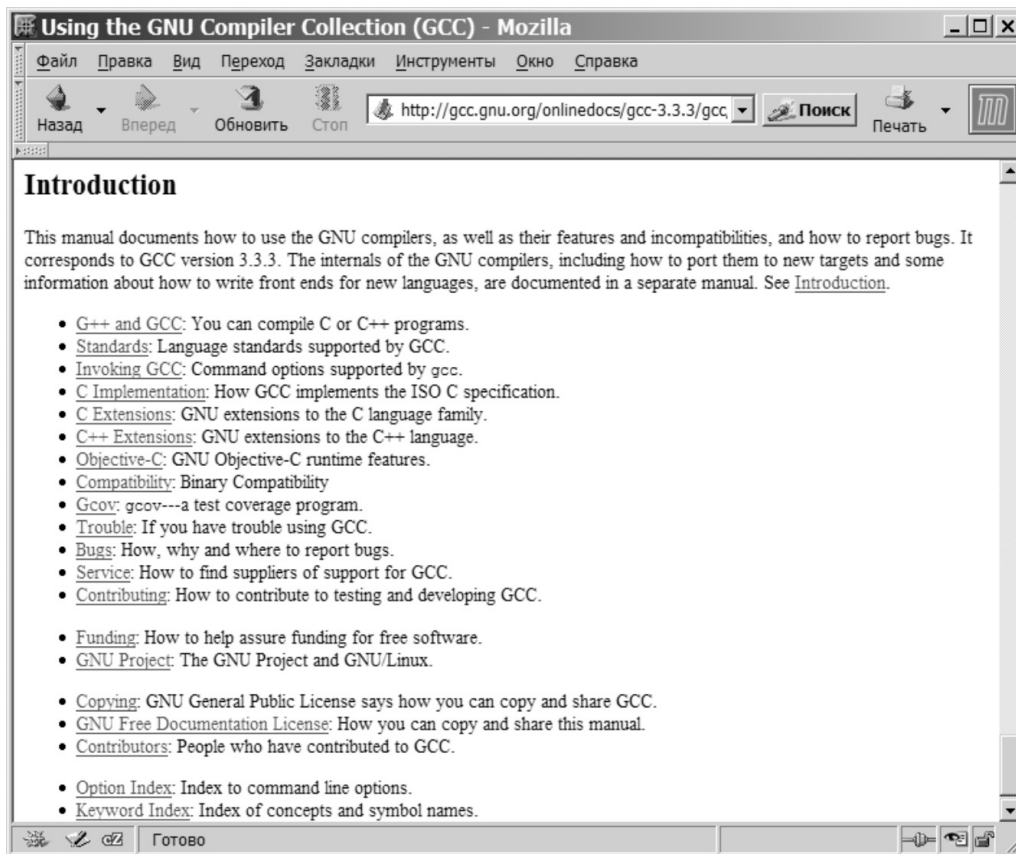


Рис. 1.3.

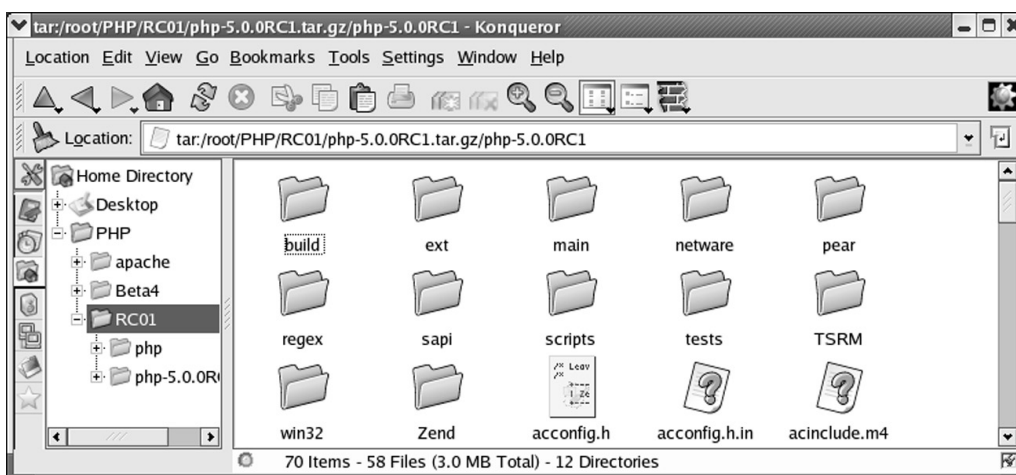
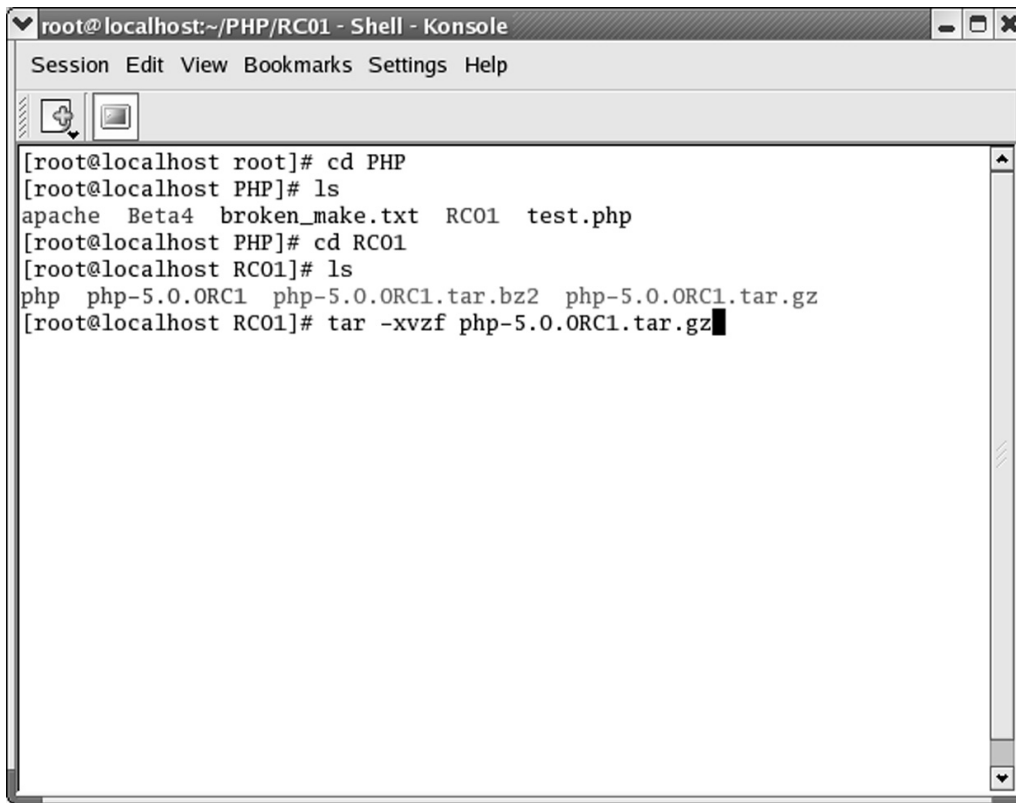


Рис. 1.4.



```
root@localhost:~/PHP/RC01 - Shell - Konsole
Session Edit View Bookmarks Settings Help

[root@localhost root]# cd PHP
[root@localhost PHP]# ls
apache Beta4 broken_make.txt RC01 test.php
[root@localhost PHP]# cd RC01
[root@localhost RC01]# ls
php  php-5.0.0RC1  php-5.0.0RC1.tar.bz2  php-5.0.0RC1.tar.gz
[root@localhost RC01]# tar -xvzf php-5.0.0RC1.tar.gz
```

Рис. 1.5.

Для перехода в каталог дистрибутива PHP 5 следует использовать команду `cd`:

```
cd php-5.0 (остальная часть номера версии)
```

После перехода в каталог `php-5.0.0RC1` на экране появится много новых каталогов и файлов. Подробные инструкции, связанные с установкой, находятся в текстовом файле `INSTALL` (рис. 1.6).

Для целей данной книги PHP устанавливается как динамический общий объект (Dynamic Shared Object — DSO), так чтобы не требовалось перекомпилировать сервер Apache.

Последние версии Apache поддерживают DSO-объекты, кроме того, общие объекты могут использоваться другими программами, такими как PostgreSQL. Компилировать PHP 5 как статический модуль не рекомендуется, хотя это и возможно. Если PHP статически связан, например, с Apache или PostgreSQL, каждую из этих программ придется перекомпилировать, прежде чем они смогут взаимодействовать с PHP. В общих объектах (DSO) можно легко изменять конфигурационные файлы программ без их перекомпиляции.

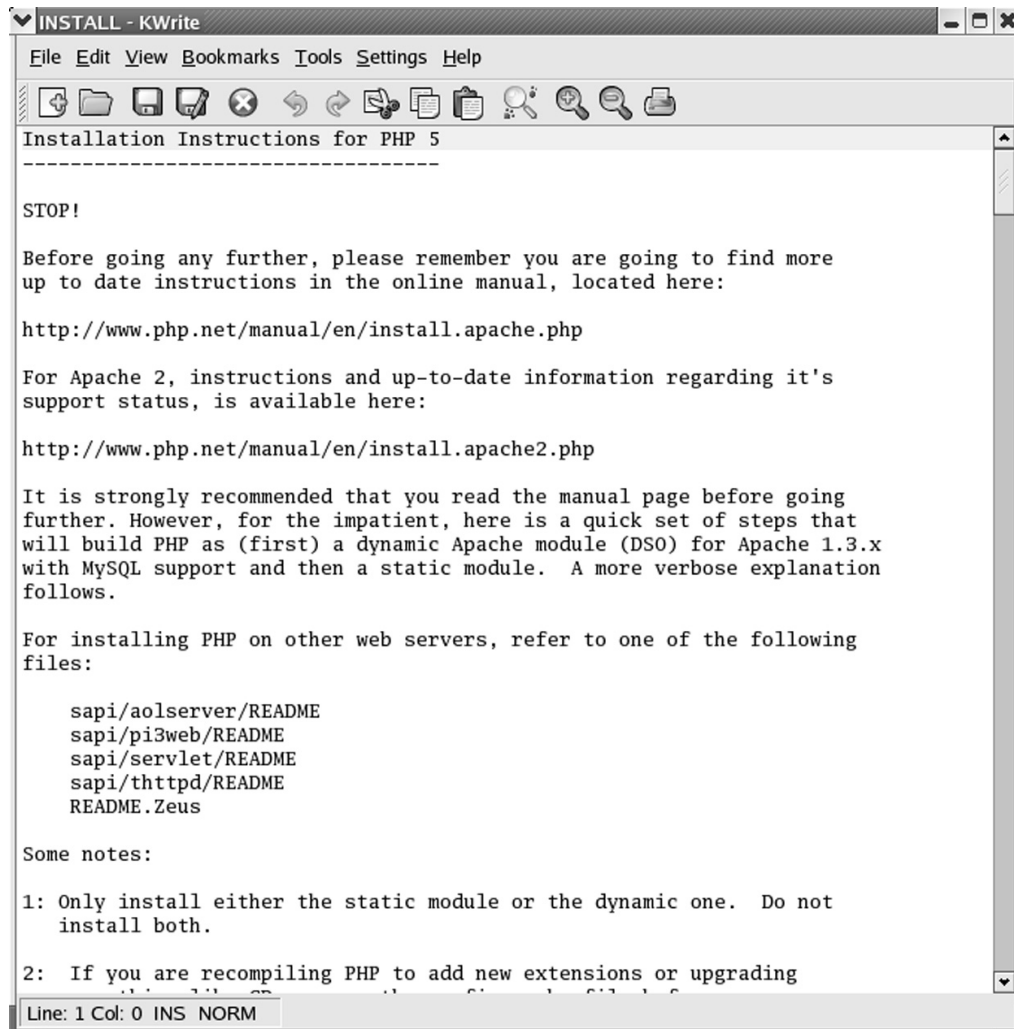


Рис. 1.6.

Проверка Apache для DSO-инсталляции

Прежде чем компилировать PHP 5 как DSO-объект, необходимо убедиться, что Apache установлен и настроен для работы с динамическими модулями. Для проверки Apache в терминале можно использовать следующую команду:

```
httpd -l
```

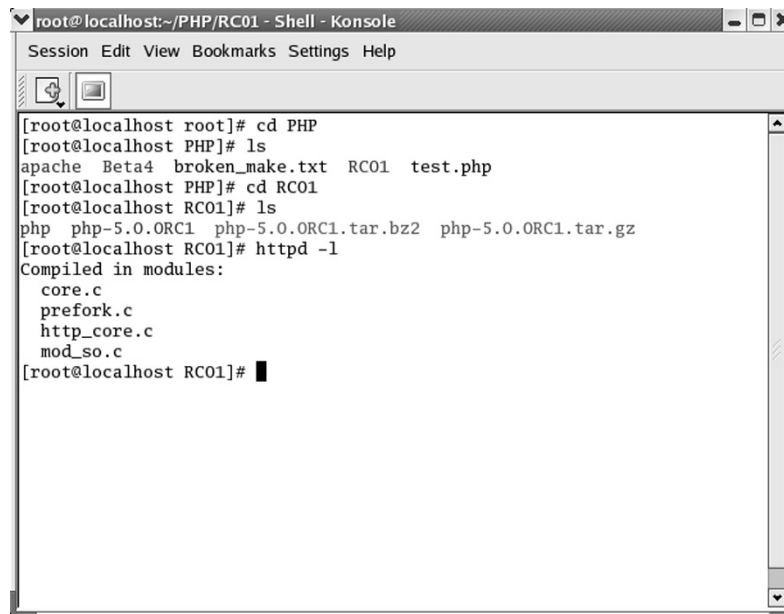
Должен появиться вывод наподобие вывода, показанного на рис. 1.7.

Если в выводе присутствует строка `mod_so.c`, то можно продолжать установку.

Запуск конфигурационного сценария.

Внутри дистрибутивного каталога PHP 5 (под названием `php-5.0.0RC1` или очень похожим) находится shell-сценарий `configure`. Данный сценарий принимает ряд аргументов, позволяющих управлять функциями, которые будут поддерживаться PHP.

Этот сценарий необходимо запустить для того, чтобы сконфигурировать компиляцию, а не сам PHP 5 (конфигурация PHP задается в файле `php.ini`, который будет рассматриваться далее).



```
root@localhost:~/PHP/RC01 - Shell - Konsole
Session Edit View Bookmarks Settings Help

[root@localhost root]# cd PHP
[root@localhost PHP]# ls
apache Beta4 broken_make.txt RC01 test.php
[root@localhost PHP]# cd RC01
[root@localhost RC01]# ls
php php-5.0.0RC1 php-5.0.0RC1.tar.bz2 php-5.0.0RC1.tar.gz
[root@localhost RC01]# httpd -l
Compiled in modules:
  core.c
  prefork.c
  http_core.c
  mod_so.c
[root@localhost RC01]#
```

Рис. 1.7. Запуск конфигурационного сценария

Команды, доступные для конфигурационного сценария

По умолчанию задается конфигурация, которая позволяет компилировать PHP 5 как CGI-программу. Чтобы скомпилировать PHP 5 как статический модуль, используется флаг `-with-apache`; как DSO-объект — флаг `-with-apxs`. Здесь описывается компиляция с параметром `-with_apxs` (а фактически с параметром `-with_apxs2`, так как используется Apache 2).

Далее приводятся некоторые аргументы командной строки, которые можно использовать для компиляции PHP 5. За конфигурационной командой `./configure` (текущий каталог `./` необходимо указывать для того, чтобы система могла выполнить сценарий `configure`) после одного пробела вводятся некоторые или все необходимые из перечисленных ниже параметров.

- ❑ `-enable-track-vars`: автоматически заполняет ассоциативные массивы значениями, переданными как часть GET- и POST-запросов или в cookie-файлах.
- ❑ `-with-gd = /путь/к/каталогу`: разрешает поддержку библиотеки GD, позволяющей сценариям динамически создавать GIF- и PNG-изображения. Можно компилировать PHP с поддержкой GD, а можно добавить соответствующий модуль позднее (см. главу 16).
- ❑ `-with-mysql = /путь/к/каталогу`: с поддержкой MySQL.
- ❑ `-with-pgsql = /путь/к/каталогу`: с поддержкой PostgreSQL.

Для быстрой инсталляции можно использовать только параметры `-with-mysql` и `-with-apxs2`. Если в процессе компиляции появится сообщение об ошибке, указывающее на то, что какой-либо файл не найден, то следует указать полный путь к каталогу, содержащему данный файл (или файлы). Например, как правило, конфигурационная команда находит путь к `mysql`, но если это не так, то можно указать полный путь к `mysql` как часть команды для запуска конфигурационного сценария.

Другие параметры конфигурации

Существует гораздо больше возможных аргументов командной строки. Например, можно ввести команду

```
./configure --help
```

и получить полный перечень используемых аргументов наряду с их описаниями.

Быстрая инсталляция (DSO)

Текстовый файл `INSTALL` рекомендует начинать инсталляцию с двух параметров: `-with-mysql` и `-with-apxs`. Для быстрой инсталляции следует запустить конфигурационный сценарий следующим образом:

```
./configure --with-mysql --with-apxs
```

При работе с более новой версией Apache 2 необходимо использовать флаг `--with-apxs2`, а не `--with-apxs`. Сценарий информирует об этом, что весьма полезно. Если прочесть все команды, появляющиеся в окне терминала во время выполнения сценария, то можно отметить, что сценарий выполняет немало подобных проверок, прежде чем можно будет выполнить команду `make`.

После выполнения конфигурационного сценария необходимо ввести еще две команды:

```
make
make install
```

Команда `install` создает в каталоге `/usr/local/lib` подкаталог с именем `php`, куда затем помещает копию каталога `PEAR` (репозиторий PHP-расширений и приложений) и файла `php.ini`. На рис. 1.8 показано, как выглядит каталог `php`.

Использование дополнительных конфигурационных параметров

Для компиляции PHP 5 можно использовать несколько других параметров конфигурационного сценария, например, `enable_track_vars`, `with-gd` и `with_pgsq1`. Однако если есть необходимость использовать конфигурационные параметры для `gd` (графический модуль) и `pgsq1` (база данных), то чтобы все работало правильно, следует убедиться, что данные программы также загружаются, а кроме того, необходимо указать полные пути к их каталогам.

Использование PHP как CGI-программы

PHP 5 компилируется как модуль в случае использования параметра `-with-apache` или `-with-apxs2`. Если при запуске конфигурационного сценария не использовать ссылок на Apache или `apxs`, то PHP будет скомпилирован как исполняемый бинарный CGI-файл. И если PHP 5 компилируется как CGI, то при установке бинарный файл фактически помещается в каталог `/usr/local/bin`. Его следует скопировать в каталог `cgi-bin`, используя такую команду:

```
cp /usr/local/bin/php /usr/local/apache/cgi-bin/php.cgi
```

Это позволит программам, использующим Apache, запускать различные PHP-страницы для разных идентификаторов пользователя. Однако бюллетень CERT advisory CA-96.11 не рекомендует помещать какие-либо интерпретаторы (например, PHP 5) в каталог `cgi-bin`,

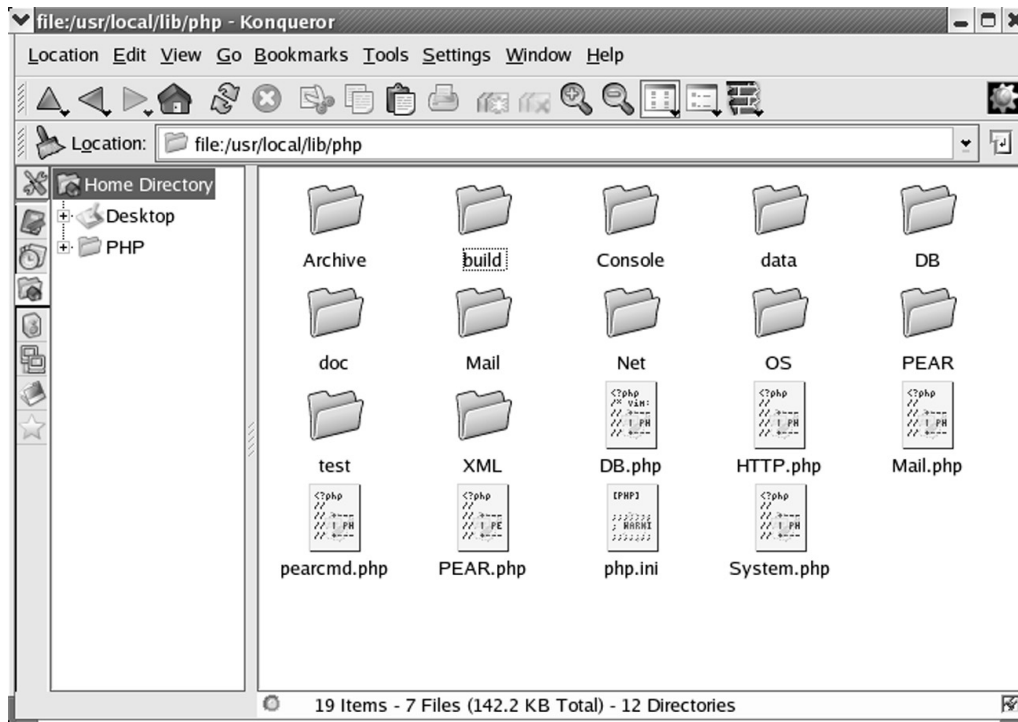


Рис. 1.8.

поскольку это позволяет осуществлять такие атаки на сервер, как доступ к системным файлам и Web-документам. При компиляции PHP 5 с параметром `-with-apache` создается интерфейс разработки серверных приложений (Server Application Programming Interface — SAPI), обеспечивающий более высокий по сравнению с CGI-методом уровень производительности и безопасности.

Настройка Apache для работы с PHP

Чтобы установить Apache, необходимо воспользоваться соответствующими RPM-пакетами или скомпилировать исходный код, но в большинстве случаев Apache поставляется с Linux-дистрибутивами и соответствующим образом устанавливается в ходе инсталляции Linux. Например, на моей инсталляции Red Hat Fedora Apache уже был установлен, и требовалось только проверить его, нажав кнопку Red Hat, выбрав пункты меню **System Settings** ⇒ **Server Settings** ⇒ **Services** (Системные настройки ⇒ Параметры серверов ⇒ Услуги). В появившемся списке необходимо найти строку `httpd` (рис. 1.9).

`httpd` означает HTTP-демон. Демон — название служб, работающих на Linux-машинах в фоновом режиме. Таким образом, `httpd` означает *HTTP-служба, работающая в фоновом режиме*, например, Web-сервер.

Если используется графический пользовательский интерфейс Linux (например, KDE), то следует щелкнуть на пункте `httpd`, чтобы проверить, запущена ли данная служба. В противном случае ее необходимо запустить, а затем ввести в браузере адрес `http://localhost` и нажать Enter. После этого в браузере должна появиться страница наподобие показанной на рис. 1.10.

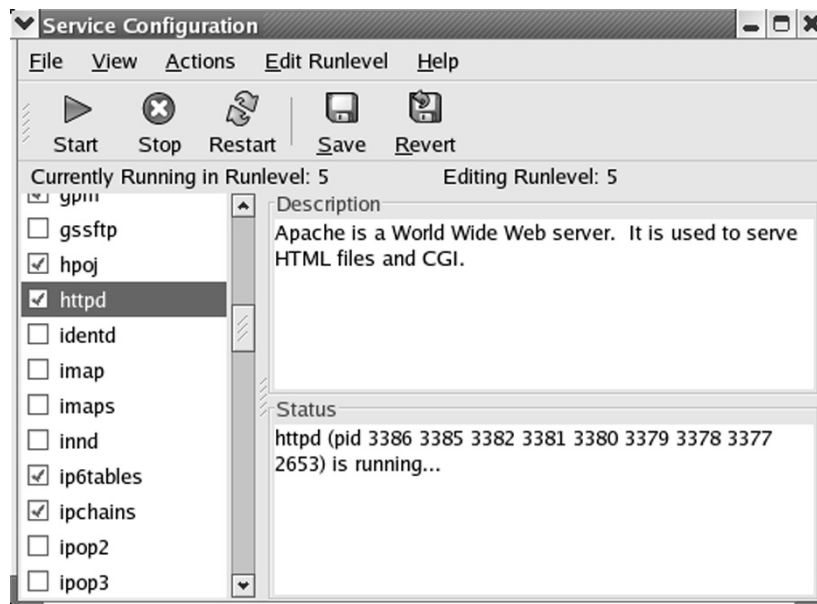


Рис. 1.9.

Если Apache еще не установлен, то для его инсталляции можно использовать следующие команды в окне терминала:

```
lynx http://httpd.apache.org/download.cgi
gzip -d httpd-2_0_NN.tar.gz
tar xvf httpd-2_0_NN.tar
./configure--prefix=PREFIX
make
make install
vi PREFIX/conf/httpd.conf
PREFIX/bin/apachectl start
```

NN необходимо заменить на второстепенный номер версии, а вместо PREFIX подставить корректный путь установки Apache (по умолчанию используется путь /usr/local/apache2).

Конфигурирование Apache для работы с PHP 5

Если PHP 5 устанавливается в качестве DSO-объекта (как в рассматриваемой здесь инсталляции), то необходимо проверить наличие нескольких записей в конфигурационном файле Apache (он называется httpd.conf). В Fedora данный файл находится в каталоге /etc/httpd/conf. Файл httpd.conf следует открыть в любом текстовом редакторе и модифицировать.

Во-первых, необходимо убедиться, что в Apache-сервере разрешено использовать PHP. Среди строк, начинающихся с LoadModule, должна присутствовать подобная строка:

```
LoadModule php5_module /usr/local/apache/lib/libphp5.so
```

Если такой строки нет, то ее необходимо добавить; если же задан неверный путь, его нужно исправить. Чтобы найти каталог, в который во время компиляции был помещен файл libphp5.so, можно воспользоваться Konqueror, выбрав в нем пункты меню Tools⇒Find file (Сервис⇒Найти файл). Обычно файл libphp5.so помещается в каталог /usr/lib/httpd/modules

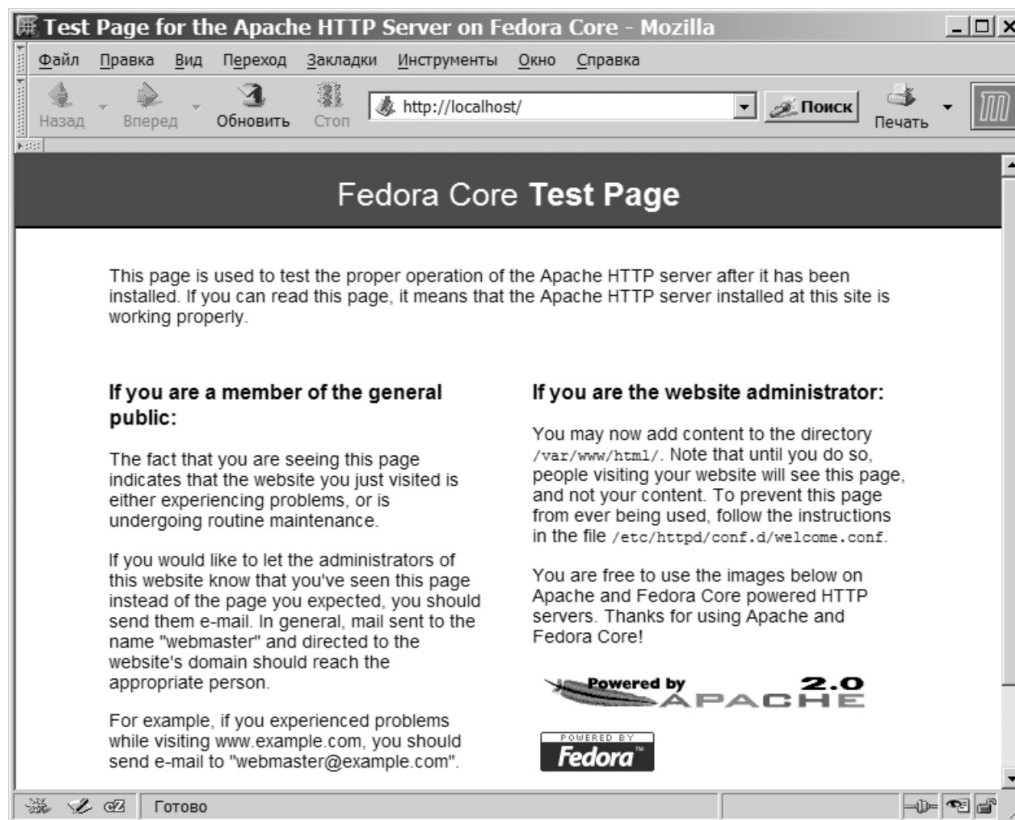


Рис. 1.10.

Эти сведения нужны для того, чтобы указать Apache, как выполнять PHP-сценарии. Для этого в конфигурационный файл Apache `httpd.conf` добавляется инструкция загрузки PHP 5 — `LoadModule` с указанием найденного только что пути. Эту инструкцию можно вставить после любой другой строки `LoadModule`.

```
LoadModule php5_module      /usr/lib/httpd/modules/libphp5.so
```

Теперь, когда Web-сервер “знает”, как загружать PHP 5, необходимо активизировать PHP в Apache. В следующем файле имеется раздел, строки которого начинаются с `AddModule`. Если в файле есть строка `ClearModulesList`, то необходимо добавить в файл следующую строку. Не имеет значения, куда будет вставлена данная строка, однако ее лучше располагать поближе к другим строкам `AddModule`, в дальнейшем это облегчит к ней доступ.

```
AddModule mod_php5.c
```

Строку `AddModule` вставлять необязательно, если нет строки `ClearModulesList`.

Наконец, необходимо указать Apache, как распознавать PHP-программы по расширению файла. В конфигурационном файле имеется несколько директив, начинающихся с `AddType`. В конце этого раздела необходимо добавить следующую строку:

```
AddType application/x-httpd-php .php
```

Она означает, что все файлы, имеющие расширение `.php`, являются PHP-программами. Теперь, когда конфигурирование закончено, следует сохранить файл.

Запуск или перезапуск Apache

Чтобы проверить, запущен ли Apache, необходимо снова вернуться в окно “Службы” и проверить состояние службы httpd. Если она не запущена, то ее следует запустить. Работу сервера можно проверить, открыв в браузере тестовую страницу <http://localhost>. Если все работает (а также если в ваши планы не входит изучение Windows-инсталляции PHP), то можно перейти к разделу “Тестирование и устранение неисправностей”, чтобы проверить работу PHP.

На большинстве Web-хостинговых серверов используются какие-нибудь версии Linux, например, Debian или RedHat, либо FreeBSD или другой клон Unix. Для таких машин предпочтительным Web-сервером является Apache. PHP совместим с Linux и Apache, поэтому его можно устанавливать и конфигурировать на этих системах. Однако если разработчик не владеет Web-хостинговым компьютером, то, скорее всего, контролировать установку и настройку PHP он не сможет. В такой ситуации (например, работая над существующим Web-сайтом, размещенным на чужом сервере) можно просто получить информацию об операционной системе, программном обеспечении Web-сервера и версии PHP, для того чтобы знать, как справиться с проблемами, возникающими при разработке PHP-программ.

Установка PHP 5 на Windows 2000/Internet Information Server 5

Прежде чем начинать процесс установки, следует кратко рассмотреть IIS-сервер. Если он был соответствующим образом установлен с использованием параметров по умолчанию, то, скорее всего, он уже работает. Убедиться в том, что IIS в данный момент работает, можно, нажав Пуск⇒Программы⇒Администрирование⇒Службы (Start⇒Programs⇒Administrative Tools⇒Services) и найдя в перечне службу Веб-публикации (World Wide Web Publishing). Если эта служба не запущена, то ее следует запустить.

Если нужно установить IIS, то перейдите к пункту Настройка⇒Панель управления (Settings⇒Control Panel) и откройте окно Установка и удаление программ (Add/Remove Programs). Затем щелкните на кнопке Компоненты, чтобы отобразить список доступных для установки в Windows 2000 компонентов, в число которых входит Internet Information Server. Выберите IIS и щелкните на кнопке Состав (Details), чтобы просмотреть все доступные для установки службы (такие как FTP, SMTP и т.д.). После этого следует выбрать любые необходимые службы, в частности обязательно должна быть выбрана служба Веб-публикации, и нажать кнопку Готово. В результате IIS будет установлен и запущен.

Можно исследовать инсталляцию IIS, открыв его документацию в браузере; для этого в адресной строке необходимо набрать URL <http://localhost/iisHelp>. Должна появиться страница наподобие представленной на рис. 1.11, которая демонстрирует работу IIS на Windows 2000.

Чтобы рассмотреть конфигурацию IIS, можно воспользоваться консолью управления IIS (Internet Service Manager). Для этого необходимо нажать кнопку Пуск и выбрать пункты Программы⇒Администрирование⇒Internet Services Manager. В окне Консоли управления Microsoft (Microsoft Management Console — MMC) появится пункт IIS (рис. 1.12).

Консоль MMC обеспечивает простой способ изучения и управления службами, представленными IIS, а также настроенными в нем Web- и FTP-сайтами. На рис. 1.13 показана иерархия стандартной инсталляции IIS.

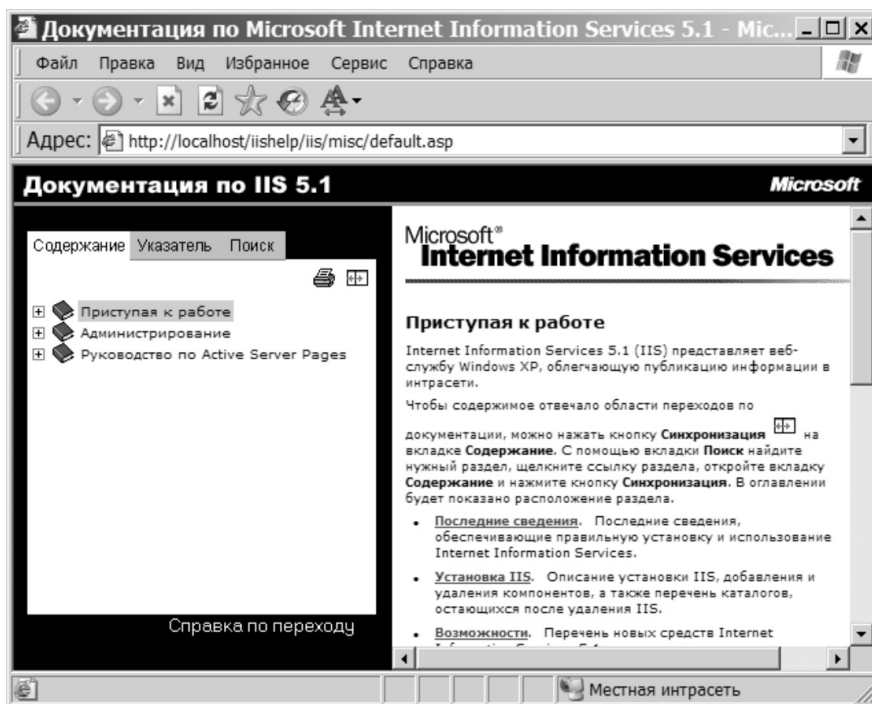


Рис. 1.11.

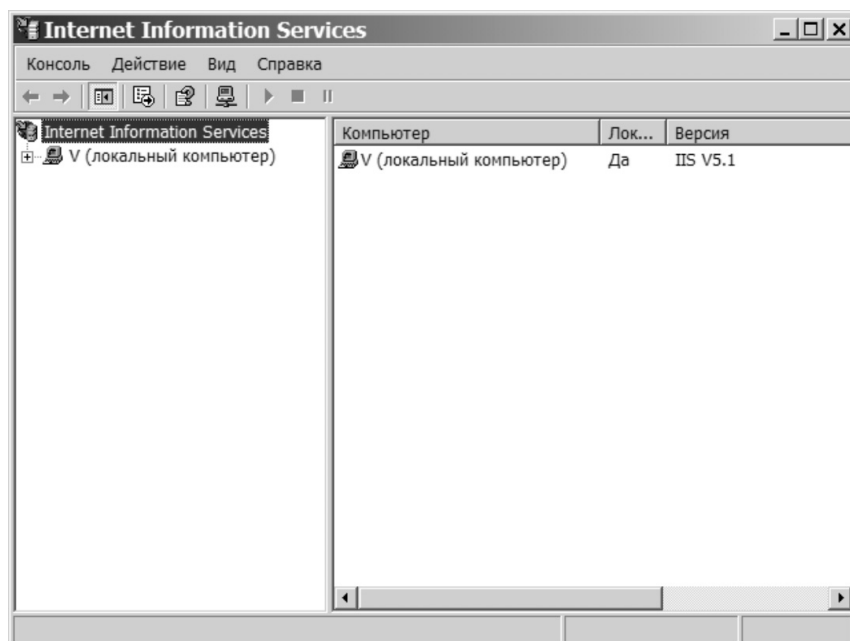


Рис. 1.12.

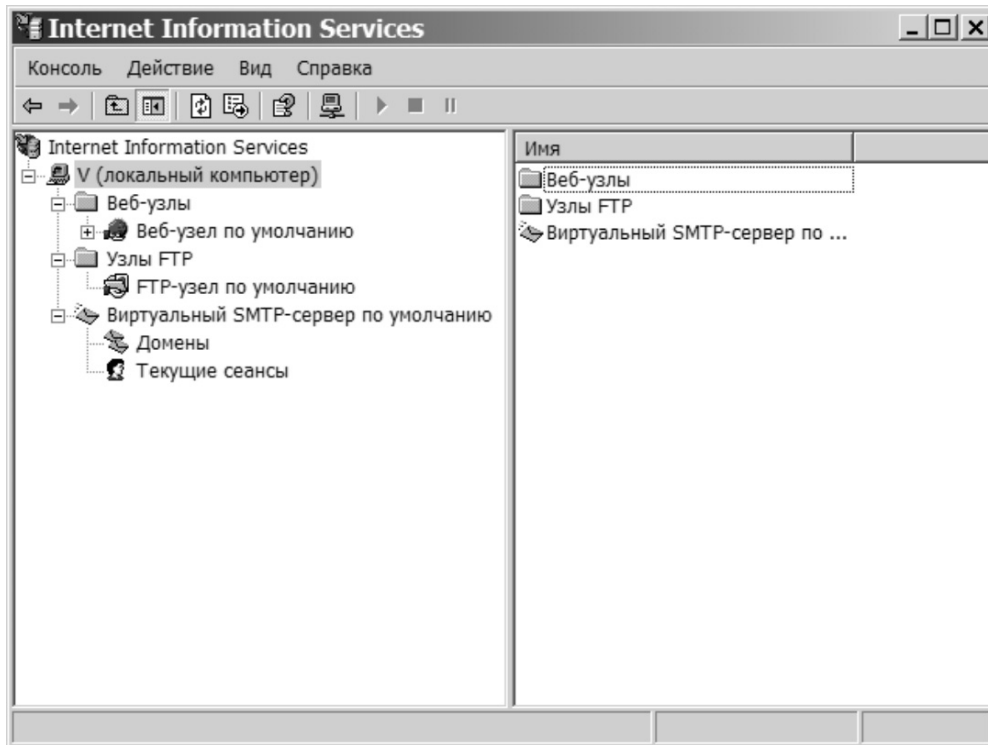


Рис. 1.13.

Чтобы установить PHP, необходимо отключить службы IIS, а затем после некоторых изменений перезапустить их. Для этого нужно щелкнуть правой кнопкой мыши на пункте **Веб-узел по умолчанию** (Default Web site) и выбрать пункт **Остановить** контекстного меню. После того как инсталляция PHP будет завершена, можно будет снова включить Web-сервер, а пока можно закрыть консоль для управления IIS.

Загрузка дистрибутива PHP 5

Получить наиболее свежую версию PHP 5 можно на сайте www.php.net в разделе “Downloads”, который показан на рис. 1.14 (внешний вид этого раздела со временем может измениться).

Загрузите бинарный файл Windows-инсталляции (в zip-архиве), поместите его в какой-либо заранее созданный каталог (например, `C:\PHP5RC01`) жесткого диска, а затем разархивируйте. В результате в каталоге PHP5RC01 должна образоваться иерархия, показанная на рис. 1.15 (как она выглядит в окне **Проводника Windows**).

Имена файлов со временем могут измениться, но рассматриваемая в данной книге версия PHP 5 весьма близка к окончательному выпуску, поэтому инсталляция PHP 5 должна работать так же, как здесь описано. И хотя Windows инсталлируется чаще всего на диск C:, книжная версия установлена на диске D:. Читателю, возможно, придется изменить некоторые пути с тем, чтобы использовать диск C: (или другой диск, на котором установлена Windows), для того чтобы PHP-инсталляция работала корректно.

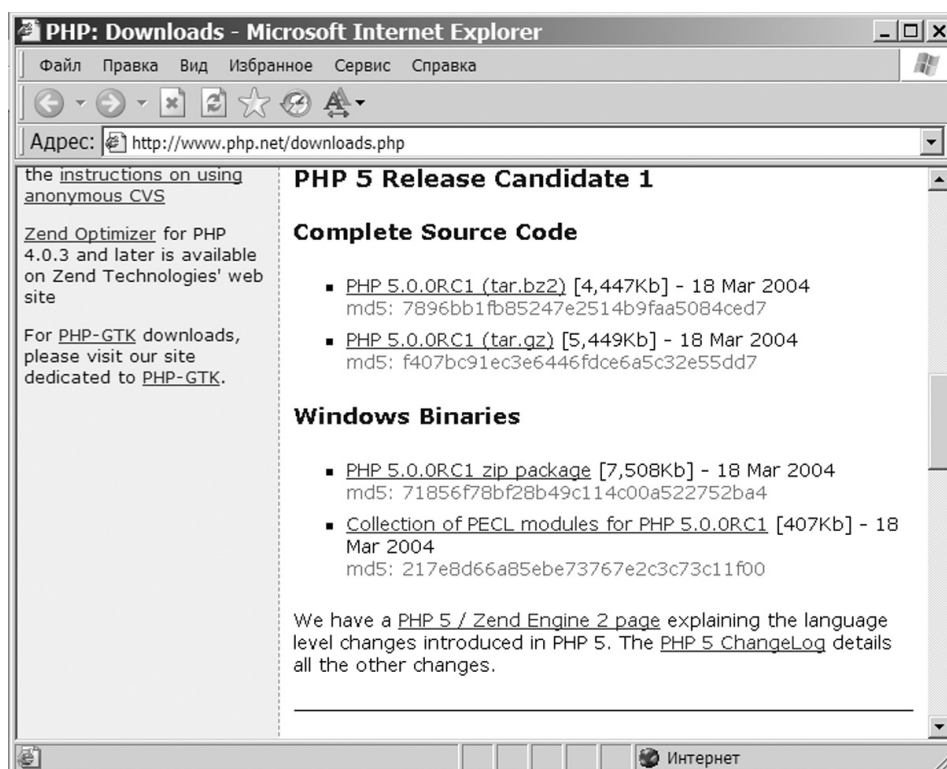


Рис. 1.14.

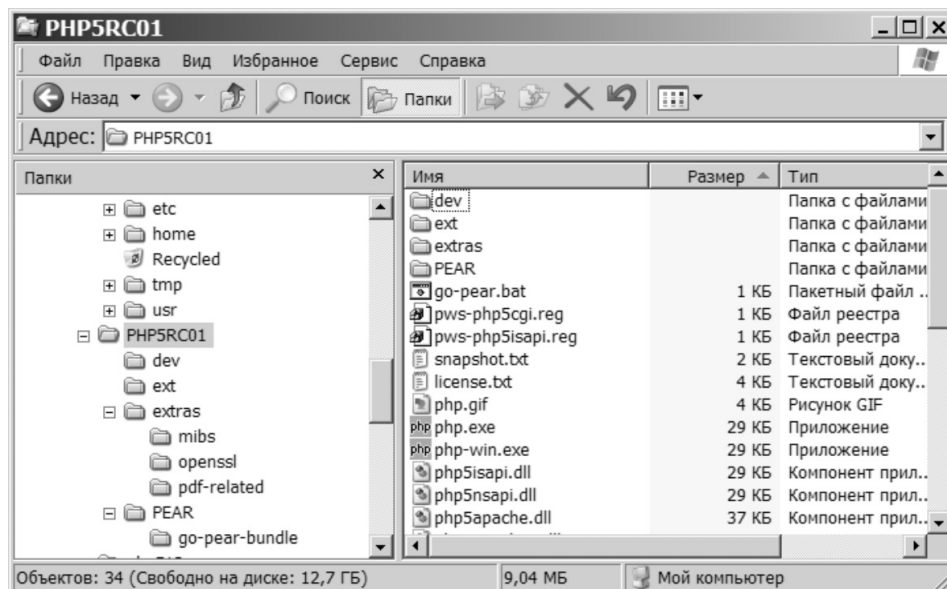


Рис. 1.15.

Теперь созданный для инсталляции PHP каталог (в данном случае PHP5RC01) содержит несколько подкаталогов и несколько текстовых файлов, программный файл `php.exe`, который фактически и будет использоваться, а также файл библиотеки `php5ts.dll` (в подкаталоге `dev`). Этот `dll`-файл необходимо скопировать в каталог `D:\WINNT\System32` (заменив `D`: корректной буквой диска на используемой машине). Кроме того, необходимо скопировать остальные `dll`-файлы в тот же каталог. Если Windows выдаст сообщение о том, что тот или иной файл уже имеется в системе, то следует оставить старый файл, не заменяя его только что загруженным.

Если копировать все `dll`-файлы в каталог `System32` нежелательно и PHP инсталлируется как SAPI (как и в данной книге), то для этих файлов можно создать другой каталог и изменить соответствующим образом системную переменную `PATH`. Системная переменная `PATH` содержит список каталогов (путей), в которых Windows ищет необходимые для работы программ файлы, например, `dll`-файлы. Если, предположим, для `dll`-файлов PHP пользователь создал каталог `C:\php5\dlls`, то в эту переменную необходимо записать строку `C:\php5\dlls`, после чего содержащиеся в нем файлы будут использоваться системой. Чтобы установить переменную `PATH`, необходимо нажать кнопку `Пуск`, выбрать пункты `Панель управления⇒Система⇒Дополнительно⇒Переменные среды` (`Control Panel⇒System⇒Advanced⇒Environment Variables`), а затем найти и задать необходимое значение переменной `PATH`.

php.ini и расширения

Как уже отмечалось ранее, файл `php.ini` содержит инструкции, которые обрабатываются во время запуска PHP, по установке конфигурационных параметров PHP и загрузке определенных расширений. Конфигурационные установки подобны выключателям — они включают или отключают различные режимы работы PHP. Расширения предоставляют дополнительные или усовершенствованные встроенные возможности PHP.

В корне созданного для PHP каталога должны находиться файлы `php.ini-dist` и `php.ini-recommended`. Файл `php.ini-dist` следует скопировать в каталог `D:\WINNT` (используя соответствующую букву диска), переименовать его в `php.ini`, а затем открыть в `Notepad`. В этом файле необходимо найти строки наподобие следующих:

```
extension_dir = C:\php\extensions ; directory in which the loadable extensions
(modules) reside
```

Убедитесь, что данный путь соответствует пути к каталогу расширений разархивированного дистрибутива PHP 5. Если это не так, то его следует соответствующим образом изменить (задать путь к каталогу `ext` в разархивированном каталоге PHP). В каталоге расширений содержится множество файлов, имена которых начинаются с `php_` и заканчиваются на `.dll`.

В следующем разделе файла `php.ini` задаются расширения, загружаемые PHP. Расширения, которые указаны в строках, начинающихся с точки с запятой, не загружаются — точка с запятой в начале директивы означает, что PHP игнорирует данную директиву. Следует удалить точку с запятой из строки `extension=php_gd.dll`, так чтобы текст файла выглядел примерно следующим образом:

```
;extension=php-filepro.dll
extension=php-gd.dll
;extension=php_mssql.dll
```

Это даст доступ к функциональности библиотеки GD, которая позволяет генерировать изображения в PHP-программах (подробнее данная тема рассматривается в главе 16, “Генерирование графики”). Модифицированный файл `php.ini` следует сохранить.

Теперь снова следует запустить консоль для управления IIS — Пуск⇒Программы⇒Администрирование⇒Internet Services Manager и открыть иерархию служб. Затем, щелкнув правой клавишей мыши на пункте Веб-узел по умолчанию (Default Web Site), выберите пункт Свойства (Properties) (рис. 1.16).

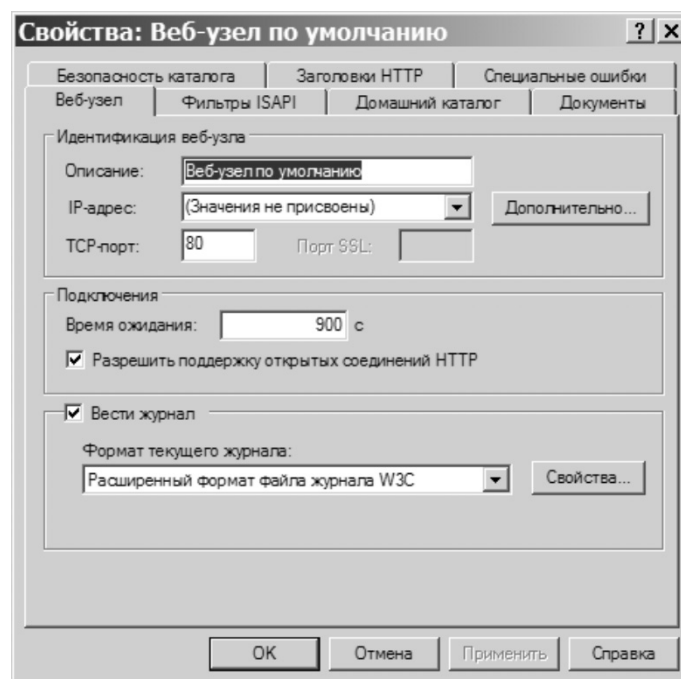


Рис. 1.16.

Здесь следует внести два изменения. Во-первых, необходимо зарегистрировать ISAPI-фильтр PHP 5, поскольку планируется установить PHP с его собственным SAPI-интерфейсом, а не как CGI-программу. Выберите вкладку Фильтры ISAPI (ISAPI Filters). Нажмите на кнопку **Добавить** (Add) и создайте новый фильтр, который называется PHP. В подкаталоге sapi дистрибутива PHP содержится файл `php5isapi.dll` — ISAPI-фильтр PHP. В появившемся диалоговом окне необходимо указать корректный путь к данному файлу, (рис. 1.17).

Во-вторых, нужно определить, к каким файлам IIS будет применять PHP5 фильтр. IIS должен интерпретировать все файлы, имеющие расширение `.php`, как PHP-программы. Во вкладке **Домашний каталог** (Home Directory) нажмите кнопку **Настройка** (Configuration). В следующем диалоговом окне нажмите кнопку **Добавить**. В результате на экране появится диалоговое окно **Добавление или изменение сопоставления расширений** (рис. 1.18).

Нажмите кнопку **Обзор** (Browse) и укажите путь к файлу `php5isapi.dll`. В поле **Расширение** (Extension) необходимо ввести `.php`, указав таким образом, что IIS должен применять данный фильтр к `.php`-файлам. Дважды нажмите **ОК**. Теперь необходимо перезапустить IIS. Закройте диалоговое окно **Свойства**, щелкните правой кнопкой мыши на пункте Веб-узел по умолчанию и выберите в контекстном меню пункт **Пуск** (Start).

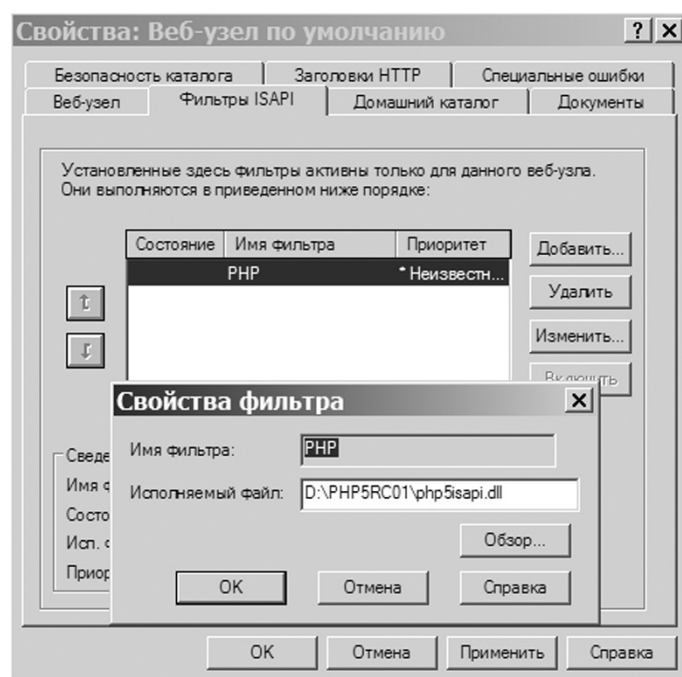


Рис. 1.17.

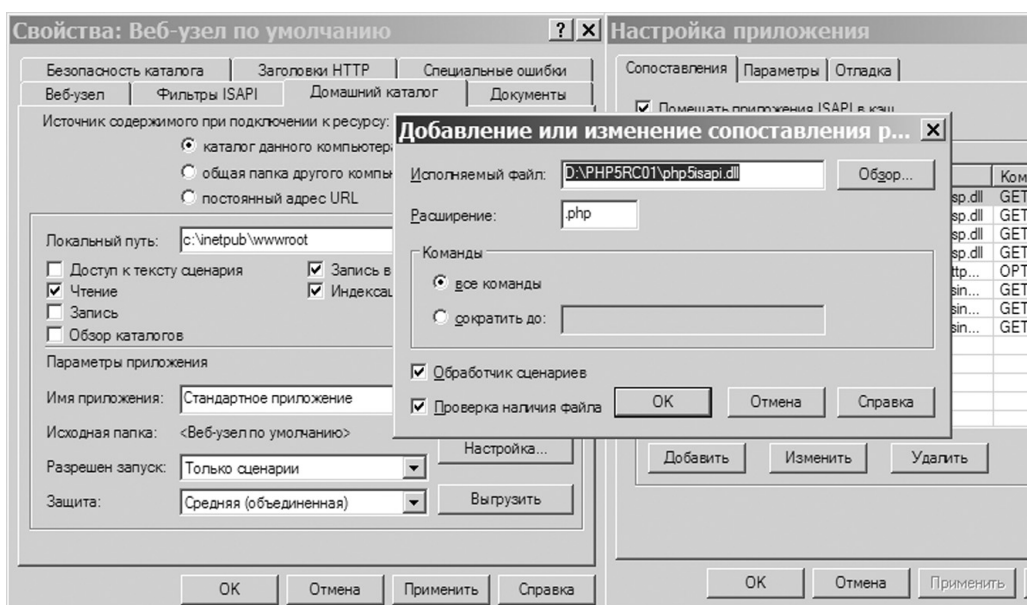


Рис. 1.18.

Если ММС-консоль сообщила о том, что служба Web-публикаций запустилась успешно, то PHP 5 установлен. Необходимо запомнить имя корневого каталога Web-сайта (в данном случае — `C:\Inetpub\wwwroot`).

Создайте в каталоге `wwwroot` какой-либо подкаталог. Назвать его можно как угодно (желательно, чтобы имя было понятным). Затем в каталог следует поместить файлы с расширением `.php`. Когда браузер будет запрашивать эти файлы, они будут обрабатываться посредством PHP-машины.

Теперь можно открыть текстовый редактор и создать текстовый файл, содержащий следующий код:

```
<?php
phpinfo();
?>
```

Созданный файл можно сохранить с именем `test01.php` (или любым другим именем, но с расширением `.php`) в только что созданном подкаталоге каталога `wwwroot`. Чтобы открыть данный файл в браузере, введите в адресной строке адрес `http://localhost` плюс имя его каталога, а также имя самого файла, например, так: `http://localhost/php_file/myfile.php`. В браузере должна появиться страница наподобие страницы, показанной на рис. 1.19 (хотя номер версии PHP может несколько отличаться, если вы используете более свежий выпуск PHP).

Если это так, то все работает. Если же нет, обратитесь к следующему разделу.

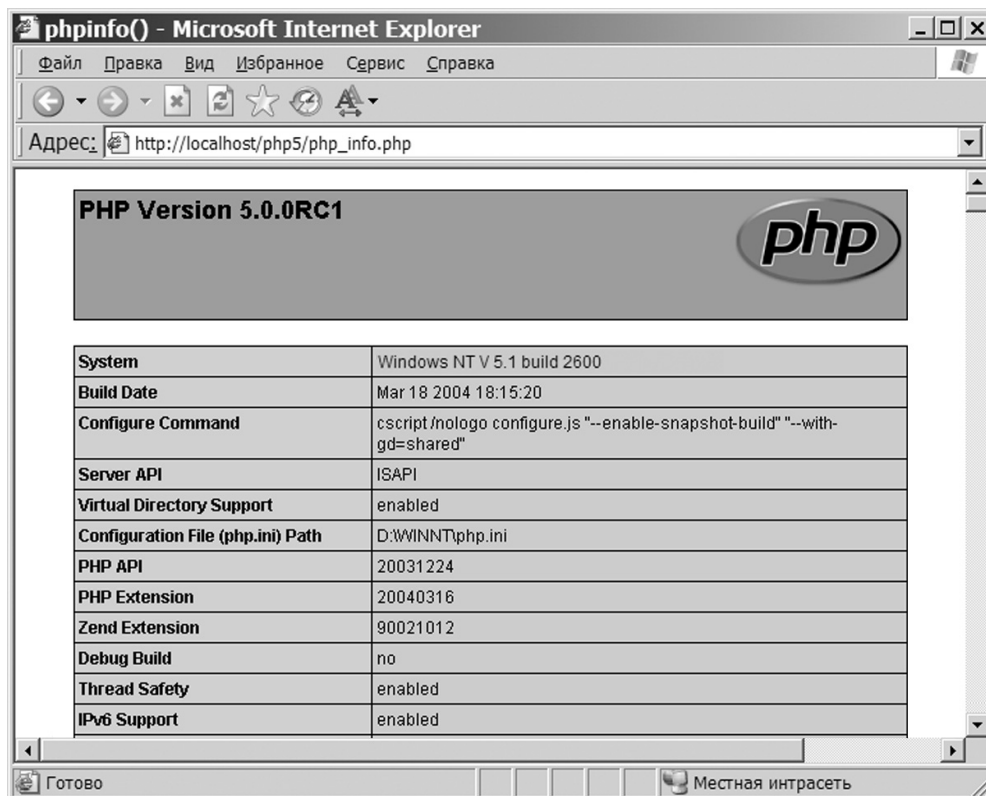


Рис. 1.19.

Тестирование и устранение неисправностей

Протестировать установку PHP несложно. Следует начать с написания и запуска небольшой PHP-программы. Создается небольшой тестовый PHP-файл, например, со следующим кодом:

```
<?php
echo "Ура, заработало!";
?>
```

Созданный файл необходимо сохранить в любом подкаталоге каталога wwwroot (или в соответствующем каталоге, если используется не Windows 2000/IIS, а другая комбинация операционная система/Web-сервер).

Затем следует открыть данный файл в браузере. В результате этого в браузере должны появиться слова: “Ура, заработало!”. Если вместо них появляется ошибка “Невозможно отобразить страницу” (“Page cannot be displayed”), то либо проблема в установке PHP, либо система не может найти файл, либо в конфигурации Web-сервера есть ошибка. Если в Web-браузере появляется сообщение о синтаксической ошибке (parse error), то, возможно, при вводе кода была допущена опечатка. Ошибки в коде подробнее рассматриваются в нескольких последующих главах, а здесь описаны некоторые пути устранения неисправностей в базовой установке PHP.

Работает ли тестовый файл? Или вы читаете данный раздел именно потому, что он не работал? Беспокоиться не стоит. Осваивая что-нибудь новое, человек неизбежно сталкивается с весьма распространенными проблемами, это особенно верно, когда речь идет о компьютерах и программировании. Фактически, ошибки в данном случае можно считать удачей, поскольку они стимулируют освоение PHP и программирования.

Устранение неисправностей и отладка включают в себя процесс идентификации проблем, отслеживания возможных причин, логической изоляции этих причин до тех пор, пока не будет найден наиболее вероятный источник проблемы, и последующих попыток найти решение. Конечный результат заключается в том, что проблема исправлена. Если программист хорошо выполнил свою работу, то проблема будет исправлена (изящно и надежно), и это не вызовет других проблем. (В главе 5 отладка программ рассматривается более подробно.)

Процесс устранения неисправностей установки PHP можно разбить на такие этапы:

1. Необходимо проверить, запущен ли сервер и правильно ли он работает. В Windows 2000 это можно сделать, проверив службы (Пуск⇒Программы⇒Администрирование⇒Службы (Start⇒Programs⇒Administrative Tools⇒Services)), особенно службы Internet-администрирования и службу Веб-публикаций. Только тот факт, что они настроены на автоматическую работу, не означает, что они включены. Чтобы подстраховаться, их можно остановить и перезапустить снова. Кроме того, можно проверить (в консоли управления ИС), работает ли Web-сервер с Веб-узлом по умолчанию. В случае Apache и Linux необходимо проверить httpd-службу (это также можно сделать путем ввода `http://localhost` в адресную строку браузера).
2. Можно поместить простой HTML-файл в каталог wwwroot, убедившись, что он имеет расширение .htm или .html (например, test01.htm), и вызвать его в браузере. При этом следует убедиться, что в адресной строке используется URL

`http://localhost/test01.htm`, а не путь к файлу (например, `D:\inetpub\wwwroot\test01.htm`).

3. Если HTML-страница отображается правильно, значит можно быть уверенным, что Web-сервер функционирует. Это означает (предположим, что не отображаются PHP-страницы), что проблема заключается в инсталляции PHP. Если появляются другие сообщения (такие как 404 Файл не найден (404 Page Not Found)), то весьма вероятно, что система просто не может найти данный файл, поэтому необходимо еще раз проверить имя файла, имя Web-каталога и т.д.
4. Если проблема предположительно заключается в инсталляции PHP, то следует еще раз проверить процесс установки, тщательно отработать каждый этап и убедиться, что все файлы PHP расположены в соответствующих каталогах. Особое внимание необходимо уделить именам системных каталогов, поскольку они могут отличаться в зависимости от операционной системы Windows или Linux.
5. Следует проверить права доступа к файлам. Права доступа очень важны в Linux-системах и в меньшей степени — в Windows 2000 или настольных версиях Windows. Можно зарегистрироваться в системе как root или администратор в Linux или Windows и попытаться изменить права доступа для запуска сценариев из каталогов Web-сервера. В случае использования внешнего хостинга изменить права доступа на Linux-системах можно при помощи любой хорошей FTP-утилиты.

Данные этапы помогут локализовать проблему и подсказать возможное решение.

Конфигурирование PHP

В ходе инсталляции, для того чтобы повлиять на работу PHP и его функций, модифицировался файл `php.ini`. В Приложении E “Конфигурация PHP5” данной книги обсуждаются основные установки в файле `php.ini`, а также некоторые доступные для PHP расширения. Наиболее важные конфигурационные параметры и расширения PHP рассматриваются в данном разделе.

Файл `php.ini`

Файл `php.ini` анализируется при первой загрузке и запуске PHP, для того чтобы задать определенный режим работы PHP (для любого сценария, выполняемого на Web-сервере). Все строки, в начале которых нет точки с запятой, являются действующими командами; все остальное в данном файле следует рассматривать как комментарии. Ниже приведен текст нескольких разделов файла `php.ini-recommended`. Показанные установки важны потому, что они непосредственно влияют на то, как работает PHP (в обычных обстоятельствах), они также могут повлиять на код или безопасность разрабатываемых приложений.

```
;;;;;;;;;;
; Ограничения ресурсов ;
;;;;;;;;;;

max_execution_time = 30    ; Максимально возможное время выполнения
;                           сценария в секундах.
max_input_time = 60       ; Максимально возможное время
;                           синтаксического анализа данных запроса
memory_limit = 8M         ; Максимальный объем памяти, выделяемый
;                           сценарию (8MB)
```



```
; Должен ли PHP регистрировать EGPCS-переменные как глобальные
; переменные. Можно отключить эту функцию, чтобы не "засорять"
; глобальную область видимости сценария. Это особенно полезно,
; если используется директива track_vars — в этом случае
; получить доступ к GPC-данным можно через переменные $_GET, $_POST, $_REQUEST.
; Желательно так писать сценарии, чтобы они по возможности
; обходились без директивы register_globals. Использование
; данных, поступивших из формы, как глобальных переменных,
; потенциально может породить проблемы в защите сценария, если
; программист специально не позаботится об их устранении.
register_globals = Off
```

Гораздо более полная информация, касающаяся настроек в файле `php.ini`, приведена в Приложении Е.

PHP-расширения

PHP-расширения представляют собой программируемые возможности, которые добавляются к PHP или улучшают его встроенные средства для выполнения полезной работы в PHP-программах. В первых главах данной книги специальные расширения не используются. Все доступные расширения рассматриваются в Приложении Е.

Кэширование

Кэширование представляет собой метод временного сохранения некоторых результатов, для того чтобы не повторять всю обработку каждый раз при поступлении новых запросов к серверу. Один из потенциальных недостатков выполнения всего кода на сервере заключается в том, что если на клиенте (или на какой-либо машине между конечным пользователем и сайтом) включено кэширование, то пользователь может не получить наиболее свежей сгенерированной страницы. Чтобы обойти кэширование (по крайней мере, для большинства браузеров и серверов), можно вставлять в сценарии следующий код:

```
<?php
header("Cache-Control: no-cache, must-revalidate");
header("Pragma: no-cache");
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
?>
```

Первая строка хорошо работает с протоколом HTTP версии 1.1, вторая — с HTTP 1.0, а третья работает, определяя дату в прошлом (более подробно протокол HTTP рассматривается в главе 2).

Резюме

В данной главе вкратце рассматривалась история PHP и несколько способов установки PHP наряду с распространенным программным обеспечением Web-серверов.

В главе были даны рекомендации по установке PHP на Windows и Linux-платформы в виде CGI-программы или в виде отдельного модуля, раскрыто основное назначение некоторых PHP-настроек, описано месторасположение файлов PHP после инсталляции, представлена методика тестирования инсталляции. Кроме того, в главе освещалось базовое определение поиска и устранения неисправностей и отладки, а также последовательность этапов, которые следует выполнить, если инсталляция PHP не работает.

Упражнения

Предлагаемое упражнение будет полезным всякий раз при установке или работе с РНР на новой платформе.

Чтобы выполнить данное упражнение, необходимо создать документ, в котором собраны все следующие сведения:

- ☐ Каковы аппаратные возможности компьютера, на котором работает РНР? Опишите процессор, жесткий диск, оперативную память и т.д., а также любые предполагаемые ограничения.
- ☐ Какая операционная система работает на данном оборудовании? Укажите версию, а также все текущие заплатки и известные дефекты.
- ☐ Какой Web-сервер работает на данной машине? Укажите версию, заплатки и известные дефекты. Кроме того, опишите конфигурацию Web-сервера, корневой каталог, то, как РНР настраивается для работы с сервером, а также права доступа, которые пришлось установить.
- ☐ Какая версия РНР была установлена? Укажите версию, установленные файлы, каталоги, в которые они были установлены, а также все параметры реестра, настроенные или созданные для поддержки инсталляции РНР.
- ☐ Какие конфигурационные установки были заданы или изменены (по сравнению со стандартными) для установки РНР? Перечислите их.
- ☐ Какие расширения были включены? Перечислите их все, а также укажите причины, по которым вы их включили.

2

Написание простых программ

Читатель, вероятно, к этому моменту уже установил РНР и все остальные программные компоненты, необходимые для того, чтобы приступить к использованию РНР 5. Кроме того, читатель должен знать, что большинство создаваемых им программ предназначены для работы на Web-сервере и в них используются HTML- или XHTML-страницы для отображения пользовательского интерфейса и результатов обработки данных в браузере (для PC-пользователей браузером, скорее всего, будет Internet Explorer).

В данной главе рассматриваются основополагающие принципы написания РНР-программ, взаимодействие с которыми пользователь осуществляет через браузер. Здесь описаны основные аспекты правильного написания РНР-программ, в частности, вставки РНР-кода в HTML- или XHTML-страницы, использование нескольких широко распространенных РНР-функций (таких как `echo`, `date()`, `strlen()` и т.д.), а также создание и использование переменных. В данной главе представлено несколько примеров разработки простых программ, демонстрирующих типичное использование РНР 5 в простых Web-страницах, а также кратко описана работа РНР-программ.

Материал данной главы знакомит читателя с операторами и выражениями, а также с исключительно полезными переменными, которые называются массивами. Массивы, как и обычные переменные, применяются для хранения данных, однако они имеют ряд функций, которые делают их весьма мощным средством для различных типов обработки данных.

В этой главе представлены фундаментальные понятия для материала последующих глав, поскольку почти вся обработка данных в РНР-программах зависит непосредственно от умения программиста соответствующим образом именовать, а также использовать переменные и связанные с типами данных РНР-функции и особенно массивы.

Создание PHP-программы

Для начала напишем Web-страницу, которая будет отображаться в любом существующем браузере. Для этого выполните следующие действия.

1. Откройте программу Блокнот или любой другой доступный текстовый редактор и наберите в нем следующий HTML-код:

```
<html>
<head>
  <title>Web-страница</title>
</head>
<body>
  Этот текст появляется в окне браузера
</body>
</html>
```

2. Сохраните данный файл с каким-либо именем и расширением .htm (например, simple01.htm).
3. Откройте файл в браузере. Результат показан на рис. 2.1.

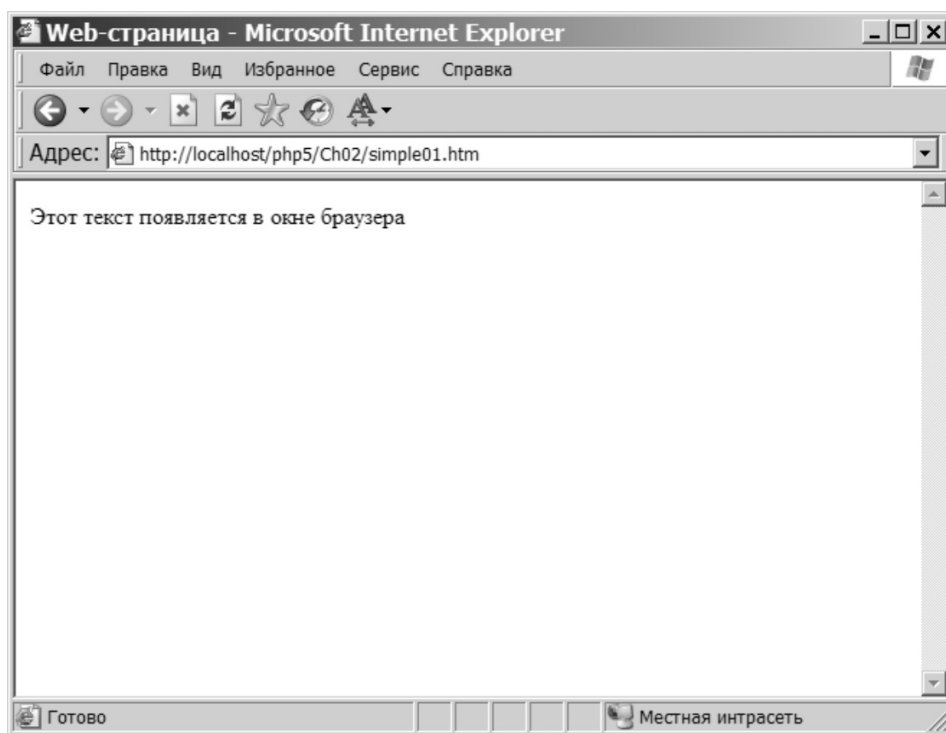


Рис. 2.1.

4. В зависимости от инсталляции и настройки, описанной в главе 1, загрузите файл в соответствующий каталог сервера (если Web-сервер работает на локальной машине, то файл следует просто скопировать в каталог, обслуживаемый

Web-сервером). Снова откройте файл в браузере, используя HTTP-адрес для локального узла. Результат должен выглядеть аналогично.

5. Теперь замените строку, начинающуюся с “Этот текст ...”, следующими строками:

```
Этот текст представляет данные, полученные в результате работы PHP 5: Сегодня
<?php
$todaydate = date("m",time()) . "-" . date("d",time()) . "-" . date("Y",time());
echo $todaydate;
?>
```

6. Сохраните файл, скопируйте его на сервер, если необходимо, и обновите страницу в браузере. Возможно, ничего не изменилось, если расширение файла не было изменено с .htm на .php. Очевидно, что для того чтобы Web-сервер передал данный файл PHP-процессору, во-первых, он (сервер) должен определить, что данный файл является PHP-файлом, а определить это можно по расширению файла. Во-вторых (предположим, что файл имеет соответствующее расширение), PHP-процессор выбирает для обработки разделы PHP-кода путем *синтаксического анализа* этого файла и поиска в нем PHP-тегов (<?php и ?>), а затем *выполняет* PHP-код. Синтаксический анализ означает, что PHP-процессор считывает отдельные команды и проверяет их на синтаксические ошибки (синтаксические ошибки — весьма распространенное явление при создании PHP-кода). Под выполнением кода следует понимать просто фактическую обработку кода PHP-процессором.
7. Измените расширение файла на .php и введите его адрес в браузере. На этот раз код должен работать и в браузере должна появиться текущая дата в конце строки текста.

Некоторые детали

В PHP, как и в любом другом языке программирования, существует множество команд, ключевых слов, операторов, языковых конструкций и функций. PHP-процессор ищет эти элементы в процессе синтаксического анализа и, если синтаксис корректен, то обрабатывает их, а затем возвращает результат обработки. Когда выполняется Web-приложение, результаты должны быть совместимы с HTML или языками написания сценариев на Web-странице, отображающей результаты. Поэтому часто результаты обработки включают в себя HTML-теги (подробнее о том, как работает HTML, рассказывается в главе 3).

Функция echo, по сути, не является функцией — она представляет собой конструкцию языка, т.е. ей не требуется передавать параметры в скобках, она не возвращает никаких значений (однако она делает именно то, что предполагается — отправляет в браузер строковое значение), а, кроме того, имеет несколько других ограничений, не свойственных функциям. Конструкция echo — не единственная языковая конструкция в PHP; другим примером является unset. В данном случае важно понимать то, что echo отправляет браузеру строку; в последующих главах различия между языковыми конструкциями и функциями рассматриваются более подробно.

Итак, команда echo отправляет пользовательскому браузеру строковые данные. В состав этих данных может включаться HTML-код, но если его нет, то пользователь увидит в браузере текст (браузер интерпретирует такие данные как текстовый файл

и “подставляет” HTML-код, чтобы данные отображались как обычно). В следующем простом примере `echo` отправляет строковые данные, сформированные на основании текущей даты и записанные в переменную `$todaysdate`:

```
echo $todaysdate;
```

Функция `date()` встроена в PHP, т.е. нет необходимости создавать или копировать эту функцию в разрабатываемой PHP-программе; она доступна для использования в любой момент. Она вызывается так же, как и функции в большинстве языков — указывается имя функции, за которым следуют круглые скобки. В скобках указываются аргументы — значения, выражения или другие функции, возвращающие значения, которые затем используются функцией `date()` для вычисления окончательного результата. Рассмотрим следующую строку из приведенного ранее примера программы:

```
$todaysdate = date("m",time()) . "-" . date("d",time()) . "-" . date("Y",time());
```

Первый аргумент `"m"` сообщает функции `date()` о том, что она должна вернуть порядковый номер месяца. В качестве второго аргумента используется функция `time()`, которая возвращает текущее время. Функция `date()` обрабатывает текущее время и извлекает из него номер месяца в виде значения, состоящего из двух символов. Функцию `date()` можно использовать снова, для того чтобы получить день и месяц из значения, возвращаемого функцией `time()`.

Как работает PHP-код

Как следует из предыдущего примера, для запуска PHP-программ внутри Web-страницы необходимо выполнение следующих основных требований:

- ☐ наличие Web-страницы, с которой может взаимодействовать пользователь;
- ☐ файл с расширением `.php`;
- ☐ опознаваемые PHP-теги;
- ☐ синтаксически корректный PHP-код.

Рассмотрим каждое требование подробнее.

Web-страница (пользовательский интерфейс)

Важно четко представлять себе, как отображается Web-страница, а также то, как работает PHP 5, поскольку отображаемая страница — это все, что видит конечный пользователь PHP-программы. Уместно подчеркнуть, что, хотя отображаемый вывод часто называют HTML-кодом, HTML сдает свои позиции языку XHTML, а браузеры поддерживают обработку другого подобного вывода, например, XML с XSLT. Несмотря на то, что в данной книге для обозначения вывода, отправляемого Web-сервером, используется аббревиатура HTML, под выводом также следует понимать код других языков, который может быть визуализирован браузером.

Можно писать PHP-программы так, чтобы PHP-код встраивался непосредственно в HTML-код, или же можно по мере необходимости ссылаться на HTML-код из PHP-кода. В любом случае весь вывод, отправляемый конечному пользователю, должен представлять собой HTML-код.

Почему? Попытаемся открыть `.php` не как Web-страницу, а как файл (для этого в браузере необходимо использовать пункт меню **Файл**⇒**Открыть** и выбрать файл на жестком диске). При этом в браузере PHP-код не отображается (браузеры игнорируют

PHP-теги и все, что находится между ними), однако, воспользовавшись меню Вид⇒ Просмотр HTML-кода, можно увидеть необработанный PHP-код. Это связано с тем, что ни Web-сервер, ни PHP-процессор не обрабатывали данный файл до того, как он был открыт в браузере.

Расширения файлов

Если вы уже создавали Web-страницы, скорее всего, вы знакомы с расширениями .htm и .html, а также, возможно, .shtml. С помощью данных расширений браузер распознает тип открываемого файла. Web-сервер также способен распознавать расширения файлов, и если он встречает .php-файл, то передает его PHP-машине для обработки. Расширение .php появилось не в PHP 5, а является стандартным расширением PHP-файлов для большинства Web-серверов.

Существует возможность так сконфигурировать Web-сервер, чтобы он отправлял PHP-процессору файлы с другими расширениями (например, .htm или .html). При использовании такой конфигурации обычные HTML-страницы будут обрабатываться PHP-машинкой (в дополнение ко всем PHP-страницам), хотя обычные HTML-файлы не изменяются в результате этой обработки (это просто вносить некоторые издержки). Так как файлы, отправленные пользователям, будут иметь расширения .htm или .html вместо .php, пользователь не узнает, что для серверной обработки данных файлов используется PHP. Добавлять эти расширения или нет — дело выбора разработчика, однако данный способ рекомендуется в тех случаях, когда от пользователя необходимо скрыть PHP-природу Web-приложения.

PHP-разделители

Разделители используются в различных типах кода для указания блоков кода, данных и т.д. Разделители представляют собой специальные символы, которые сообщают программе синтаксического анализа начало и окончание данных. Так, в формате с разделяющими запятыми разделителями являются запятые. Программа или процессор, анализирующий синтаксис потока данных, “знает”, что между двумя соседними запятыми должны быть данные соответствующего типа.

Та же идея применяется к PHP-коду, встроенному в Web-страницу. Стандартными разделителями для PHP 5 являются последовательности символов <?php и ?>. Можно также использовать в качестве разделителей последовательности <? и ?>, но <?php и ?> более предпочтительны — они стандартизированы в репозитории PHP-расширений и приложений (PHP Extension and Application Repository — PEAR), который является хорошим источником стандартного PHP-кода и предметом обсуждения главы 14.

Как и другие настройки в PHP 5, разделители, которые будут распознаваться PHP-машинкой, можно установить или расширить путем редактирования конфигурационного файла. Например, можно заставить PHP 5 распознавать в качестве разделителей символы <% и %>. Такие разделители называются ASP-разделителями, поскольку они соответствуют разделителям, используемым для написания ASP-кода, встраиваемого в Web-страницы. (ASP или Active Server Pages (активные серверные страницы) — технология Microsoft, подобная PHP.)

Кроме того, ограничивать PHP-код можно с помощью HTML-тегов сценариев, например:

```
<script language="PHP">Здесь расположен PHP-код</script>
```

Корректный PHP-код

Как и в любом другом языке программирования, PHP-код должен быть корректно написан. Когда PHP-программа выполняется на Web-сервере и проходит через PHP-процессор, любые ошибки в коде приводят к тому, что пользователь видит страницу с сообщением об ошибке.

Итак, очевидно, что PHP-код должен быть корректно написан. Вместе с тем синтаксически корректное написание программы (и ее выполнение без сообщений об ошибках) не гарантирует того, что программа выдает “правильный” ответ — в коде могут присутствовать логические ошибки. (Эта тема более подробно обсуждается в главе 5.)

Кстати, код в рассматриваемой здесь простой PHP-программе работает прекрасно.

Общие маркеры в коде

В PHP используется несколько символов для указания конца строк и ограничения блоков кода. Следует отметить, что обе строки кода программы заканчиваются точкой с запятой (;):

```
$todaysdate = date("m",time()) . "-" . date("d",time()) . "-" .  
date("Y",time());  
echo $todaysdate;
```

В PHP-коде:

- ☐ выражения заканчиваются точкой с запятой (;);
- ☐ блоки кода заключаются в фигурные скобки ({});
- ☐ комментарии в коде начинаются с символов // (для однострочных комментариев) или начинаются с /* и заканчиваются */ (для многострочных комментариев).

Ниже показано, как эти маркеры выглядят в блоке *псевдокода* (фиктивный код, который используется для описания обработки в PHP или иллюстрации какой-либо идеи):

```
<?php  
//разместим здесь оператор echo  
echo "небольшое количество псевдокода";  
if ($var1 == $var2) {  
    //что-то сделать  
    /* это многострочный комментарий  
    делаем что-то  
    делаем что-то другое  
    */  
}  
?>
```

Очень важно помнить об этих требованиях, поскольку в противном случае PHP-процессор будет генерировать и отображать сообщения о синтаксических ошибках. Вероятнее всего, запомнить эти требования будет труднее всего программистам, работавшим с Visual Basic или ASP, поскольку в этих языках точка с запятой и фигурные скобки не используются, а комментарии начинаются с апострофа (а не с //).

Как работают PHP-программы в Web-среде

В отличие от настольных приложений, которые запускаются на локальных системах, когда активизируются .exe-файлы, PHP-программы в сети выполняются, когда Web-серверу поступает запрос. Запрос пытается заставить Web-сервер получить и отправить запрашиваемый файл, но перед тем как будет сформирован ответ, PHP-процессор имеет возможность обработать PHP-код в данном файле.

В этой главе обсуждается работа написанных на PHP Web-программ. Можно запускать PHP-программы из командной строки, если имеется доступ к системе, на которой PHP установлен. Эта тема рассматривалась в главе 1.

Часто умалчивают об одном важном факте: в один момент времени может работать только один PHP-файл (поскольку только один файл может одновременно запрашиваться с сервера), а это означает, что даже если на сервере существует множество PHP-файлов, каждый из них должен функционировать как одна небольшая программа. Можно подключать к этой программе другие PHP-файлы, используя конструкции `include` или `require` (как это сделать, будет показано далее), однако на самом деле при этом код внешних файлов только копируется, а в качестве программы выполняется всего один файл. Это не является серьезным ограничением, но об этом стоит упомянуть, потому что все данные и переменные теряются каждый раз после обработки одной страницы и выполнения HTTP-запроса. Существует способ сохранения данных между запросами страницы (с помощью сессий), но нужно четко понимать идею “один файл — одна программа”.

Любую программу, которая взаимодействует с сервером, можно назвать клиентской программой, а любую программу, которая обслуживает клиентские программы, можно назвать сервером. На практике некоторые программы работают и как клиенты, и как серверы. Однако широко распространенные программы, такие как браузеры, FTP-программы и e-mail-программы, являются клиентскими и для выполнения своих функций они инициируют соединение с сервером. Серверы, к которым они обычно подключаются, являются Web-серверами, FTP-серверами и почтовыми серверами соответственно.

Клиент-серверная связь важна для PHP, так как весь PHP-код выполняется на сервере, тогда как HTML и/или JavaScript-код внутри Web-страниц передается клиенту нетронутым для обработки на стороне последнего. Технически можно отправлять браузеру для обработки нетронутый PHP-код, однако это не будет работать, так как у большинства пользователей нет возможности обработки PHP-кода в браузере. Одним из главных преимуществ, связанных с тем, что PHP-код обрабатывается на сервере, является то, что в отличие от использования JavaScript, конечный пользователь не имеет возможности просмотреть исходный код программы.

Web-соединения: Internet-протоколы и HTTP

Internet-протоколы определяют формат для всех Internet-соединений между компьютерами. Это означает, что для того чтобы один компьютер мог обмениваться данными с другим компьютером через Internet, оба компьютера должны использовать для этого обмена данными один и тот же язык. Для передачи файлов используется FTP (File Transfer Protocol — протокол передачи файлов), а для Web-коммуникаций применяется HTTP (HyperText Transfer Protocol — протокол передачи гипертекста).

В нескольких последующих разделах представлено введение в Internet-протоколы и описывается, как они способствуют обмену данными в Web-среде. Хорошее понимание происходящего между браузером и сервером является неотъемлемым для Web-программирования, потому что внутри запросов и ответов, которые пересылаются от клиента к серверу и обратно, имеется множество полезных данных, и эти данные можно перехватывать и использовать в программе.

TCP/IP

Internet обеспечивает обмен данными между многими взаимосвязанными Internet-узлами. Узлами в Internet являются все компьютеры или устройства, имеющие IP-адреса (четыре числа, разделенных точками, например, 64.71.134.49). Основным протоколом (а фактически набором сетевых протоколов), используемым для форматирования предназначенных к отправке данных, является TCP/IP (Transmission Control Protocol/Internet Protocol — протокол управления передачей/Internet-протокол). TCP/IP представляет собой просто метод описания *информационных пакетов* (индивидуально передаваемые через сеть блоки битов), так чтобы их можно было передавать по телефонным или кабельным сетям или T1-каналам от одного узла к другому, пока они не достигнут заданного пункта назначения.

Одним из преимуществ протокола TCP/IP является то, что он может очень быстро направить информацию по другому маршруту, если определенный узел или маршрут вышел из строя или работает недостаточно быстро. Когда пользователь дает браузеру команду получить какую-либо страницу, браузер, используя TCP, делит эту инструкцию на части (превращает в пакеты). TCP — транспортный протокол, обеспечивающий для данной инструкции надежный формат передачи. Данный протокол гарантирует, что все сообщение корректно разбирается и упаковывается для передачи (а также что оно корректно распаковывается и собирается в единое целое после того, как достигнет пункта назначения).

Прежде чем пакеты данных будут отправлены через сеть, необходима их адресация (пакеты должны включать в себя IP-адреса отправителя и получателя). Поэтому второй протокол, который называется протоколом передачи гипертекста (или HTTP), добавляет в них адресные метки, так чтобы TCP/IP “знал”, куда следует направлять данную информацию. HTTP — протокол, используемый World Wide Web при транспортировке данных от одной машины к другой — если URL-адрес предваряется последовательностью `http://`, то это означает, что используется протокол HTTP. TCP/IP можно представить себе как почтовую службу, которая осуществляет маршрутизацию и передачу писем, а HTTP — штампы и адреса на письмах (данных), которые гарантируют, что письма попадают куда следует.

Сообщения, передаваемые от браузера к Web-серверу, называются *HTTP-запросами*. Получив такой запрос (фактически запрос на какую-либо Web-страницу или файл), Web-сервер проверяет свои хранилища данных в поисках соответствующей страницы. Если страница найдена, то содержащийся в ней HTML-код разделяется сервером (с помощью TCP) на пакеты, которые адресуются браузеру (с помощью HTTP) и отправляются обратно через сеть. Если Web-сервер не может найти необходимую страницу, то в ответ он генерирует страницу, содержащую сообщение об ошибке (в данном случае Error 404: Page Not Found (Ошибка 404: Невозможно найти страницу)), разделяет ее на пакеты и отправляет браузеру. Сообщения, которые отправляются Web-сервером браузеру, называются *HTTP-ответами*.

HTTP-протокол

Рассмотрим работу протокола HTTP более подробно. Отправляемый Web-серверу запрос содержит не только необходимый URL-адрес. Как часть запроса передается множество дополнительных сведений. То же верно и для ответа — кроме самой страницы сервер отправляет обратно браузеру также дополнительную информацию.

Большая часть информации, которая передается внутри HTTP-сообщения, генерируется автоматически, пользователю нет необходимости непосредственно иметь

с ней дело, поэтому и разработчик не должен беспокоиться о передаче такой информации. И все же необходимо помнить, что эта дополнительная информация передается между машинами как часть HTTP-запросов и ответов, причем PHP-сценарий позволяет непосредственно влиять на точное содержание передаваемых данных.

Независимо от вида сообщения (запрос клиента или ответ сервера) каждое HTTP-сообщение имеет один и тот же формат, состоящий из трех разделов: строка запрос/ответ, HTTP-заголовок и HTTP-тело. Содержимое этих частей зависит от того, является ли сообщение запросом или ответом.

HTTP-запрос

HTTP-запрос, который браузер отправляет Web-серверу, содержит строку запроса, заголовок и тело. Ниже приведен пример строки запроса и заголовка.

```
GET /testpage.htm HTTP/1.1
Accept: */*
Accept-Language: en-us
Connection: Keep-Alive
Host: www.wrox.com
Referer: http://webdev.wrox.co.uk/books/SampleList.php?bookcode=3730
User-Agent: Mozilla (X11; I; Linux 2.0.32 i586)
```

Строка запроса

Первой строкой в каждом HTTP-запросе является *строка запроса (request line)*, содержащая три блока информации:

- HTTP-команда, которая называется метод (например, GET или POST);
- путь от сервера к запрашиваемому клиентом ресурсу;
- номер версии протокола HTTP (например, HTTP 1.1).

Ниже приведен пример строки запроса:

```
GET /testpage.htm HTTP/1.1
```

Метод используется для того, чтобы указать серверу, как обрабатывать данный запрос. В следующей таблице описывается три наиболее распространенных метода.

Метод	Описание
GET	Запрос на информацию, расположенную по определенному URL-адресу. Большинство запросов в Internet — GET-запросы (когда пользователь нажимает на гиперссылку, генерируется GET-запрос). Информация, запрашиваемая данным запросом, может быть любой — от HTML- или PHP-страницы, до вывода JavaScript или Perl-программы и т.д. Браузер может отправлять серверу некоторые ограниченные данные в форме расширения URL-строки
HEAD	То же, что и GET-метод, однако HEAD-метод запрашивает только HTTP-заголовок без данных
POST	Указывает на то, что данные отправляются серверу как часть HTTP-тела (например, поля формы). Эти данные затем передаются программе обработки данных на Web-сервере

Протокол HTTP поддерживает большое количество других методов, включая PUT, DELETE, TRACE, CONNECT и OPTIONS. Как правило, эти методы менее распространены, поэтому они выходят за рамки материала данной книги. Подробнее данные методы описаны в документе RFC 2068, который можно найти на сайте www.rfc.net.

Заголовок HTTP-запроса

Следующей порцией отправляемой информации является HTTP-заголовок. В нем содержатся сведения о том, документы каких типов клиент принимает от сервера, тип браузера, запросившего страницу, дата и общая конфигурационная информация. Заголовок HTTP-запроса содержит информацию, которая разделяется на три различных категории:

- ☐ общий заголовок: общая информация либо о клиенте, либо о сервере;
- ☐ заголовок объекта: информация о передаваемых между клиентом и сервером данных;
- ☐ запрос: информация о клиентской конфигурации и различных типах принимаемых документов.

Ниже приведен пример заголовка запроса:

```
Accept: */*
Accept-Language: en-us
Connection: Keep-Alive
Host: www.wrox.com
Referer: http://webdev.wrox.co.uk/books/SampleList.php?bookcode=3730
User-Agent: Mozilla (X11; I; Linux 2.0.32 i586)
```

Очевидно, что HTTP-заголовок составляется из нескольких строк, каждая из которых содержит описание блока информации в HTTP-заголовке, а также значение этого блока.

В HTTP-заголовок может входить множество различных строк и большинство из них является необязательными, поэтому HTTP требует указывать окончание передачи заголовочной информации. Для этого используется пустая строка.

Тело HTTP-запроса

Если в строке запроса HTTP используется метод POST, то в HTTP-теле содержатся любые данные, которые отправляются серверу, например, данные, введенные пользователем в HTML-форму (соответствующие примеры будут приведены далее). В противном случае тело HTTP-запроса пустое, как в данном примере.

HTTP-ответ

HTTP-ответ отправляется клиентскому браузеру от сервера и состоит из строки ответа, заголовка и тела. Ниже приведен пример строки ответа и заголовка.

```
HTTP/1.1 200 OK //строка состояния
Date: Fri, 31st Oct 2003, 18:14:33 GMT //общий заголовок
Server: Apache/1.3.12 (Unix) (SUSE/Linux) PHP/4.0.2 //заголовок ответа
Last-modified: Fri, 29th Oct 2003, 14:09:03 GMT //заголовок объекта
//пустая строка (конец заголовка)
```

Строка ответа

Строка ответа содержит только два блока информации:

- ☐ номер версии HTTP;
- ☐ код HTTP-запроса, который указывает на успешное или безуспешное выполнение данного запроса.

Например, такая строка ответа

```
HTTP/1.1 200 OK
```

возвращает код HTTP-состояния 200, соответствующий сообщению ОК, которое означает успешное выполнение запроса и то, что в ответе содержится затребованная

страница или данные от сервера. Если строка ответа содержит код HTTP-состояния 404 (уже упоминавшийся в этой главе), это значит, что Web-сервер не смог найти необходимый ресурс. Значения кодов ошибок представляют собой трехзначные числа, где первая цифра указывает класс ответа. Существует пять классов ответа, которые описаны в приведенной ниже таблице.

Класс кода	Описание
100–199	Информационный; указывает на то, что запрос в настоящее время обрабатывается
200–299	Отмечает, что Web-сервер успешно получил и выполнил данный запрос
300–399	Указывает на то, что запрос не был выполнен ввиду того, что необходимая информация была перемещена
400–499	Означает клиентскую ошибку (т.е. запрос был неполным, некорректным или невозможным)
500–599	Означает серверную ошибку (запрос был корректным, но сервер не смог его выполнить)

Заголовок ответа

Заголовок HTTP-ответа аналогичен рассмотренному ранее заголовку запроса. В HTTP-ответе заголовочная информация также разделяется на три типа:

- ☐ общий заголовок: содержит общую информацию либо о клиенте, либо о сервере;
- ☐ заголовок объекта: содержит информацию о данных, отправляемых между клиентом и сервером;
- ☐ ответ: содержит информацию об отправляющем данный ответ сервере, а также о возможности обработки этого ответа.

Данный заголовок состоит из множества строк, в нем также используется пустая строка, указывающая на его окончание. Ниже приведен пример заголовка; в комментариях указаны названия строк.

```
Date: Fri, 31st Oct 2003, 18:14:33 GMT //общий заголовок
Server: Apache/1.3.12 (Unix) (SUSE/Linux) PHP/4.0.2 //заголовок ответа
Last-modified: Fri, 29th Oct 2003, 14:09:03 GMT //заголовок объекта
//пустая строка (конец заголовка)
```

Назначение первой строки вполне очевидно. Во второй строке, `Server`, указывается программное обеспечение используемого Web-сервера. Так как в данном примере запрашивается файл, находящийся на данном Web-сервере, в третьей строке указывается время последней модификации запрашиваемой страницы.

Заголовок может содержать гораздо больше информации или информация может отличаться в зависимости от того, какой ресурс запрашивается. Более подробно различные типы заголовочной информации описаны в RFC 2068 (разделы 4.5, 7.1 и 7.2).

Тело ответа

Если запрос был успешным, то тело HTTP-ответа содержит HTML-код (вместе с каким-либо сценарием, выполняемым на стороне клиента), готовый для интерпретации браузером. Если запрос был безуспешным, то возвращается код ошибки.

Запуск PHP-сценариев посредством HTTP-запроса

Фактически все клиентские приложения (не только браузеры), способные отправлять HTTP-запросы Web-серверу, могут активизировать и запускать PHP-программы. На самом деле необязательно, чтобы файл отображал пользователю какой-либо вывод (т.е. разработчик не обязательно должен включать код в Web-страницу). Если Web-серверу отправляется корректно сформированный HTTP-запрос, обращающийся к файлу, содержащему PHP-код, и данный файл имеет соответствующее расширение, то PHP-программа запустится.

Web-сервер

Если Web-серверное программное обеспечение правильно настроено для работы в используемой на сервере операционной системе и для поддержки PHP, то можно ожидать, что HTTP-запросы для файлов, содержащих PHP-код, будут правильно обрабатываться, и что PHP-программы будут работать.

PHP-процессор

Язык PHP, по сути, состоит из функциональных модулей, языкового ядра (которое называется Zend Engine и к моменту написания имеет версию 2.0), а также интерфейса к Web-серверу. Этот интерфейс позволяет PHP обмениваться данными с Web-сервером. Функциональные модули снабжают PHP многими ценными возможностями, хотя Zend Engine (ядро языка) выполняет сложную работу по анализу, трансляции и выполнению поступающего кода (функции процессора Zend несколько шире, но идея заключается именно в этом). Важно отметить, что PHP-код транслируется в момент запуска PHP-программы на сервере. Это значительно упрощает работу программиста, устраняя необходимость заранее транслировать код специально для каждой машины, на которой данный код предположительно должен выполняться.

Использование переменных в PHP

Переменные используются практически в каждом языке программирования. Трудно представить себе возможность обработки данных без использования какой-либо формы переменных. Переменные являются одной из наиболее важных структур в программировании. Обычно они легко создаются и используются. В PHP переменные легко отличить, поскольку они начинаются со знака доллара (\$). Поэтому если в PHP-файле ввести знак доллара с последующим именем, то получится PHP-переменная.

В PHP переменные не обязательно объявлять и инициализировать, также нет необходимости устанавливать для них типы данных, поскольку PHP является языком с так называемой слабой типизацией (loosely typed language) (подробнее данная тема рассматривается в разделе “Строгая и слабая типизация данных” далее в настоящей главе). Переменная создается при включении ее имени в выражение и одновременно присвоении ей какого-либо значения. В первом примере данной главы переменной `$today'sdate` присваивалось значение текущей даты.

```
$today'sdate = date("m",time()) . "-" . date("d",time()) . "-" .  
date("Y",time());
```

Создание переменных

Можно отметить несколько специфических для любого языка вопросов, связанных с переменными, а именно:

- ☐ именование;
- ☐ тип данных;
- ☐ область видимости.

Рассмотрим их по порядку.

Именование переменных

Переменные предназначены для хранения данных с целью их обработки. Переменными они называются потому, что значения хранящихся в них данных могут изменяться в зависимости от их обработки.

Фактически переменная состоит из двух частей: имени переменной и значения переменной. Поскольку переменные в коде используются достаточно часто, лучше всего назначать переменным такие имена, которые можно легко понять и запомнить. Как и в других языках программирования, в РНР существует несколько правил, регламентирующих именование переменных:

- ☐ имя переменной начинается со знака доллара (\$);
- ☐ первым символом после знака доллара должна быть буква или символ подчеркивания;
- ☐ оставшимися символами имени могут быть буквы, цифры или символы подчеркивания без ограничений.

Имена переменных чувствительны к регистру символов (\$Variable и \$variable — две абсолютно разные переменные), а имена длиннее 30 символов непрактичны. В данной книге используется несколько полезных соглашений по написанию кода. Принятие какого-либо хорошего набора соглашений, скорее всего, покажется читателю стоящей идеей. Подробно соглашения по написанию сценариев рассматриваются в главе 6, однако по мере рассмотрения имен переменных все же будет представлено несколько соглашений по написанию кода.

Ниже приведен пример именованной переменной в РНР-программе:

```
$my_first_variable = 0;
```

Почему в данном случае переменной присваивается значение 0? В РНР нет особого смысла в указании имени переменной без присвоения ей тем или иным способом какого-либо значения, поскольку сам акт именованной переменной в программе создает эту переменную. Поэтому в данном случае создается переменная и ей присваивается значение 0. Естественно, можно создавать переменные и присваивать им любые значения, а не только нуль.

В некоторых языках программист ограничен, а иногда вообще не имеет возможности использовать переменную без явного предварительного ее объявления (создания). Однако РНР позволяет использовать переменные в любой точке программы, для этого нужно лишь указать их имена. И все же такое благо может оказаться обманчивым; если случайно по ошибке одно и то же имя переменной будет использовано дважды, то никакого сообщения об ошибке не будет, а в программе может возникнуть трудно обнаружимый дефект. Вместе с тем, в большинстве случаев такая возможность очень полезна и хорошо работает.

Типы данных

Другим вопросом при создании переменной является ее тип или тип хранящихся в ней данных. Что такое тип переменной? Тип переменной описывает тип хранящихся в ней данных. Читатели, которые уже работали с базами данных, вероятно, заметили, что полям в таблице базы данных часто назначается тип данных и этот тип данных позволяет различать строки, числа, даты булевы значения и т.д.

Тип данных какого-либо элемента определяет разновидность обработки, которую можно применить к элементу, а также объем необходимой для его хранения памяти. Например, если используется элемент данных строкового типа (`string`), имеющий значение 1995, и если язык программирования не способен автоматически интерпретировать типы данных и модифицировать их в соответствии с контекстом, в котором они используются, то невозможно прибавить значение 1995 к другой строке со значением 5 и ожидать при этом, что в результате получится число 2000. (Кстати, PHP способен автоматически интерпретировать и модифицировать типы данных; эта тема позднее рассматривается более подробно).

Вместо этого возникнет ошибка типа данных или, возможно, две строки будут объединены в одну строку и в результате получится значение 19955, т.е. совсем не то, что ожидалось.

Строгая и слабая типизация данных

В предыдущем примере были нарушены правила использования типов данных в языке со строгой типизацией, в результате чего возникала ошибка. Однако PHP является языком со слабой типизацией данных, и поэтому он уберегает программиста от ошибок такого рода, “понимая” намерения программиста и автоматически исправляя типы данных.

Понятие *строгой типизации* языка (*strongly typed language*) означает, что язык требует явного объявления типов переменных и сгенерирует ошибку, если попытаться использовать для переменных некорректные операторы, или выдаст некорректные результаты (т.е. не те, которые ожидаются). Языки со *слабой типизацией* (*loosely typed*) не требуют объявления типа переменной и автоматически конвертируют типы переменных в зависимости от контекста, в котором эти переменные используются, и операций над их значениями.

PHP — слабо типизированный язык, однако он позволяет при необходимости проверять типы данных, а также устанавливать и использовать типы данных. Несмотря на то, что явно объявлять переменные и назначать им типы данных не обязательно, существует возможность определить, какой тип данных назначен переменной в процессе обработки, а также в случае необходимости привести переменные к определенному типу данных.

Типы данных в PHP

Несмотря на слабую типизацию, PHP в действительности поддерживает многие распространенные простые и структурированные типы данных. Простые типы данных содержат диапазон значений, которые можно упорядочить в одном измерении (строки, числа, булевы значения и др.), а в число структурированных типов данных включаются массивы и объекты. В PHP имеется восемь простых типов, которые описаны в следующей таблице.

Тип данных	Описание
Boolean (булев)	Скалярный тип; либо True либо False
Integer (целый)	Скалярный тип; целое число
Float (вещественный)	Скалярный тип; возможно, число с десятичными разрядами
String (строковый)	Скалярный тип; последовательность символов
Array (массив)	Сложный тип; упорядоченная таблица (содержащая имена и связанные с ними значения)
Object (объект)	Сложный тип; тип, который может содержать свойства и методы
Resource (ресурс)	Специальный тип; содержит ссылку на внешний ресурс, например, дескриптор открытого файла
NULL (нуль)	Специальный тип; в качестве значения может содержать только NULL, это означает, что переменная явно не содержит никакого значения

Программисты используют такие понятия, как скалярный, сложный и специальный, для обозначения характеристик типов данных. “Скалярный” означает, что значения такого типа данных могут быть упорядоченными по какой-либо шкале. Например, числа упорядочиваются от наименьшего к наибольшему, а символы упорядочиваются по алфавиту. “Сложный” означает, что данные состоят из множества элементов; например, массивы содержат индексы и связанные с ними значения. “Специальный” означает специальное число или значение, имеющее важный смысл для приложения, например, дескриптор файла.

Массивы описываются далее в данной главе, в последующих главах подробно рассматриваются объекты (и новые объектно-ориентированные свойства PHP).

Преобразование типов данных в PHP

В обычных обстоятельствах программисту редко приходится преобразовывать значение переменной из одного типа в другой. Однако иногда это полезно, например, когда требуется убедиться, что используется определенный тип данных, или при подготовке вывода, который будет использоваться другой программой. В PHP включены встроенные функции *приведения (casting)* (или установки) типов.

PHP-функцию `gettype()` можно использовать для определения текущего типа переменной, а функция `settype()` преобразует переменную в заданный тип. Например, в приведенном ниже коде в качестве значения переменной устанавливается целое число, затем тип меняется на строковый, при этом каждый раз распечатывается тип данных. Конкретные символы, составляющие значение, остаются одними и теми же, изменяется лишь тип данных:

```
$my_var = 1995; // $my-var содержит числовое значение
echo "Текущий тип переменной " . gettype($my_var) . "<br>";
$my_var = settype($my_var, "string");
// $my_var теперь имеет строковый тип
echo "Текущий тип переменной " . gettype($my_var);
```

PHP-функция `gettype()` возвращает строковое значение, описывающее тип переданной функции переменной (например, `string`, `integer` и т.д.). В PHP также имеются функции, которые проверяют определенный тип данных, например, `is_string`, `is_int` и др. Эти функции следует применять всегда, когда требуется проверить определенный тип, не сравнивая при этом строку, возвращенную функцией `gettype()` (например, `integer`), с предполагаемой строкой (также `integer`).

Область видимости переменной

В рассмотренных ранее примерах кода подразумевалось, что если создается переменная путем назначения ей имени, а затем данной переменной присваивается значение, то эту переменную можно использовать как угодно долго и когда угодно для передачи ее в любые функции обработки данных. В некоторой степени так оно и есть, однако имеется ряд ограничений. Эти ограничения связаны с *областью видимости* (*scope*) переменной.

Область видимости переменной соответствует пространству в коде, где эта переменная (а фактически ее значение) доступна для манипуляции. Как уже отмечалось, большинство переменных доступны в любой точке РНР-программы, однако в описанной формально функции (подробное обсуждение функций приведено в главе 6) переменные являются *локальными*, т.е. они распознаются и используются в пределах данной функции. Ниже приведен пример простой функции:

```
$my_data = "Внешние данные";
function send_data() {
    $my_data = "Внутренние данные";
    echo $my_data;
}
send_data(); //отправляет внутренние данные пользователю
echo $my_data; //отправляет внешние данные пользователю
```

Данная функция просто возвращает данные. При вызове функции (предпоследняя строка кода) пользователю отправляется строка Внутренние данные.

Однако если переменная `$my_data` выводится вне функции, как показано в последней строке кода, пользователю отправляется строка Внешние данные, поскольку переменная `$my_data` вне функции и переменная `$my_data` внутри функции представляют собой разные переменные, несмотря на то, что они имеют одинаковые имена. Область видимости переменной `$my_data` внутри функции называется локальной для данной функции. Кроме того, как только функция завершает свою работу, ее внутренняя переменная `$my_data` уничтожается, а ее значение теряется.

Ключевое слово `global`

Существует способ получить доступ к внешним переменным изнутри функции. Если переменная объявляется с ключевым словом `global`, то она будет доступна внутри функции, как показано в следующем примере (в главе 6 использование ключевого слова `global` описано более подробно):

```
$my_data = "Внешние данные";
function send_data() {
    global $my_data;
    echo $my_data;
}
send_data(); //отправляет пользователю внешние данные
echo $my_data; //также отправляет пользователю внешние данные
```

Статические переменные

Если при создании переменной внутри функции (т.е. область действия переменной локальна для данной функции и обычно теряется после завершения работы функции) используется ключевое слово `static`, то эта переменная и ее значение будет сохраняться между вызовами функции. Подобная возможность полезна в определенных ситуациях, когда желательно знать, сколько раз была вызвана данная функция. Например,

предположим, что требуется запретить вызывать какую-либо функцию для возвращения записей из базы данных более 100 раз. Добиться этого можно, задав статическую переменную, значение которой увеличивается на единицу при каждом вызове функции. Ниже приведен соответствующий пример (с небольшим количеством псевдокода):

```
function get_record() {
    static $counter = 0;
    $counter++;
    //проверить условие $counter < 100
    //если условие выполняется, то запустить код,
    //извлекающий запись из базы данных
    //если $counter = или > 100, то echo "Записей больше нет"
}
```

Каждый раз при вызове функции значение переменной `$counter` увеличивается на единицу и сохраняется до следующего вызова функции.

Определение констант

Кроме переменных в РНР можно определять другой вид контейнеров значений — *константы* (*constants*). Константы, как очевидно из названия, могут быть определены (с помощью функции `define()`) в РНР-программе только однажды, а их значения невозможно изменять и они не могут быть неопределенными. Константы отличаются от переменных тем, что в начале имен у них нет символа доллара, а в остальном они именуются так же, как и переменные.

Константы могут содержать только скалярные значения, например, булевы или целые числа, числа с плавающей точкой и строки (но не массивы или объекты). К константам можно обращаться из любой точки программы, не обращая внимания на область действия, а их имена чувствительны к регистру символов. Чтобы определить константу, используется функция `define()`, которой в качестве параметров передается имя константы и ее значение, см. пример ниже:

```
define("my_constant", "1995");
//константа my-constant всегда содержит строковое значение "1995"
echo my_constant; //отправляет пользователю строку "1995"
// (обратите внимание, строку, а не целое число)
```

Операторы и выражения

Обработка данных в РНР, как и в других языках программирования, осуществляется с помощью операторов и выражений. Операторы представляют собой символы, которые указывают РНР, какую операцию необходимо выполнить, а выражения являются отдельными группами переменных и операторов, предназначенных для вычисления результатов.

РНР-операторы

В РНР значение переменной присваивается с помощью знака равенства:

```
$my_data = "Hello";
```

Знак равенства является *оператором*. Операторы используются для выполнения обработки значений переменных. В данном случае знак равенства называется *оператором присваивания*, так как с его помощью только что созданной переменной присваивается строковое значение.

В большинстве языков программирования используется много операторов; некоторые из них выполняют арифметические действия, как в простых уравнениях, другие оперируют строками или датами, а третьи осуществляют другие функции. Все они производят обработку значений переменных.

Операторы, которым нужен только один операнд, называются унарными; например, оператор ++ может присоединяться справа к имени переменной (операнду) для увеличения ее значения на единицу. Поскольку этот оператор может быть помещен как перед именем переменной, так и после него, говорят, что он допускает префиксную и постфиксную нотацию.

Операторы, которым требуется два операнда, называются бинарными; знак равенства в выражении `$my_data = "Hello"` представляет собой бинарный оператор. Данный оператор помещается между операндами, поэтому такая форма записи называется инфиксной нотацией.

В некоторых языках имеются тернарные операторы. Например, РНР позволяет использовать оператор `?`, который представляет собой сокращенное выражение `if`. Для его использования сначала пишется выражение, за которым следует знак вопроса, а затем два возможных результата, разделенных двоеточием (например, запись “выражение ? результат01 : результат02”, означает “если выражение справедливо, то результатом является результат01, иначе результат02”). Поскольку оператор помещается между операндами, такая форма записи также называется инфиксной нотацией.

РНР-выражения

Выражения представляют собой любой код, который вычисляется в значение. Присвоение переменной значения само по себе является выражением, хотя часто выражения рассматриваются как уравнения (например, `$a = $b + $c`, где `$b + $c` — выражение).

Следовательно, `$a = 5` — выражение, поскольку оно вычисляется в значение 5. Известно, что запись вида `$a = $b + $c` означает сложение `$b` и `$c`, а затем присвоение результирующего значения переменной `$a`. Можно создавать выражения произвольной сложности и для получения результатов применять любые операторы к любым соответствующим значениям. По сути, основная часть РНР-функциональности связана с обработкой (вычислением) выражений.

Одним из ключевых моментов при вычислении выражения, особенно сложного выражения, является приоритет операторов (как и в арифметике или математике часто имеет значение то, какая часть выражения вычисляется первой). Существует приоритет операций по умолчанию, а управлять приоритетом можно с помощью круглых скобок. Например, поскольку приоритет оператора умножения (`*`) выше, чем у оператора сложения (`+`), то выражение `2 + 2 * 12` будет равно 26 (часть `2 * 12` вычисляется первой по умолчанию, а затем к произведению прибавляется 2), тогда как результатом выражения `(2 + 2) * 12` является 48 (скобки приводят к тому, что сначала выполняется сложение, а затем полученная сумма умножается на 12).

Типы операторов

В PHP доступны следующие типы операторов:

Тип	Описание
Арифметические	Выполняют обычные арифметические операции, такие как сложение и вычитание
Присваивания	Присвоение значения переменной
Битовые	Выполняют операции над отдельными битами целого числа
Сравнения	Сравнивают значения методом Буля (возвращается true или false)
Операторы контроля ошибок	Влияют на обработку ошибок (в PHP 5 появилось несколько новых операторов)
Выполнения	Приводят к выполнению команд, как если бы они были shell-командами
Инкрементные/декрементные	Инкрементируют или декрементируют переменные
Логические	Булевы операторы, такие как И, ИЛИ и НЕ, которые можно использовать для включения или исключения (эта тема подробнее рассматривается в главе 4)
Строковые	Выполняют конкатенацию (объединение) строк
Для работы с массивами	Выполняют операции над массивами (например, добавление значений или разделение массива)

Справочную информацию по каждому оператору можно получить на сайте www.php.net (перечень операторов дан в разделе документации). По мере необходимости в данной книге рассматривается некоторая специфика используемых операторов.

Строковые операторы и функции

Существует только один строковый оператор: точка (.). Вместе с тем в PHP имеется множество строковых функций, которые позволяют эффективно манипулировать строками. В последующих разделах обсуждается оператор конкатенации и работа нескольких строковых функций.

Использование оператора конкатенации

Оператор конкатенации (.) может использоваться между строковыми значениями с целью их объединения. Ниже показан пример конкатенации в PHP-программе:

```
<?php
$first_name = "Иван";
$last_name = "Петров";
$whole_name = $first_name . " " . $last_name;
echo "Имя плюс фамилия = <b>$whole_name</b>";
?>
```

Следует отметить, что в значение `$whole_name` между значениями переменных `$first_name` и `$last_name` путем конкатенации добавляется пробел " " (пробел между кавычками). Пробелы перед и после каждого оператора конкатенации необязательны, но они позволяют сделать код более простым для чтения. Следующий код работает точно так же:

```
$whole_name = $first_name." ".$last_name;
```

Чтобы сделать результат более читабельным при отображении на Web-странице, в возвращаемый ответ добавляются HTML-теги (теги `` и ``, выделяющие текст жирным шрифтом):

```
echo "Имя плюс фамилия = <b>$whole_name</b>";
```

Если нет необходимости включать в HTML-код специальные символы, такие как кавычки, то можно просто вставлять HTML-теги в текст, который впоследствии будет правильно отформатирован браузером при отображении страницы. В данном случае переменная `$whole_name` была вставлена непосредственно в строку. Во многих языках программирования это невозможно, но PHP достаточно развитый язык и автоматически использует значение переменной, а не ее имя, когда она помещена в строку. Чтобы отобразить в строке имя переменной, необходимо экранировать знак доллара (добавив перед ним символ обратной косой черты):

```
echo "First name plus last name = <b> \ $whole_name</b>";
```

Использование функции `strlen()`

Функция `strlen()` определяет длину строки. Она подсчитывает и возвращает количество всех символов в строке. В следующем примере общее число символов записывается в переменную с именем `$string_length`:

```
$string_length = strlen($whole_name);
```

Примечательно, что использование оператора конкатенации для объединения строки (например, “длина строки с именем”) с числовым значением (например, длиной строки, содержащейся в `$string_length`) приводит к тому, что весь результат будет иметь строковый тип данных.

Эта функция полезна для подсчета количества символов в строке, например, при проверке данных, которые должны записываться в базу данных.

Использование функции `strstr()`

Функция `strstr()` извлекает любую часть строки, которая находится после первого вхождения определенного символа, или строку внутри другой строки. В следующем примере в значении переменной `$whole_name` (Иван Петров) функция `strstr()` ищет первое вхождение символа пробела, а затем возвращает всю оставшуюся после него часть строки. Дополнительные вхождения искомой подстроки внутри строки ничего не меняют. Если искомая подстрока не найдена внутри просматриваемой строки, то функция возвращает значение `FALSE`.

```
$part_after_space = strstr($whole_name, " ");  
echo "Часть строки после пробела - <b>" .  
$part_after_space . "</b>";
```

Использование функции `strpos()`

Функция `strpos()` используется для того, чтобы определить, существует ли в просматриваемой строке искомая подстрока. Данная функция возвращает числовое значение, представляющее собой позицию, с которой начинается искомая подстрока (если такая подстрока есть). В следующем примере поиск подстроки `a` в значении переменной `$whole_name` (Иван Петров) возвращает номер позиции — 2. Можно было бы предположить, что возвращаемым значением будет 3, так как буква `a` в имени стоит третьей, однако, как и во многих других языках программирования, в PHP часто используется нумерация значений, начиная с 0, а не с 1, поэтому возвращается позиция 2.

```
$letter_position = strpos($whole_name, "o");
echo "Позиция буквы "a": <b>" .
$letter_position . "</b>";
```

В данном случае PHP изучает строку Иван Петров так, как если бы она была массивом символов (на самом деле так оно и есть в PHP), а затем для возвращения позиции любого символа использует индексы массива (0,1,2,3,4 и т.д.). (Массивы рассматриваются далее в настоящей главе).

Использование функции chr()

Функция chr() возвращает строковый символ для переданного ей в качестве аргумента десятичного ASCII-значения. Таблицы ASCII-символов можно найти в Internet, и часто они весьма удобны в использовании, особенно для специальных символов. Например, ASCII-код для символа перевода строки равен 10, а для возврата каретки 13. В действительности таких символов на клавиатуре нет, и чтобы включить их в строку, необходимо воспользоваться данной функцией (chr(10) и chr(13)) и представленные ею соответствующие символы будут вставлены в строку.

Практика Работа со строками

Создадим простую PHP-программу, демонстрирующую использование операторов в выражениях с переменными. Данная программа должна продемонстрировать работу со строками. В ней используется строковый оператор, точка (.) и несколько встроенных строковых PHP-функций. Ниже описана последовательность действий для создания данной программы.

1. Создать файл в любом текстовом редакторе и сохранить его как `working_with_strings.php`. Файл необходимо поместить в каталог, поддерживаемый Web-сервером (если Web-сервер работает на локальной машине) или загрузить в соответствующий каталог удаленного Web-сервера (и загружать его после каждых изменений).
2. Ввести в данный файл следующий код (фрагменты PHP-кода выделены серым цветом):

```
<html>
<head>
<title>PHP5 для начинающих</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body bgcolor="#FFFFFF">
<table width="100%" border="1">
  <tr>
    <td width="49%"><font face="Arial, Helvetica, sans-serif"><b>Работа
со строками</b>
    </font></td>
    <td width="51%">&nbsp;</td>
  </tr>
  <tr>
    <td width="49%"><font face="Arial, Helvetica, sans-serif"
size="-1">Использование конкатенации - оператор точка </font></td>
    <td width="51%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$first_name = "Иван";
$last_name = "Петров";
$whole_name = $first_name . " " . $last_name;
echo "Имя плюс фамилия = <b>$whole_name</b>";
?>
```

```

</font></td>
</tr>
<tr>    <td width="49%"><font face="Arial, Helvetica, sans-serif" size="-1">
Определение
        длины строки - использование <b>strlen()</b></font></td>
        <td width="51%"><font face="Arial, Helvetica, sans-serif" size="-1">

<?php
$string_length = strlen($whole_name);
echo "Длина строки <b>" . $string_length . "</b>";
?>

</font></td>
</tr>
<tr>
    <td width="49%"><font face="Arial, Helvetica, sans-serif" size="-1">Получение
        подстроки - использование <b>strstr()</b></font></td>
        <td width="51%"><font face="Arial, Helvetica, sans-serif" size="-1">

<?php
$part_after_space = strstr($whole_name, " ");
echo "Часть строки после пробела - <b>" . $part_after_space .
    "</b>";
?>

</font></td>
</tr>
<tr>
    <td width="49%"><font face="Arial, Helvetica, sans-serif" size="-1">Определение
        позиции начала подстроки - использование <b>strpos()</b></font></td>
        <td width="51%"><font face="Arial, Helvetica, sans-serif" size="-1">

<?php
$letter_position = strpos($whole_name, "a");
echo "Позиция буквы &quot;a&quot;;: <b>" . $letter_position .
    "</b>";
?>

</font></td>
</tr>
<tr>
    <td width="49%"><font face="Arial, Helvetica, sans-serif" size="-1">Возвращение
        символа по его ASCII-значению - использование <b>chr()</b></font></td>
        <td width="51%"><font face="Arial, Helvetica, sans-serif" size="-1">

<?php
$ascii_character_returned = chr(224);
echo "Символ, соответствующий ASCII-коду 224: <b>"
    . $ascii_character_returned . "</b>";
?>

</font></td>
</tr>
</table>
</body>
</html>

```

3. Сохранить файл, при необходимости выгрузить его на сервер, а затем вызывать в браузере. Результат представлен на рис. 2.2.

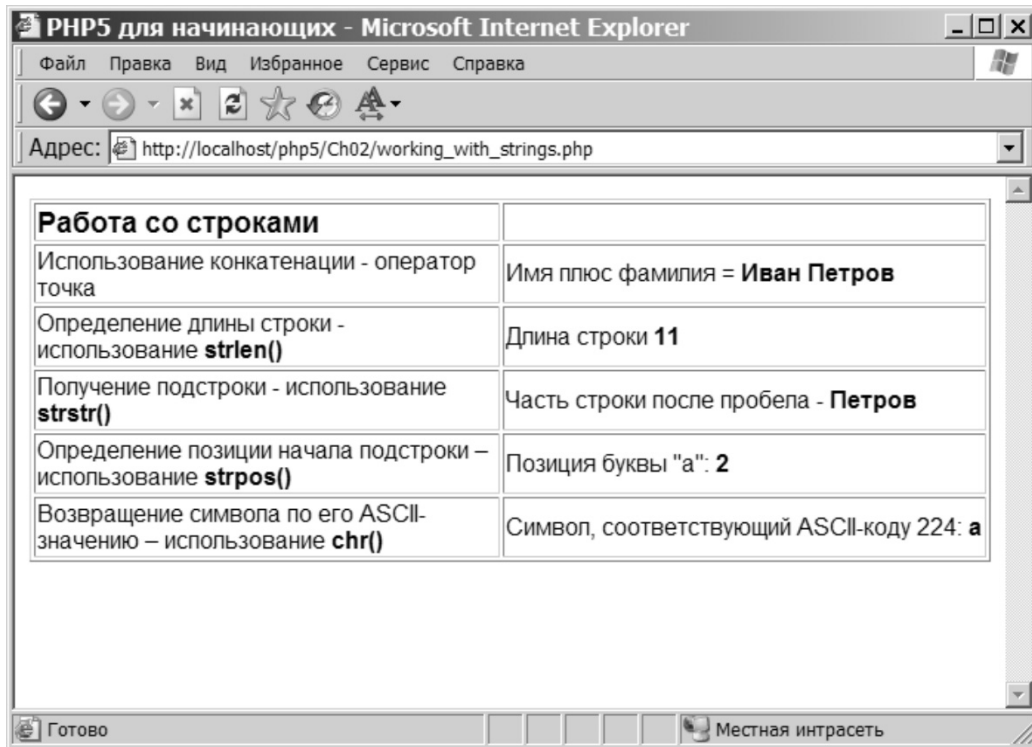


Рис. 2.2.

Как это работает

Только что созданная программа внедряется в завершенную Web-страницу. Внутри HTML-элемента `<body>` фрагменты программы содержатся в HTML-таблице исключительно с целью удобства чтения. Когда браузер запрашивает у Web-сервера файл данной страницы, PHP-код проходит синтаксический анализ и выполняется, а результат помещается в HTML-поток, возвращаемый браузеру.

Для объединения строковых значений (включая пробелы) используется оператор конкатенации (`.`):

```
<?php
$first_name = "Иван";
$last_name = "Петров";
$whole_name = $first_name . " " . $last_name;
echo "Имя плюс фамилия = <b>$whole_name</b>";
?>
```

Функция `strlen()` определяет и возвращает длину строки. В данной программе эта функция возвращает число, соответствующее количеству символов в строковом значении переменной `$whole_name`.

Функция `strstr()` определяет и возвращает любую часть строки, которая находится после первого вхождения определенного символа или подстроки в строке. В данной программе эта функция просматривает значение переменной `$whole_name` ("Иван Петров") до тех пор, пока не найдет первое вхождение символа пробела, а затем возвращает оставшуюся после пробела часть строки.

Функция `strpos()` определяет и возвращает число, соответствующее позиции символа в строке. В данном случае поиск буквы `a` в значении переменной `$whole_name` возвращает позицию 2. (Напомним, что значения начинаются с 0, поэтому третьей по счету позицией является позиция номер 2.)

Функция `chr()` возвращает строковый символ, соответствующий десятичному ASCII-значению, переданному функции в качестве аргумента. В данной программе возвращается строковый символ для ASCII-значения 224.

Арифметические операции в PHP

В PHP арифметические операторы (“плюс”, “минус” и т.д.) работают так же, как от них следует ожидать, позволяя создавать выражения как простые уравнения. Например, в выражении `$c = $a + $b` складываются значения переменных `$a` и `$b`, а затем результат присваивается переменной `$c`. (Оператор присваивания = полностью отличается от операторов сравнения `==` и `===`, которые рассматриваются в главе 4.)

Кроме того, как и в обычных уравнениях, имеет значение приоритет операторов; повлиять на него можно с помощью круглых скобок, см. пример ниже:

```
<?php
$first_number = 20;
$second_number = 30;
$third_number = 3;
$fourth_number = 2;
$total = $first_number * $second_number / $third_number + $fourth_number;
$total2 = $first_number * $second_number / ($third_number + $fourth_number);
echo "Двадцать умножить на тридцать, разделить на три, прибавить два равно
<b>$total</b><br>";
echo "Двадцать умножить на тридцать и разделить на (три плюс два) равно
<b>$total2</b>";
?>
```

Если запустить программу, то различие, вызванное использованием скобок, будет очевидным, поскольку первый оператор `echo` выведет значение 202, а второй — 120.

Специальные операторы присваивания

Знак равенства можно комбинировать с другими операторами. Это позволяет создавать специальные операторы присваивания, которые упрощают написание некоторых выражений. Специальные операторы присваивания (такие как `+=`, `-=` и др.) позволяют использовать стенографический метод для выполнения обычных арифметических операторов; при этом не требуется несколько раз писать имя одной и той же переменной. Например, можно использовать следующий код:

```
$first_number += $second_number;
```

вместо такого кода:

```
$first_number = $first_number + $second_number;
```

Такой подход также применим для других видов операторов. Например, оператор конкатенации можно комбинировать со знаком равенства (`.=`), так чтобы к текущему значению левой стороны выражения присоединялось значение правой стороны, см. пример ниже:

```
$a = "Начало и ";
$b = "конец предложения.";
$a .= $b //в результате значением $a является строка "Начало и конец предложения."
```

Основные арифметические операторы, строковый и битовые операторы поддерживают такое комбинирование. Более подробная информация о комбинировемых операторах приведена на официальном Web-сайте PHP.

Использование инкрементных и декрементных операторов

Очень часто возникает необходимость многократно прибавлять к числу или вычитать из него одно и то же число. Для решения подобных задач существуют специальные операторы: инкрементные и декрементные. Они записываются соответственно как два знака “плюс” или два знака “минус” перед или после имени переменной, например:

```
$a = ++$a; //прибавляет единицу к $a, а затем возвращает результат
$a = $a++; //возвращает $a, а затем прибавляет к $a единицу
$b = --$b; //вычитает единицу из $b, а затем возвращает результат
$b = $b--; //возвращает $b, а затем вычитает из $b единицу.
```

Расположение данных операторов имеет большое значение. Оператор, предшествующий имени переменной, приводит к тому, что операция (сложение или вычитание единицы) выполняется перед тем, как возвращается значение переменной; оператор, следующий после имени переменной, возвращает текущее значение переменной, а после этого выполняет операцию.

Следует отметить, что инкрементные и декрементные операторы можно (ограниченно) использовать и для символов. Например, можно “прибавить” единицу к символу В и возвращаемым значением будет символ С. Однако вычитать единицу (декрементировать) из символьных значений нельзя.

Использование математических PHP-функций

В PHP встроены многие распространенные математические функции. Некоторые из них требуют указания аргументов, другие не принимают аргументы, а для третьих аргументы являются необязательными. Например, можно использовать функцию `floor()` для округления числа в меньшую сторону независимо от величины его дробной части. Однако данной функции необходимо передать аргумент. В противном случае, что она будет округлять? Аргумент представляет собой первоначальное значение, которое необходимо округлить. Например, чтобы округлить число 100.0, необходимо использовать следующий код:

```
$a = 100.01;
$floor_a = floor($a);
```

С другой стороны такие функции, как `pi()` и `rand()`, не требуют аргументов. Функция `pi()` возвращает число π до 14 знаков после запятой (по умолчанию 14 знаков, но фактическая точность зависит от параметра директивы `precision` в файле `php.ini`). Функция `rand` генерирует (псевдо) случайное число в диапазоне от 1 до `RAND_MAX` (максимальное число, разное для разных операционных систем), если ей не переданы аргументы, ограничивающие диапазон, из которого функция может выбрать случайное число.

Практика Работа с числами

Следующая программа демонстрирует работу с числами. В ней используются как уже знакомые читателю, так и новые операции, а также некоторые доступные в PHP операторы и встроенные функции. В данном случае PHP-код также внедряется в HTML-код с тем, чтобы можно было вывести результаты на Web-странице.

1. Откройте HTML-редактор и введите в него следующий код. (Хотя ввод всего кода будет хорошим упражнением, вместо этого можно загрузить файл `working_with_numbers.php` с Web-сайта данной книги).

```
<html>
<head>
<title>PHP5 для начинающих</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>

<body bgcolor="#FFFFFF">
<table width="100%" border="1">
  <tr>
    <td width="57%"><font face="Arial, Helvetica, sans-serif"><b>Работа с
      числами</b></font></td>
    <td width="43%">&nbsp;</td>
  </tr>
  <tr>
    <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
      Использование оператора сложения (+)</font></td>
    <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$first_number = 20;
$second_number = 30;
$total = $first_number + $second_number;
echo " Двадцать плюс тридцать равно <b>$total</b>";
?>
```

```
</font></td>
</tr>
<tr>
  <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
    Использование инкрементного оператора (++)</font></td>
  <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$first_number = 20;
$first_number = ++$first_number;
echo " Двадцать, инкрементированное на единицу, равно <b>$first_number</b>";
?>
```

```
</font></td>
</tr>
<tr>
  <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
    Использование операторов умножения и деления (* и /)</font></td>
  <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$first_number = 20;
$second_number = 30;
$third_number = 3;
$fourth_number = 2;
$total = $first_number * $second_number / $third_number + $fourth_number;
$total2 = $first_number * $second_number / ($third_number + $fourth_number);
echo "Двадцать умножить на тридцать, разделить на три, прибавить два равно
  <b>$total</b><br>";
echo "Двадцать умножить на тридцать, разделить на (три плюс два) равно
  <b>$total2</b>";
?>
```

```
</font></td>
</tr>
```

```

<tr>
  <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
    Специальные операторы присваивания — использование += и *=
  </font></td>
  <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
    <?php
      $first_number = 20;
      $second_number = 30;
      $total = $first_number += $second_number;
      $total2 = $first_number *= $second_number;
      echo " Двадцать += тридцать равно <b>$total</b><br>";
      echo " Предыдущий результат *= тридцать равно <b>$total2</b>";
    ?>

  </font></td>
</tr>
<tr>
  <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
    Получение абсолютного значения числа — использование функции abs()
  </font></td>
  <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
    <?php
      $first_number = -2.7;
      echo " Абсолютное значение -2,7 равно <b>" . abs($first_number) . "</b>";
    ?>

  </font></td>
</tr>
<tr>
  <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
    Преобразование двоичного числа в десятичное — использование функции
    bindec()</font></td>
  <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
    <?php
      $binary_number = 10101111;
      $decimal_number = bindec($binary_number);
      echo "Десятичным эквивалентом двоичного числа 10101111 является число
        <b>$decimal_number</b>";
    ?>

  </font></td>
</tr>
<tr>
  <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
    Округление чисел в большую и меньшую сторону — использование функций
    ceil() и floor()</font></td>
  <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
    <?php
      $first_number = 2.4;
      echo "Число 2,4, округленное в большую сторону, равно <b>"
        . ceil($first_number)
        . "</b>, в меньшую сторону — <b>" . floor($first_number)
        . "</b>";
    ?>
  </font></td>
</tr>

```

```

</font></td>
</tr>
<tr>
  <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
    Определение максимального или минимального значения – использование
    функций max() и min()</font></td>
  <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">

<?php
$max_value = max(2,3,4);
$min_value = min(2,3,4);
echo "Максимальным числом из 2,3,4 является <b>"
    . $max_value . "</b>,"
    а минимальным – <b>" . $min_value . "</b>";
?>

</font></td>
</tr>
<tr>
  <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
    Получение числа пи – использование функции pi()</font></td>
  <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">

<?php
echo " Число пи равно <b>" . pi() . "</b>";
?>

</font></td>
</tr>
<tr>
  <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
    Получение случайного числа – использование функции rand()</font></td>
  <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">

<?php
echo " Случайное число: <b>" . rand() . "</b>";
?>

</font></td>
</tr>
<tr>
  <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
    Извлечение квадратного корня – использование функции sqrt()</font></td>
  <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">

<?php
$first_number = 20;
echo " Квадратный корень из двадцати равен <b>"
    . sqrt($first_number) . "</b>";
?>

</font></td>
</tr>
</table>
</body>
</html>

```

2. Сохраните файл как `working_with_numbers.php`, в случае необходимости выгрузите его на сервер и отобразите в браузере. Результат показан на рис. 2.3.

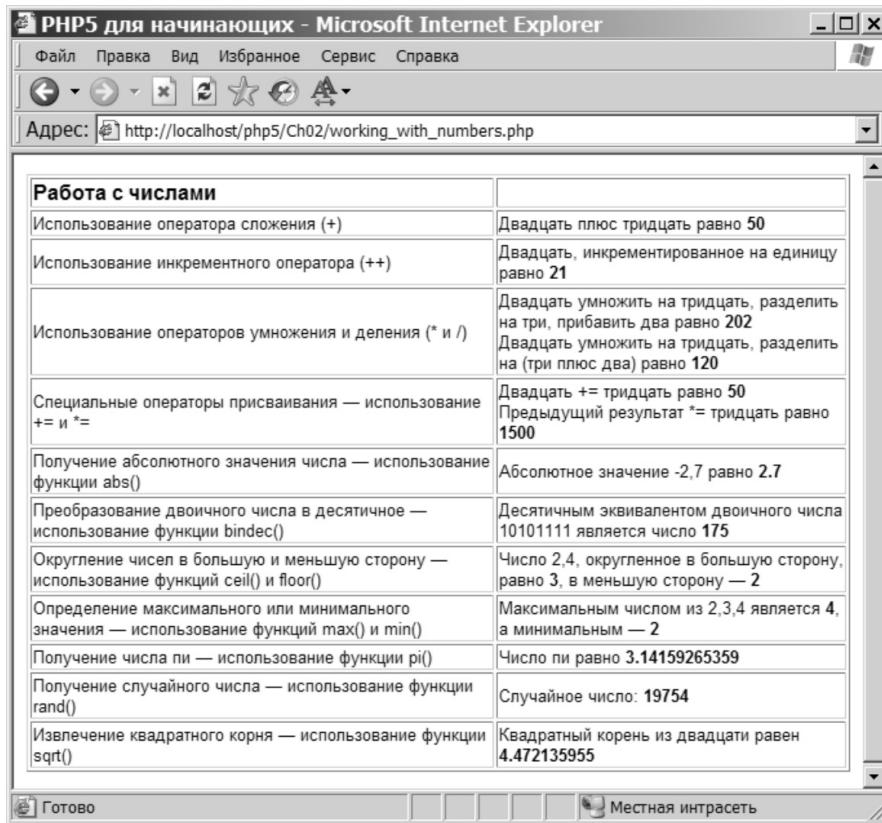


Рис. 2.3.

Как это работает

В данной программе используется тот же формат отображения, что и в предыдущем примере. В программе показаны некоторые простые вычисления с помощью операторов, простейшее использование встроенных функций, а также результаты работы нескольких встроенных функций, не принимающих аргументов (например, `pi()`).

В коде задавались некоторые значения, а затем производилась их обработка при различном расположении круглых скобок. Это иллюстрирует приоритетность операторов.

```
<?php
$first_number = 20;
$second_number = 30;
$third_number = 3;
$fourth_number = 2;
$total = $first_number * $second_number / $third_number + $fourth_number;
$total2 = $first_number * $second_number / ($third_number + $fourth_number);
echo "Двадцать умножить на тридцать, разделить на три, прибавить два равно
<b>$total</b><br>";
echo "Двадцать умножить на тридцать, разделить на (три плюс два) равно
<b>$total2</b>";
?>
```

Затем использовались инкрементный и декрементный операторы для прибавления и вычитания единицы из численного значения, содержащегося в переменных `$a` и `$b`.

```
$a = ++$a; //прибавляет единицу к $a, а затем возвращает результат
$a = $a++; //возвращает $a, а затем прибавляет к $a единицу
$b = --$b; //вычитает единицу из $b, а затем возвращает результат
$b = $b--; //возвращает $b, а затем вычитает из $b единицу
```

Пример использования функций `pi()` и `rand()` показывает, как заставить РНР генерировать значение для числа π или случайные значения. Все что для этого требуется — просто написать имя функции без параметров в скобках (и, конечно, использовать знак равенства для присвоения значения переменной).

Массивы

Массивы представляют собой переменные типа `array` (`array` — в данном случае тип данных), однако это весьма специфические и эффективные переменные, которые заслуживают отдельного раздела в книге.

Строго говоря, массивы представляют собой списки, состоящие из *ключей* (*keys*) (индексов) и *значений* (*values*), которые содержатся в каждом *элементе* (*element*) массива. Элементами являются контейнеры значений в массиве. Элементы можно представить как отдельные переменные, состоящие из пар имя/значение.

И хотя в литературе часто отмечается, что массивы могут быть весьма сложными и трудными для понимания, в них достаточно просто разобраться: массивами являются переменные с множеством контейнеров значений и несколькими способами доступа к определенному значению. Они в некотором смысле похожи на реляционные мини-базы данных, которые являются динамическими, поскольку существуют в памяти до тех пор, пока выполняется текущая программа (если между запросами страницы они не сохраняются в сеансах или в реальных базах данных). Имена массивов подобны именам таблиц в базе данных, а элемент массива, содержащий другой массив, подобен связанной таблице в базе данных. Подобная аналогия не претендует на абсолютную точность, но может оказаться весьма полезной, раскрывающей механизм создания или доступа к массивам и их значениям.

Индексы массивов

Функция (а фактически конструкция языка) `array()` используется для создания массивов и принимает в качестве аргументов значения, которые необходимо поместить в создаваемый массив. Доступ к элементу массива можно получить по его *индексному номеру* (или *индексу*). Индексный номер подобен небольшому адресу, по которому можно получить доступ к определенной переменной внутри массива. Так как имена всех переменных в массиве начинаются с имени массива, каждая такая переменная должна иметь собственный уникальный номер. Индексы массивов начинаются с нуля (0). Например, в следующей строке кода в переменную `$my_array` записывается массив, содержащий четыре элемента, пронумерованные 0, 1, 2 и 3:

```
$my_array = array ("кошка", "собака", "лошадь", "золотая рыбка");
```

Переменной `$my_array` присваивается результат выполнения функции `array()`. Если теперь вызвать для данной переменной функцию `is_array()`, то результат будет истинным (функция вернет `true`), указывая на то, что переменная `$my_var` действительно структурирована как массив.

Чтобы получить доступ к значениям только что созданного массива, можно использовать следующий код:


```
$zero_element = $my_array[0];
$one_element = $my_array[1];
$two_element = $my_array[2];
$three_element = $my_array[3];
```

Использование строк в качестве индексов массивов

Новый элемент массива получает следующий номер (начиная с нуля) в последовательности. Однако массивы можно использовать в разных ситуациях, поэтому часто полезно давать элементам имена, а не последовательные числа. Например, в следующем фрагменте кода создается массив, в котором каждый элемент имеет в качестве имени строку, а затем нескольким переменным присваиваются значения именованных строк:

```
$my_named_array = array("dog" => "Пират", "cat" => "Мурзик", "hamster" =>
"Пушок");
$my_dog = $my_named_array["dog"];
$my_cat = $my_named_array["cat"];
$my_hamster = $my_named_array["hamster"];
echo "Мую собаку зовут $my_dog, кота - $my_cat,
а хомячка $my_hamster";
```

Возможность доступа к любому значению по его имени важна, потому что не требуется знать последовательность значений или фактический индексный номер — необходимо знать только имя, заданное элементу. При использовании строк в качестве индексов массивов лучше всего использовать кавычки вокруг имен элементов. Несмотря на то, что опускать кавычки просто и удобно, официальная документация предостерегает от подобной практики, предвосхищая то время, когда кавычки станут обязательными, а их пропуск будет нарушать работу кода.

При необходимости можно использовать индексные номера вместо имен, так как РНР-массивы всегда содержат индексы наряду с любыми присвоенными именами, поэтому следующий код будет работать точно так же, как и в предыдущем примере:

```
$my_named_array = array("dog" => "Пират", "cat" => "Мурзик", "hamster" =>
"Пушок");
$my_dog = $my_named_array[0];
$my_cat = $my_named_array[1];
$my_hamster = $my_named_array[2];
echo "Мую собаку зовут $my_dog, кота - $my_cat,
а хомячка $my_hamster";
```

Инициализация массивов

Инициализировать (т.е. создавать первоначальные значения) массивы можно по-разному. Например, можно использовать функцию `array()` или завершать имя переменной квадратными скобками (`[]`). Написание имени переменной с пустыми квадратными скобками указывает РНР на то, что необходимо создать массив и начать инкрементировать индекс с нуля, если данный элемент является в массиве первым, см. код ниже.

```
$my_array[] = "первый элемент";
```

Если требуется назначить имя новому элементу, то следует вставить это имя в квадратные скобки:

```
$my_array["first"] = "первый элемент";
```

Если после этого снова присвоить элементу `$my_array["first"]` какое-либо значение, то РНР не будет создавать новых элементов, а просто перепишет исходное значение. Однако если снова использовать запись `$my_array[]`, то РНР создаст

в массиве новый элемент и назначит ему индексный номер (следующий номер в последовательности).

Интересно отметить: если программа того требует, то в одном массиве могут содержаться различные значения, каждое из которых может иметь свой тип данных (индексами для всех элементов могут быть исключительно строки или целые числа). Это означает, что массивы можно использовать для хранения данных почти так же, как записи в таблице базы данных.

Работа с массивами

Иногда после создания и инициализации массива (особенно со значениями, взятыми из записи в таблице базы данных) могут возникать некоторые трудности при определении того, какими могут быть эти значения, а, следовательно, в таком случае будет труднее отлаживать код. Эту задачу решает функция `print_r()`, которая позволяет распечатывать все значения элементов массива наряду с их именами и индексами. Продолжая данный пример, создадим простую HTML-страницу, содержащую следующий PHP-код:

```
<?php
$my_named_array = array("dog" => "Пират", "cat" => "Мурзик",
    "hamster" => "Пушок");
print_r($my_named_array);
?>
```

Результат работы этого сценария показан на рис. 2.4.

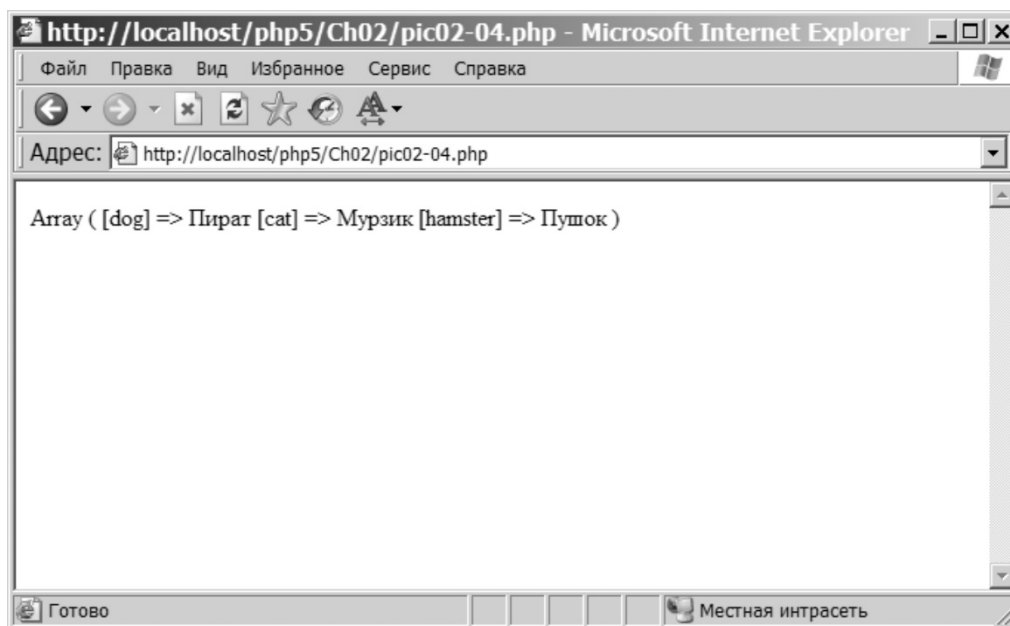


Рис. 2.4.

Использование функции `print_r` весьма полезно, когда требуется просмотреть все содержимое массива. (В главе 4 рассказывается о специальных циклах, которые также значительно облегчают доступ ко всем значениям массива.)

Массивы в PHP, как и во многих других языках программирования, можно смело назвать “рабочими лошадками”; в PHP имеется множество встроенных функций, специально предназначенных для работы с массивами. Далее рассматриваются некоторые из наиболее широко используемых функций. (Более серьезная работа с данными функциями описана в главах 3 и 4.) Многие из функций для работы с массивами аналогичны функциям, которые можно применять к базам данных.

Часто возникают ситуации, когда требуется выяснить, сколько элементов имеется в массиве. Для подсчета количества элементов можно использовать функцию `count()` (при необходимости ее можно также использовать для подсчета элементов в любой переменной):

```
$number_of_elements = count($my_array);
```

Функция `array_count_values()` решает другую задачу — она возвращает (в виде массива) частоту появления значений в массиве, который передан ей в качестве аргумента. Имена элементов в массиве `$returned_array` — значения в массиве `$argument_array`, а значения элементов `$returned_array` — числа, показывающие, сколько раз данные значения встречаются в `$argument_array`. Работу функции проще понять, если рассмотреть следующий код:

```
$argument_array = array("dog", "dog", "cat", "cat", "hamster");
$returned_array = array_count_values($argument_array);
print_r($returned_array);
```

В результате выполнения этого фрагмента выводится следующая информация:

```
Array
(
    [dog] => 2
    [cat] => 2
    [hamster] => 1
)
```

Функция `array_flip()` оказывается полезной, когда требуется поменять местами значения и имена ключей. Например, когда имеется список имен людей в качестве имен элементов, а значением каждого элемента является SSN (Social Security number — номер социального страхования), и необходимо “перевернуть” массив так, чтобы доступ к имени человека можно было получить посредством его SSN-номера (так как SSN должен быть уникален, а имена могут дублироваться). Добиться этого можно с помощью следующего кода:

```
$my_people_array = array("John" => "555-66-7777", "John" => "444-55-3333");
$my_ssn_array = array_flip($my_people_array);
```

Затем массив `$my_ssn_array` можно использовать для поиска людей, даже если имя John повторяется дважды.

Сортировка массивов с помощью функций `sort()` и `asort()`

Часто возникает необходимость отсортировать элементы массива (например, требуется создать список имен в алфавитном порядке). Это можно сделать с помощью функции `sort()`. Для того чтобы отсортировать элементы массива и сохранить при этом связь значений и индексов, используется функция `asort()`. Функция `sort()` сортирует значения и последовательно присваивает им индексы, а функция `asort()` сохраняет связь индексов и значений элементов. Код может выглядеть так:

```
$my_unsorted_array = array("Jim", "Bob", "Mary");
$my_sorted_array = sort($my_unsorted_array);
$my_sorted_array_with_unchanged_indexes = asort($my_unsorted_array);
```

Резюме

В данной главе рассматривались основные этапы написания простой PHP-программы, внедрение PHP-кода в HTML (с использованием корректного расширения имен файлов), запуск программы с Web-сервера, а также принципы написания корректного PHP-кода (использование точки с запятой, разделителей, обратной косой черты и т.д.).

В данной главе описано, как PHP-программы интерпретируются процессором Zend Engine, и как Web-сервер “узнает”, когда отправлять код данному процессору. PHP-код выполняется на сервере, следовательно, он не передается конечному пользователю в формате исходного кода.

Как и во многих языках программирования, в PHP широко применяются переменные, но необходимость объявлять переменные перед использованием отсутствует. И хотя PHP-переменные всегда имеют какой-либо тип данных, явно объявлять этот тип не требуется, PHP автоматически конвертирует типы данных в зависимости от контекста, в котором используются данные.

В главе описывались некоторые встроенные PHP-функции для работы со строками и числами, а также с массивами; на примерах страниц было показано, как код работает на практике. В данной главе рассматривается лишь создание начальных, простейших PHP-программ. Глава 3 проливает свет на то, как сконструированы “настоящие” PHP-программы. Кроме того, в следующей главе подробно рассматривается работа протокола HTTP и HTML-форм.

Упражнения

1. Создайте PHP-программу, которая трансформирует первое предложение во второе, и выводит и результат. Оба предложения представлены ниже:
 - А. Теперь пора всем хорошим людям прийти на помощь стране;
 - Б. Пора теперь стране прийти на помощь всем хорошим людям.
2. Напишите PHP-программу, которая создает два массива чисел и прибавляет значения одного массива к соответствующим (по индексу) значениям другого массива. Массивы должны содержать следующие значения:
 - А. 2, 4, 6, 8, 10;
 - Б. 3, 5, 7, 9, 11.

3

PHP, HTML и состояние сеанса

Теперь, когда читатель знаком с HTTP, простыми PHP-программами, переменными и некоторыми встроенными PHP-функциями и вполне может применить полученные знания на практике, следует изучить ключевое понятие — взаимодействие программы с пользователем. Именно этому моменту разработки Web-приложений целиком посвящена данная глава.

В далеком 1994 году первые Web-страницы состояли из текста, изображений и гиперссылок. Фон Web-страниц был серым; не было таблиц (значительно меньше DHTML и таблиц стилей), которые способствуют структурированию страниц, а интерактивность страниц была минимальной. Введение HTML-дескрипторов для форм и полей форм открыло путь к прямому взаимодействию пользователя и Web-сервера с помощью форм. HTML-формы в настоящее время являются одним из наиболее широко используемых (и наиболее удобных) способов взаимодействия пользователей с Internet-приложениями.

Материал главы поможет читателю изучить HTML-формы и доступные в HTTP-запросах и ответах данные, а также методику применения PHP для получения этих данных и последующего их использования в своих программах.

В этой главе также описана специфика обращения к Web-серверу (с помощью методов GET и POST), формат запросов и ответов, отправляемых между клиентом и сервером (и все полезные данные, которые можно из них извлечь), а также природа запуска приложений через Internet. В главе рассматривается понятие состояния сеанса, недостаток информации о состоянии в HTTP-коммуникациях и несколько методов компенсации этого недостатка. Кроме того, здесь рассматриваются PHP-сеансы.

Основы HTML

Читатели, которые хорошо разбираются в структуре HTML, вполне могут пропустить данный раздел, остальным настоятельно рекомендуется его изучить. Первоначально аббревиатура *PHP* означала PHP Hypertext Preprocessor. Уже один только этот

факт говорит о том, что PHP тесно связан с HTML (Hypertext Markup Language — язык гипертекстовой разметки). Понимание работы HTML, а особенно HTML-дескриптора `<form>` очень важно для квалифицированного PHP-программиста.

HTML был создан Тимом Бернерс-Ли (Tim Berners-Lee) и Робертом Кайлау (Robert Caillau) в 1989 году. HTML представляет собой подмножество стандартного обобщенного языка разметки (Standard Generalized Markup Language — SGML), который в 1986 году был определен Международной организацией по стандартизации как ISO 8879:1986. Язык SGML предназначен для обеспечения общего формата для языков разметки. HTML называется SGML-приложением, поскольку является языком, тогда как XML — просто подмножество SGML-спецификации, используемое для создания собственных языков разметки (более подробно XML описан в главе 8).

Как и большинство SGML-приложений, HTML включает в себя определение типа документа (Document Type Definition — DTD), которое точно определяет синтаксис элементов разметки. В данном разделе приведено несколько примеров HTML DTD.

Информацию о Консорциуме W3C (World Wide Web Consortium) можно найти на сайте www.w3.org. Данная организация поддерживает спецификацию HTML (теперь спецификацию XHTML). Информацию обо всех дескрипторах и атрибутах можно получить на указанном сайте для спецификации HTML 4.01.

HTML является языком разметки, а не языком программирования. Основная задача HTML заключается в отображении данных или содержимого страниц (такого как текст и изображения) наряду с гипертекстовыми ссылками. HTML-теги (“команды” в HTML) помогают Web-дизайнеру упорядочить отображения текста, графики и мультимедиа данных. Единственные дескрипторы, которые предоставляют функции, отчасти сходные с программируемой функциональностью, используются для создания таблиц, ссылок, форм и фреймов.

HTML-код записывается в виде простого текста, при запросе страницы с сервера клиенту возвращается код также в текстовом формате. Ниже приведена простая HTML-страница (без тела):

```
<html>
<head>
<title>Заголовок страницы</title>
</head>
</html>
```

Несмотря на то что в течение многих лет действовало соглашение о том, что HTML-теги следует писать в верхнем регистре (например, <HTML>), HTML-спецификация фактически этого не требует, и писать соответствующие стандарту HTML-теги можно как в верхнем, так и в нижнем регистре или даже смешивать верхний и нижний регистры (например, <hTmL>). Вместе с тем последним стандартом для HTML является XHTML, четко придерживающийся XML-спецификации, поэтому различие между тегами в верхнем и нижнем регистре все-таки есть. В XHTML нижний регистр определен для названий тегов, именно поэтому почти все HTML-теги в данной книге написаны в нижнем регистре. Браузеры не различают регистр написания тегов, однако использование нижнего регистра значительно упростит изменение HTML-кода, когда потребуется обеспечить его соответствие со спецификацией XHTML.

HTML-страница состоит из HTML-тегов, и большинство из них (но не все) имеют начальные (открывающие) и конечные (закрывающие) теги. HTML-теги ограничи-

ваются угловыми скобками (<>). HTML-тег называется именем дескриптора, который он представляет. Например, <html> и </html> представляют собой соответственно открывающий и закрывающий теги дескриптора html. Эти теги обозначают начало и конец всего HTML-документа. Между данными тегами находятся теги для дескрипторов <head> и <title> документа. Теги, которые размещаются внутри других тегов, называются *вложенными*.

Некоторые HTML-дескрипторы имеют только начальный тег, например, дескриптор IMG. Для вставки дескриптора IMG (который вставляет в Web-страницу внешний графический файл) в код Web-страницы требуется записать только тег без конечного тега . Чтобы сообщить браузеру, где расположен внешний файл с изображением, внутри тега необходимо поместить то, что называется *атрибутом*. HTML-атрибуты подобны полям в базе данных или свойствам объекта или переменным в программе. Они имеют имена (например, SRC) и представляют собой контейнеры для значений. Фактически в данном случае необходимо задать атрибуту SRC в теге значение, равное URL-адресу графического файла (например,). Пользовательский браузер, получая HTML-код Web-страницы, считывает этот код, находит URL графического файла, запрашивает данный файл, а затем вставляет его в соответствующую ячейку отображаемой Web-страницы.

HTML DTD

В DTD определяется, какие дескрипторы и атрибуты допускаются в HTML-документе, а также предоставляется дополнительная информация. Хотя HTML-документ состоит из HTML-тегов, в HTML DTD используется специальный формат, определяющий дескрипторы и атрибуты, которые можно использовать. Например, так как HTML DTD определяет дескриптор IMG, этот дескриптор можно использовать в Web-странице.

Однако правильное распознавание и отображение дескрипторов и атрибутов, определенных в HTML DTD, все равно обеспечивает производитель браузера. Отклонения от HTML-спецификации — основная причина, по которой Web-страница может выглядеть (и работать) хорошо в одном браузере и плохо в другом.

Строго говоря, HTML-документы должны начинаться со строки, указывающей используемое DTD-определение, которая содержится в дескрипторе <!DOCTYPE>. Объявление DOCTYPE указывает браузеру правильное DTD, однако браузеры не требуют включения данной строки. Ниже приведено DOCTYPE-объявление, вставленное программой Dreamweaver (популярный инструмент для разработки Web-страниц):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

Дескрипторы form и input

Одним из важнейших HTML-дескрипторов, с которыми приходится работать PHP-программисту, является дескриптор <form>. Рассмотрим, как этот дескриптор определяется в HTML DTD:

```
<!ELEMENT FORM - - (%block;|SCRIPT)+ - (FORM) -- interactive form -->
<!ATTLIST FORM
  %attrs;                                -- %coreattrs, %il8n, %events --
  action      %URI;                      #REQUIRED -- server-side form handler --
  method      (GET|POST)                  GET -- HTTP method used to submit the form --
  enctype     %ContentType;               "application/x-www-form-urlencoded"
  accept      %ContentTypes;              #IMPLIED -- list of MIME types for file upload --
  name        CDATA                      #IMPLIED -- name of form for scripting --
```

```

onsubmit    %Script;          #IMPLIED    -- the form was submitted --
onreset     %Script;          #IMPLIED    -- the form was reset  --
accept-charset %Charsets;    #IMPLIED    -- list of supported charsets --
>

```

DTD-определение для дескриптора `<form>` начинается со строки именования дескриптора, а затем в нем указывается список атрибутов (ATTLIST). Следует отметить атрибут `action`, сообщающий браузеру, куда необходимо отправлять содержимое формы, а также атрибут `method`, который указывает браузеру, как это содержимое отправлять.

Дескриптор `<input>` создает в HTML-форме текстовые поля, переключатели, флажки и т.д. Ниже приведено DTD-определение данного дескриптора.

```

<!ENTITY % InputType
"(TEXT | PASSWORD | CHECKBOX |
  RADIO | SUBMIT | RESET |
  FILE | HIDDEN | IMAGE | BUTTON)"
>
<!-- attribute name required for all but submit and reset -->
<!ELEMENT INPUT - O EMPTY          -- form control -->
<!ATTLIST INPUT
  %attrs;                          -- %coreattrs, %i18n, %events --
  type          %InputType; TEXT   -- what kind of widget is needed --
  name          CDATA              #IMPLIED -- submit as part of form --
  value         CDATA              #IMPLIED -- Specify for radio buttons and checkboxes --
  checked       (checked)         #IMPLIED -- for radio buttons and check boxes --
  disabled      (disabled)        #IMPLIED -- unavailable in this context --
  readonly      (readonly)        #IMPLIED -- for text and passwd --
  size          CDATA              #IMPLIED -- specific to each type of field --
  maxlength     NUMBER             #IMPLIED -- max chars for text fields --
  src           %URI;              #IMPLIED -- for fields with images --
  alt           CDATA              #IMPLIED -- short description --
  usemap        %URI;              #IMPLIED -- use client-side image map --
  ismap         (ismap)            #IMPLIED -- use server-side image map --
  tabindex      NUMBER             #IMPLIED -- position in tabbing order --
  accesskey     %Character;        #IMPLIED -- accessibility key character --
  onfocus      %Script;           #IMPLIED -- the element got the focus --
  onblur        %Script;           #IMPLIED -- the element lost the focus --
  onselect      %Script;           #IMPLIED -- some text was selected --
  onchange      %Script;           #IMPLIED -- the element value was changed --
  accept        %ContentTypes;    #IMPLIED -- list of MIME types for file upload --
>

```

Атрибут `type` определяет тип элемента управления, который появляется на экране в браузере (`text` — текстовое поле, `radio` — переключатель и т.д.).

Для того чтобы создать Web-страницу с формой, можно написать следующий HTML-код в виде простого текста и загрузить его на Web-сервер (или даже открыть непосредственно в браузере). В результате этого должна отобразиться хорошо отформатированная Web-страница.

```

<html>
<head>
<title>
</title>
</head>
<body bgcolor="white">
<form method="post" action="http://www.example.com">
Имя пользователя:<input type="text" name="username"><br>
Пароль:<input type="password" name="password"><br>
<input type="submit" value="Login">
</form>
</body>
</html>

```


Указанный код создает простую форму с двумя полями (имя пользователя и пароль) и кнопкой отправки данных. При нажатии этой кнопки содержимое формы отправляется серверу `www.example.com`.

Каким образом можно заставить HTML-формы и PHP работать сообща для создания динамически генерируемых Web-страниц (а не просто копировать страницы в браузер с Web-сервера)? Прежде всего, с помощью HTML-тегов в виде простого текста необходимо создать Web-страницу и включить в нее HTML-форму. Затем в данную страницу следует ввести PHP-код и убедиться, что он соответствующим образом ограничен разделителями `<?php` и `?>`.

Когда браузер запрашивает Web-страницу, любой PHP-код в ней преобразовывается или обрабатывается PHP-интерпретатором, прежде чем результаты будут возвращены пользователю, а результаты PHP-обработки подставляются в те места Web-страницы, где был расположен исходный PHP-код. Можно также написать код, который будет обрабатываться, только если на форме нажата кнопка отправки данных (как написать такой код, показано в нескольких последующих разделах). Этот код может даже использовать в своей работе некоторые из переданных посредством формы данных.

Окончательный результат PHP-обработки является HTML-кодом, но поскольку PHP-интерпретатор перед отправкой окончательной версии Web-страницы имеет возможность выполнить какую-либо обработку, некоторое содержимое страницы (любые фрагменты, сгенерированные PHP-кодом) может отличаться при каждом новом запросе этой страницы. А если обработка выполняется в ответ на передачу данных формы, то появляется возможность создавать множество различных интерактивных функций.

Доступ к PHP- и HTTP-данным

Итак, вы уже знаете, как работает протокол HTTP и как происходит взаимодействие клиентов и серверов, поэтому понимание процесса взаимодействия между пользовательским браузером (клиентским приложением) и Web-сервером (серверным приложением) во время работы PHP-программ в сети не должно вызвать у вас трудностей. В данной главе рассматриваются HTML-формы и то, как они способствуют взаимодействию пользователя с Web-сервером (и, следовательно, с PHP-программами). Эти темы заслуживают пристального внимания, поскольку HTTP, HTML и PHP тесно связаны друг с другом. Понимание происходящего между браузером и сервером очень важно для PHP-программиста, так как внутри запросов и ответов, передаваемых от клиента к серверу и обратно, находятся полезные данные, которые можно использовать.

Каждый раз, когда пользователь нажимает на ссылку или кнопку отправки в форме, браузер отправляет Web-серверу множество информации о самом себе и операционной системе на компьютере пользователя. Аналогично во время ответа Web-сервер отправляет множество данных о самом себе. В PHP имеются возможности получения данных, которые передаются браузером, а также способы открытия данных о PHP-инсталляции на сервере. Например, когда пользователь направляется на какой-либо Web-сайт на основе PHP и регистрируется на нем, вероятнее всего, на сервере заполняется предопределенная переменная `$_POST` (в нее, в частности, попадает имя пользователя и пароль), а также предопределенная переменная `$_SERVER`, в которой содержится информация о текущей серверной среде. Обе эти переменные доступны PHP-приложению, работающему на сайте. Эти аспекты PHP более подробно обсуждаются в нескольких последующих разделах. Количество доступных данных действительно удивительно (особенно учитывая то, что большинство начинающих разработчиков даже не подозревают о них). Далее описано, как получить доступ к этим данным.

Предопределенные переменные

PHP автоматически создает довольно много переменных (которые называются *предопределенными переменными*), доступных в любой точке программы. Эти переменные являются массивами, и доступ к ним может осуществляться по имени, как и к любым другим переменным. По умолчанию PHP сконфигурирован так, чтобы не передавать непосредственно эти переменные в сценарии (в файле `php.ini register_globals=Off`). Чтобы получить доступ к хранящимся в них данным, необходимо использовать их полные имена. Например, если в форме имеется текстовое поле с именем `username` и это поле заполнено и форма отправлена (в данном случае предполагается, что метод отправки `POST`), то получить доступ к данным поля можно с помощью следующего кода:

```
$my_new_username = $_POST[username];
```

Предопределенные переменные также называются *суперглобальными*, поскольку они доступны независимо от области действия. Предопределенные переменные хранят большинство информации, содержащейся в HTTP-запросах и ответах, включая серверные переменные, строки запроса, переменные форм и т.д. Предопределенные переменные, так же как и обычные переменные, можно использовать в любых целях, но некоторые из них могут отсутствовать в какой-либо конкретной инсталляции PHP, так как Web-серверы отличаются данными, передаваемыми посредством HTTP. В следующем разделе показано, как получить данные, хранящиеся в предопределенных переменных.

В дополнение к предопределенным переменным, для получения базовой информации об инсталляции PHP и операционной системе можно использовать встроенную PHP-функцию `phpinfo()`. Данная функция позволяет не только протестировать инсталляцию и работу PHP (см. главу 1), но также дает возможность выяснить многие подробности о том, как установлен PHP на сервере. Например, можно выяснить версию PHP, операционную систему, используемую на данном сервере, и другие параметры.

Простейший способ определить, какую информацию предоставляет функция `phpinfo()`, заключается в запуске файла `test01.php`, созданного в главе 1. Однако в данном случае это делается не только для того, чтобы проверить работоспособность инсталляции PHP (как это было в главе 1), но и чтобы получить возможность изучить различные данные, предоставляемые функцией `phpinfo()`. В ходе выполнения функция создает хорошо отформатированную и детально проработанную страницу (включая все необходимые HTML-теги), на которой представлена информация о версии PHP, операционной системе, версии Zend Engine, установках в `php.ini`, дополнительных модулях и предопределенных переменных. Фрагмент страницы, создаваемой функцией `phpinfo()`, показан на рис. 3.1.

Переменные в HTTP-запросах и ответах

Между клиентом и Web-сервером передается очень много данных. Например, вместе с каждым запросом от клиента к серверу передается IP-адрес (для того чтобы сервер знал, куда отправлять ответ). Кроме IP-адреса также передаются подробные сведения о версии браузера, инициирующего данный запрос, cookie-файлы, данные форм, версия Web-сервера и другая информация. Эти данные содержатся в предопределенных переменных, структурированных в виде ассоциативных массивов, поэтому доступ к ним можно осуществлять по имени, как и в случае любых других массивов. Содержимое каждой из этих переменных и их назначение представлено после следующего раздела “Практика”.

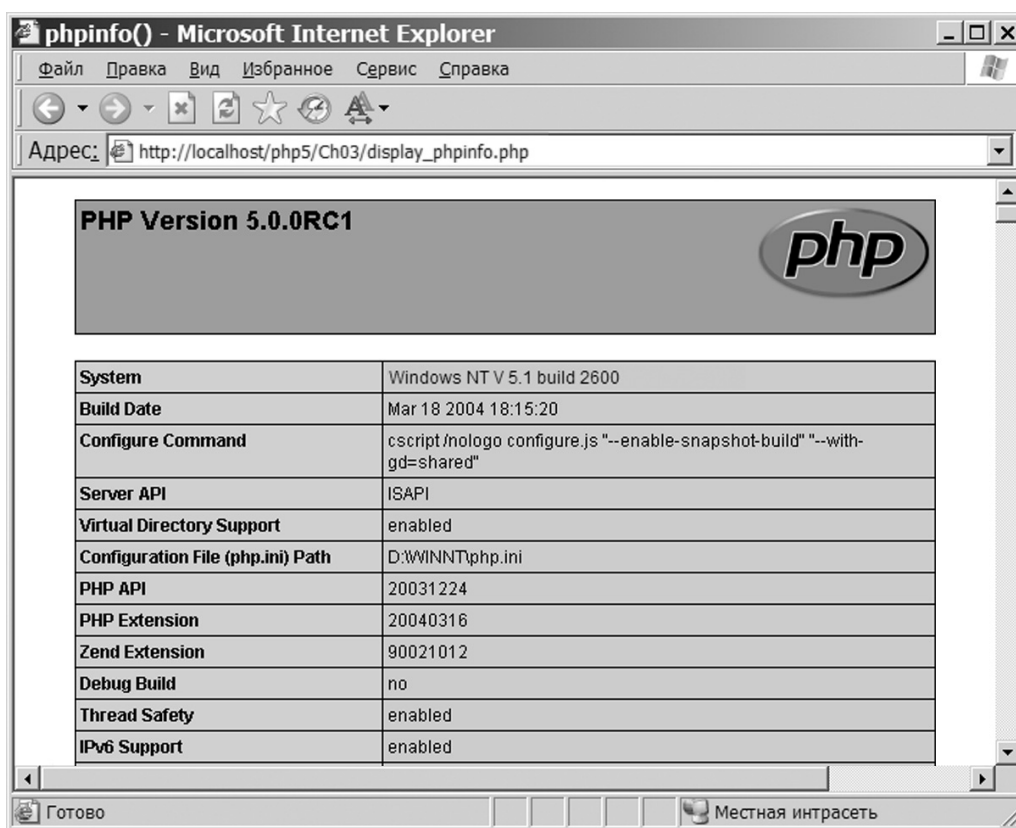


Рис. 3.1.

В конфигурационном файле PHP (php.ini) имеется директива `register_globals`. По умолчанию она имеет значение `off` (начиная с PHP 4.2) и ограничивает доступ к некоторым предопределенным переменным. С практической точки зрения это означает, что для доступа к данным предопределенных переменных необходимо использовать полное имя соответствующего массива (например, так: `$_SERVER['DOCUMENT_ROOT']`). Существует функция (`import_request_variables()`), которая импортирует переменные GET, POST и Cookie в глобальную область видимости, поэтому к ним можно получить доступ непосредственно по имени, однако рекомендуется все-таки использовать полное имя массива, поэтому в данной книге используется именно такой синтаксис.

Практика Отображение массива \$GLOBALS

Следующий код распечатывает содержимое предопределенной переменной `$GLOBALS`. Данный код можно вставить в отформатированную HTML-страницу.

```
<?php
echo "<pre>";
print_r($GLOBALS);
echo "</pre>";
?>
```

Если вызвать данный код в браузере (назовем этот файл `displaying_predefined_vars.php`), то можно посмотреть содержимое массива `$GLOBALS`. Может так случиться, что все элементы массива в данный момент окажутся пустыми. В конце этого раздела помещен рисунок, показывающий, как выглядит содержимое предопределенных переменных.

Файл следует разместить в соответствующем каталоге Web-сервера. С этого момента предполагается, что читатель создал все необходимые ему и доступные для Web-сервера каталоги и по мере необходимости помещает в них создаваемые файлы.

Как это работает

Функция `print_r()` распечатывает информацию о переменных в удобочитаемом для пользователя формате. Особенно полезна данная функция для вывода содержимого массивов, так как она распечатывает по порядку ключи и значения. Рекомендуется использовать ее с HTML-тегами `<pre>`, так чтобы данные располагались на странице не в одну строку, а по одному элементу в строке.

Суперглобальные массивы

В приведенной ниже таблице описаны предопределенные массивы. Они называются суперглобальными, потому что к ним можно получить доступ из любой точки PHP-программы, не используя ключевое слово `global` и независимо от области видимости.

Массив	Описание
<code>\$GLOBALS</code>	Содержит ссылку на каждую переменную, имеющую глобальную область видимости в PHP-программе. Многие переменные в <code>\$GLOBALS</code> также являются другими суперглобальными массивами
<code>\$_SERVER</code>	В данном массиве содержатся все данные, отправляемые сервером в HTTP-ответе, например, имя выполняемого в текущий момент сценария, имя сервера, версия HTTP, удаленный IP-адрес и т.д. Несмотря на то что большинство Web-серверов создают одинаковые серверные переменные, делают это не все серверы, и не все серверные переменные обязательно содержат данные
<code>\$_GET</code>	Содержит все переменные строки запроса, которые были прикреплены к URL или были созданы в результате использования метода <code>GET</code>
<code>\$_POST</code>	Содержит все переданные переменные формы и их данные. Массивы <code>\$_POST</code> или <code>\$_REQUEST</code> широко используются в большинстве PHP-программ. Например, чтобы получить имя или пароль (или любую другую пользовательскую информацию), переданные как часть HTML-формы, применяются PHP-переменные из массива <code>\$_REQUEST</code>
<code>\$_COOKIE</code>	Содержит все cookie-файлы, отправленные браузером серверу. Они преобразуются в переменные, которые можно считывать из данного массива. Записывать cookie-файлы в пользовательский браузер можно с помощью функции <code>setcookie()</code> . Cookie-файлы обеспечивают средство идентификации пользователя между запросами страниц (или между посещениями сайта, в зависимости от срока действия cookie-файла) и часто используются автоматически при обработке сеансов
<code>\$_FILES</code>	Содержит любые объекты, загруженные на сервер с использованием метода <code>POST</code> . Данный массив отличается от массива <code>\$_POST</code> тем, что он специально предназначен для хранения загружаемых объектов (например, графических файлов), а не содержимого полей переданной HTML-формы

Окончание таблицы

Массив	Описание
<code>\$_ENV</code>	Содержит данные о среде, в которой работает Web-сервер и PHP, например, имя компьютера, версию операционной системы и т.д.
<code>\$_REQUEST</code>	Содержит все данные массивов <code>\$_GET</code> , <code>\$_POST</code> и <code>\$COOKIE</code>
<code>\$_SESSION</code>	Содержит все переменные, которые зарегистрированы в текущий момент времени как переменные сеанса. Так как имеется возможность программируемого управления переменными, зарегистрированными в сеансе, содержимое этого массива в каждый момент времени зависит от того, используются ли сеансы, а также от действий, выполняемых программой

Следующий код генерирует страницу, демонстрирующую содержимое описанных выше суперглобальных переменных. При внимательном изучении кода становится очевидным, что в нем много повторяющихся фрагментов; это связано с тем, что отображаются все переменные (наряду с множеством HTML-кода для форматирования). Прежде чем вызывать данный код в браузере, его следует сохранить в файле с именем `displaying_predefined_vars.php`.

```
<html>
<head>
<title>Предопределенные переменные</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body bgcolor="#FFFFFF">
<table width="100%" border="1">
  <tr><td colspan="2"><font face="Arial, Helvetica, sans-serif" size=
"-1"><b>Отображение предопределенных переменных</b></font></td> </tr>
  <tr><td width="40%" valign="top"><font face="Arial, Helvetica, sans-serif"
size="-1">Глобальные - $GLOBALS</font></td>
    <td width="60%"><font face="Arial, Helvetica, sans-serif" size="-2">
<?php
echo "<pre>";
print_r($GLOBALS);
echo "</pre>";
?>
</font></td>
  </tr>
  <tr><td width="40%" valign="top"><font face="Arial, Helvetica,
sans-serif" size="-1">Серверные - $_SERVER</font></td>
    <td width="60%"><font face="Arial, Helvetica, sans-serif" size="-1">
<?php
echo "<pre>";
print_r($_SERVER);
echo "</pre>";
?>
</font></td>
  </tr>
  <tr><td width="40%" valign="top"><font face="Arial, Helvetica,
sans-serif" size="-1">Массив $_GET </font></td>
    <td width="60%"><font face="Arial, Helvetica, sans-serif" size="-1">
<?php
echo "<pre>";
print_r($_GET);
echo "</pre>";
?>
</font></td>
  </tr>
</table>
```

```
<tr><td width="40%" valign="top"><font face="Arial, Helvetica,
    sans-serif" size="-1">Массив $_POST </font></td>
    <td width="60%"><font face="Arial, Helvetica, sans-serif" size="-1">
<?php
echo "<pre>";
print_r($_POST);
echo "</pre>";
?>
</font></td>
</tr>
<tr><td width="40%" valign="top"><font face="Arial, Helvetica,
    sans-serif" size="-1">Cookie - $_COOKIE</font></td>
    <td width="60%"><font face="Arial, Helvetica, sans-serif" size="-1">
<?php
echo "<pre>";
print_r($_COOKIE);
echo "</pre>";
?>
</font></td>
</tr>
<tr><td width="40%" valign="top"><font face="Arial, Helvetica,
    sans-serif" size="-1">Файлы - $_FILES</font></td>
    <td width="60%"><font face="Arial, Helvetica, sans-serif" size="-1">
<?php
echo "<pre>";
print_r($_FILES);
echo "</pre>";
?>
</font></td>
</tr>
<tr><td width="40%" valign="top"><font face="Arial, Helvetica,
    sans-serif" size="-1">Среда - $_ENV</font></td>
    <td width="60%"><font face="Arial, Helvetica, sans-serif" size="-1">
<?php
echo "<pre>";
print_r($_ENV);
echo "</pre>";
?>
</font></td>
</tr>
<tr><td width="40%" valign="top"><font face="Arial, Helvetica,
    sans-serif" size="-1">Запрос - $_REQUEST</font></td>
    <td width="60%"><font face="Arial, Helvetica, sans-serif" size="-1">
<?php
echo "<pre>";
print_r($_REQUEST);
echo "</pre>";
?>
</font></td>
</tr>
<tr><td width="40%" valign="top"><font face="Arial, Helvetica,
    sans-serif" size="-1">Сессия - $_SESSION</font></td>
    <td width="60%"><font face="Arial, Helvetica, sans-serif" size="-1">
<?php
echo "<pre>";
print_r($_SESSION);
echo "</pre>";
?>
</font></td>
</tr>
</table>
</body>
</html>
```

Результат работы данной страницы показан на рис. 3.2.

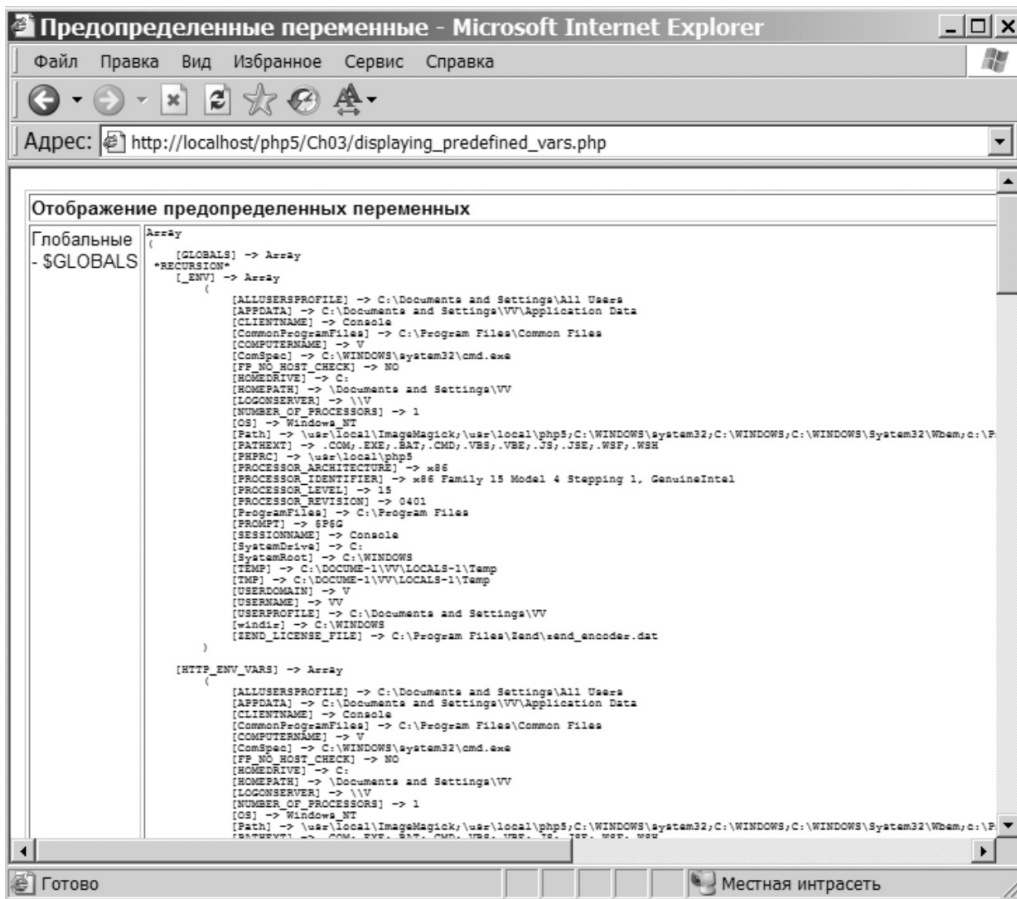


Рис. 3.2.

Гиперссылки

Читатели, которые занимались Web-дизайном или программированием, несомненно, знакомы со структурой ссылок в HTML. В технических терминах в HTML-ссылке используется указательный элемент (имеющий начальный и конечный теги, соответственно, `<a>` и ``) и один из главных атрибутов `href`, значением которого является URL-адрес. Текст или изображение, помещенное между начальным и конечным тегами `<a>`, образует ссылку, при нажатии на которую в браузере отображается страница, имеющая URL, равный значению атрибута `href`, см. пример ниже.

`Нажмите здесь`

Нажатие на ссылки является одной из форм взаимодействия пользователя с Web-приложением и позволяет обмениваться данными с сервером. Однако это взаимодей-

ствие весьма ограничено, поскольку все, что пользователь может сделать — инициировать запрос на страницу, которая уже подготовлена разработчиком Web-сайта. Например, при нажатии на ссылку “О нас” единственная причина, по которой отображается необходимая страница, заключается в том, что разработчик сайта уже жестко вписал код ссылки с URL данной страницы.

Строки запросов

Сравнительно гибкой является *строка запроса* (*query string*), которая присоединяется в конец URL ссылки. Строка запроса может состоять из множества *пар имя/значение*. Количество таких пар определяется разработчиком. Серьезное отличие строки запроса от обычной ссылки заключается в том, что PHP предоставляет способ, позволяющий пользователю задавать значения, которые могут быть вставлены в строку запроса. Предположим, например, что пользователь ввел значение John в качестве имени, тогда PHP-программа может использовать это значение посредством переменной, например, `$first_name`.

Термин “пара имя/значение” является распространенным способом обозначения любого именованного контейнера для значения. Парами имя/значение могут быть поля в таблице базы данных, переменные, HTML- и XML-атрибуты и т.д.

Используя следующий код, можно написать PHP-программу, которая генерирует строку запроса, присоединяемую к URL (предполагается, что переменные `$first_name` и `$last_name` уже заданы):

```
<a href="http://www.myplace.com?first_name=<?php echo $first_name; ?>">Нажмите  
здесь</a>
```

Данный код генерирует следующий вывод:

```
<a href="http://www.myplace.com?first_name=John">Нажмите здесь</a>
```

Строка запроса состоит из знака вопроса, за которым следует имя первой пары имя/значение, знак равенства и значение первой пары имя/значение. Если установлены обе переменные (`$first_name` и `$last_name`), то в строку запроса можно включить обе пары имя/значение, соединив их амперсандом (&), например:

```
<a href="http://www.myplace.com?first_name=<?php echo $first_name; ?>  
&last_name=<?php echo $last_name; ?>">Нажмите здесь</a>
```

Такой код будет генерировать следующий результат:

```
<a href="http://www.myplace.com?first_name=John&last_name=Wayne">Нажмите здесь</a>
```

Строки запроса весьма полезны (особенно для сохранения значений между запросами страницы; эта тема обсуждается в настоящей главе позднее), но, очевидно, они все-таки тоже довольно ограничены. Чтобы поместить в строку запроса предоставленное пользователем значение, необходимо иметь это значение. Значение можно было бы получать из базы данных. Однако если это значение перед использованием необходимо получить от пользователя, то следует предоставить пользователю какой-нибудь простой метод ввода. И здесь на помощь приходит очень мощный метод взаимодействия между пользователем и приложением — HTML-форма.

HTML-формы (или Web-формы)

Как уже отмечалось, существует два основных метода взаимодействия пользователя с Internet-приложением в браузере: нажатие на ссылку или отправка формы. Нажатие на ссылку приводит к тому, что Web-серверу отправляется запрос на определенную страницу, а если разработчик приложения включил в строку запроса какие-либо данные, то эти данные также будут отправлены. Однако HTML-форма предоставляет более широкие возможности отправлять данные Web-серверу. Хотя некоторые из полей формы могут быть предустановленными (например, данные в строке запроса), другие поля (например, поля типа `text` и `textarea`) дают возможность контролировать передаваемые данные. Таким образом, HTML-формы широко используются для создания интерактивных возможностей, которые подобны интерактивности настольных приложений.

Дескрипторы HTML-форм

В HTML-спецификации описаны HTML-дескрипторы форм, включая дескриптор `<form>` (начинающий и завершающий форму), `<input>` (создающий несколько типов полей, в частности, текстовые поля, переключатели, кнопки и др.), `<textarea>` (создающий текстовое поле с возможностью ввода нескольких строк), а также дескриптор `<select>` (который создает выпадающие и статические списки значений).

У каждого из этих дескрипторов имеются атрибуты. Например, в дескриптор `<form>` включается атрибут `method`. Значение данного атрибута может устанавливаться в начальном теге `<form>` и может быть словом `post` (например, `method="POST"`); в таком случае при отправке формы будет использоваться HTTP-метод `POST`.

Несмотря на то что HTML-дескрипторы часто называют тегами, термин “тег” обозначает только начальный или конечный HTML-разделитель. Например, `<p>` — начальный тег HTML-дескриптора `P` (создание абзаца), а `</p>` — его конечный тег. Вместе теги `<p>` и `</p>` образуют дескриптор `P`. Текст между ними (текст, который отображается в браузере как абзац) называется содержимым дескриптора.

При условии правильного написания дескрипторов HTML-форм их комбинация создает весьма мощное средство взаимодействия пользователя с Web-приложением. В последующих разделах подробно обсуждаются дескрипторы полей формы и их атрибуты.

Дескриптор `<form>`

Как работает HTML-форма? Пользователь заполняет различные текстовые поля и, когда все данные внесены, нажимает на кнопку **Отправить** (`Submit`); затем введенная им информация упаковывается одним из двух способов и отправляется Web-серверу. После этого Web-сервер может извлечь информацию и передать ее PHP-машине. PHP обрабатывает информацию и возвращает ее браузеру как часть HTTP-ответа. Чтобы создать форму, единственное, что нужно сделать — написать HTML-страницу с открывающим и закрывающим тегами `<form>`. Любые элементы управления, которые размещаются между тегами `<form>`, автоматически становятся частью формы, отправляемой Web-серверу. Например:

```
<form name="myform" action="myform_processor.php" method="POST">
<input type="text" name="first_name">
<input type="submit" name="button" value="Отправить ответ">
</form>
```

В данном случае форма получает имя `myform`. Когда пользователь нажимает кнопку **Отправить ответ**, запрос (методом POST) и все введенные пользователем данные отправляются для обработки PHP-файлу `myform_processor.php`. Естественно, данный запрос можно отправить любой странице или файлу; его даже можно отправить текущей странице, но при этом необходимо помнить, что запрашиваемая страница должна иметь код для приема отправленных формой данных и их обработки.

Первый дескриптор `<input>` имеет атрибут `type`, определяющий тип отображаемого поля формы, в данном случае это текстовое поле. Хотя в приведенном примере имя формы (`myform`) не имеет особого значения, имя первого дескриптора `<input>` крайне важно, так как PHP автоматически создает переменную `$first_name` и присваивает ей значение, равное тексту, введенному пользователем в данное поле. Эта переменная будет частью массива `$_GET` или `$_POST` в зависимости от установленного для формы метода отправки.

Второй дескриптор `<input>` имеет тип `submit`, поэтому он отображает кнопку отправки данных. Значением атрибута `value` является строка **Отправить ответ**, поэтому на кнопке появляется надпись “Отправить ответ”. В данном случае ни имя, ни значение не являются особенно важными (поскольку рассматривается только принцип PHP-обработки), однако следует учитывать, что как только форма передается серверу, имя и значение становятся доступными для обработки в PHP-программе.

В тот момент, когда пользователь заполняет форму и отправляет ее, имена и значения полей формы становятся переменными, доступными для принимающей PHP-программы (она указывается в атрибуте `action` дескриптора `<form>`). Таким образом, весь диапазон взаимодействия пользователя с HTML-формами становится очевидным.

Обратите внимание на несколько важных моментов, касающихся HTML-форм.

- ❑ На одной Web-странице может быть несколько форм, каждая из которых (в том числе и ее поля) может отправляться независимо от других.
- ❑ Обычно имена форм не важны, если не используются JavaScript-сценарии, определяющие, какую форму обрабатывать или отправлять.
- ❑ Значения, передаваемые в форме, всегда являются строками; как только они попадают в PHP-программу, их можно преобразовать в любой тип данных.
- ❑ Не все имена полей и значения становятся доступными в PHP-программе при отправке формы. Например, передается значение только выбранного переключателя (в группе переключателей).

Атрибуты дескриптора `<form>`

Дескриптор `<form>` имеет множество атрибутов, однако особое внимание следует уделить только двум из них: `action` и `method`. Другие атрибуты, такие как `id`, `class`, `dir`, `lang`, `language`, `name`, `style` и `title`, универсальны для большинства или всех HTML-тегов и не требуют дальнейших пояснений. Более сложные атрибуты `accept-charset` и `enctype`, которые определяют набор символов и MIME-тип данных формы, выходят за рамки главы (атрибут `enctype` более подробно рассматривается в главе 7). Атрибут `target` работает аналогично одноименному атрибуту дескриптора `<a>`: он позволяет указывать фрейм или окно, в котором следует отображать Web-страницу, отосланную в ответ на отправку данной формы.

Атрибут action

Атрибут `action` указывает серверу, на какую страницу необходимо перейти, как только пользователь нажмет кнопку отправки данных в форме. Не имеет значения, является ли эта страница HTML- или PHP-страницей, или в ней используется какая-либо серверная технология, главное, чтобы такая страница существовала на Web-сервере. Чтобы задать ссылку на какую-либо страницу, атрибут `action` используется так:

```
<form action="myprogram.php">
...
</form>
```

Когда в качестве значения атрибута `action` указана PHP-страница, информация, введенная в данную форму, фактически отправляется PHP-машине для обработки, таким образом, PHP-приложение получает возможность работать с введенными пользователем данными. Атрибут `action` только указывает серверу, какую страницу необходимо выдать следующей. Если сохранить предыдущую страницу как `myprogram.html` вместо `myprogram.php` и если PHP не сконфигурирован для обработки `.html`-файлов, форма не будет отправлена PHP-машине и в браузере вообще ничего не отобразится. Работа PHP-машины при получении формы будет показана позднее.

Атрибут method

Атрибут `method` управляет способом отправки информации серверу. Как уже отмечалось, существует два метода отправки `GET` и `POST` (их имена нечувствительны к регистру символов, но соглашение требует писать их прописными буквами). Метод `GET` используется по умолчанию. Значение атрибута `method` устанавливается с помощью следующего кода:

```
<form action="myprogram.php" method="GET">
```

или

```
<form action="myprogram.php" method="POST">
```

На самом деле данному атрибуту можно присвоить несколько значений: `HEAD`, `PUT`, `LINK`, `UNLINK`, `OPTIONS`, `DELETE`, `TRACE` и `CONNECT`. Однако они используются нечасто, необходимость в них возникает крайне редко, поэтому в данной книге они не обсуждаются. Вместо этого следует подробнее рассмотреть методы `GET` и `POST`.

Значение GET

Значение `GET` атрибута `method` заставляет браузер присоединять введенные пользователем в форму значения к URL. Как и в случае строки запроса, присоединенной к URL в ссылке, при отправке формы браузер добавляет в конце URL знак вопроса, чтобы обозначить окончание URL и начало информации формы. В таком случае информация, введенная в форму, передается как пары имя/значение. Если значением атрибута `method` дескриптора `<form>` является `GET`, то браузер автоматически присоединяет данную информацию к URL при отправке запроса Web-серверу.

Так же, как и в строке запроса в ссылке, можно добавить к URL множество пар имя/значение, отделяя каждую пару амперсандом (`&`). С двумя парами имя/значение конец URL после отправки формы может выглядеть следующим образом:

```
http://www.nonexistentserver.com/test.php?furryanimal=cat&spikyanimal=porcupine
```

Чтобы обеспечить соответствие с XHTML-спецификацией, можно заменить амперсанд последовательностью символов `&`, которая является соответствующим XML-элементом для амперсанда. В таком случае URL и строка запроса будут выглядеть так:

```
http://www.nonexistentserver.com/test.php?furryanimal=cat&amp;spikyanimal=porcupine
```

Как уже отмечалось, пары имя/значение очень похожи на переменные. Фактически, как только они передаются Web-серверу для обработки, PHP делает доступными эти пары в виде переменных. Поэтому если форма отправлена Web-серверу и в ответ он выдал другую страницу, то пары имя/значение будут доступны в PHP-сценарии как переменные (как часть массива `$_GET`).

Иногда возникает необходимость передавать в строке запроса значения, содержащие пробелы. Предположим, что существует форма, в которой имеется дескриптор `<textarea>`. Если пользователь ввел в это поле строку “I would like to see a dynamic menu in operation”, то в строке запроса необходимо представить несколько пробелов. В таком случае пробелы будут заменены оператором сложения:

```
http://localhost/beginning_php5/ch03/form.php?TextArea= I+would+like+
to+see+a+dynamic+menu+in+operation
```

Однако что произойдет, если понадобится вставить знак “плюс” в текстовое поле (`<textarea>`)? Как он будет представлен в строке запроса? Данный символ или оператор необходимо заменить кодом, который соответствует определенному символу. Такая замена называется URL-кодированием.

URL-кодирование

Многие символы невозможно использовать ни в URL, ни, следовательно, в строке запроса, поэтому URL с такими символами необходимо кодировать.

Процесс кодирования не требует каких-либо дополнительных действий от пользователя или разработчика. Web-браузер принимает опасный символ, будь то скобка или знак “плюс”, и заменяет его кодовым значением (при отправке серверу). И наоборот, браузер принимает кодированный URL и заменяет в нем значения соответствующими символами (во время отображения страницы на экране). URL-кодированные значения всегда одинаковы (например, пробел всегда представляется значением `%20`). В приведенной ниже таблице перечислены наиболее распространенные символы и их кодовые значения.

<i>Символ</i>	<i>URL-кодированное значение</i>
Знак табуляции	<code>%09</code>
Пробел	<code>%20</code>
!	<code>%21</code>
“	<code>%22</code>
#	<code>%23</code>
%	<code>%25</code>
&	<code>%26</code>
(<code>%28</code>
)	<code>%29</code>
+	<code>%2B</code>
,	<code>%2C</code>

Окончание таблицы

Символ	URL-кодированное значение
.	%2E
/	%2F
:	%3A
;	%3B
<	%3C
>	%3E
=	%3D
?	%3F
@	%40
\	%5C

Некоторые из этих символов должны быть закодированы, иначе в строке запроса они примут другое значение (как было показано выше, оператор сложения используется для обозначения пробела в строке запроса, а знак вопроса означает начало строки запроса).

Предыдущая строка запроса с URL-кодированным значением для пробела вместо знака “плюс” выглядела бы следующим образом:

```
http://localhost/beginning_php5/ch03/form.php?TextArea=I%20would%20like%20to%20see%20a%20dynamic%20menu%20in%20operation
```

Пример значения GET в действии будет показан позднее, а пока следует рассмотреть значение POST.

Значение POST

Одним из недостатков передачи данных в строках запроса является их общедоступная природа такой передачи. Если появление передаваемой информации в URL нежелательно, то вместо метода GET следует использовать метод POST. Данный метод работает почти идентично методу GET; различие заключается в том, что информация формы передается не в URL, а в теле HTTP-запроса. Это означает, что информация невидима, поскольку она не присоединяется к URL. Метод POST также позволяет передавать большее количество данных. Существует физический предел количества данных, которые можно переслать как часть URL.

Какой метод использовать: GET или POST?

Мнения по этому поводу расходятся. Одни разработчики утверждают, что метод GET почти никогда не следует использовать, поскольку он небезопасен и имеет ограничения по размеру передаваемых данных. Другие придерживаются той точки зрения, что метод GET можно использовать для получения информации, а POST следует применять всякий раз, когда требуется модифицировать данные на Web-сервере. Однако строгих и четких правил не существует, а эти мнения представляют собой только основные принципы.

Одним из недостатков метода POST является то, что страницы, загруженные с его помощью, невозможно записать в закладки браузера, тогда как страницы, загруженные с помощью метода GET, непосредственно в URL содержат всю необходимую для воспроизведения запроса информацию. Во многих случаях можно создать закладку на результат обработки какой-либо формы (например, результаты поиска в Alta Vista),

использующей метод GET. Именно поэтому большинство поисковых машин использует метод GET. Другой недостаток POST заключается в том, что данный метод сам по себе небезопасен — несмотря на то, что информация, помещенная в тело HTTP-запроса, невидима для рядовых пользователей, она не шифруется, и хакеры легко могут получить к ней доступ. Для того чтобы гарантировать безопасность информации, необходимо использовать защищенное соединение с безопасным сервером.

Решение о том, какой метод использовать в форме, зависит от задач, решаемых с помощью данной формы. Используя GET, следует помнить о недостатках этого метода и о его открытой природе. В то же время страницы, работающие с помощью метода POST, не могут быть проиндексированы поисковыми машинами, а, кроме того, невидимая передача данных вовсе не означает, что данный метод более безопасен.

PHP и поля (элементы управления) HTML-форм

Изучив процесс работы HTML-форм, теперь более подробно рассмотрим самые распространенные элементы управления, которые можно использовать для сбора пользовательской информации, а затем и способы использования PHP для получения этих данных. Термины *поле формы* (*form field*), *элемент управления* (*control*) и *элемент формы* (*form element*) являются взаимозаменяемыми; все они означают одно и то же.

Все последующие примеры данного раздела требуют использования двух Web-страниц. Первая страница получает переданную пользователем информацию, а вторая отправляет эту информацию от Web-сервера и PHP-машины обратно браузеру. Следует отметить, что объединить форму и ее ответ в одном файле вполне возможно, но это приведет к созданию громоздкого кода и усложнению программ. Фактически PHP-программы часто содержат множество файлов, составляющих приложение.

Первая страница вообще не должна содержать какого-либо PHP-кода. На практике во многих сайтах Web-страницы, содержащие формы, написаны исключительно на HTML и имеют расширение .htm или .html. Именно так реализованы все последующие примеры (хотя по мере увеличения сложности приложений разработчики, как правило, отклоняются от этого формата). Очевидно, что нет необходимости отправлять какую-либо информацию PHP-машине, так как если файл не содержит PHP-кода, это только увеличит издержки (продолжительность времени, необходимого на обработку и генерацию Web-страницы, возвращаемой браузеру, увеличится).

Итак, рассмотрим наиболее распространенные элементы управления HTML-форм.

Текстовые поля (текстовые окна)

Текстовые поля (text fields), или текстовые окна (text boxes), являются одними из наиболее известных элементов управления, которые встречаются практически в любой форме. Они создаются с помощью дескриптора `<input>`, в котором атрибуту `type` присваивается значение `text`.

```
<input type="Text" name="TextBox1">
```

Преимущество текстовых полей заключается в том, что они могут принимать от пользователя целые текстовые предложения. Это позволяет использовать их, например, для ответов на вопросы, допускающие многочисленные и непредсказуемые возможные ответы.

Следующий пример представляет собой код Web-страницы, которая принимает имя любимого автора и возвращает его на следующей странице.

Практика Использование текстового поля

1. Откройте текстовый редактор и введите следующий HTML-код:

```
<html>
<head><title></title></head>
<body>
<form method="GET" action="text.php">
Кто Ваш любимый автор?
<input name="Author" type="text">
<br>
<br>
<input type="submit" value="Отправить">
</form>
</body>
</html>
```

2. Сохраните данный файл под именем text.html.
3. Создайте новый файл и введите следующий код:

```
<html>
<head><title></title></head>
<body>
Ваш любимый автор:
<?php
echo $_GET['Author'];
?>
</body>
</html>
```

4. Сохраните этот файл как text.php.
5. Откройте text.html в браузере и введите в текстовое поле имя (рис. 3.3).

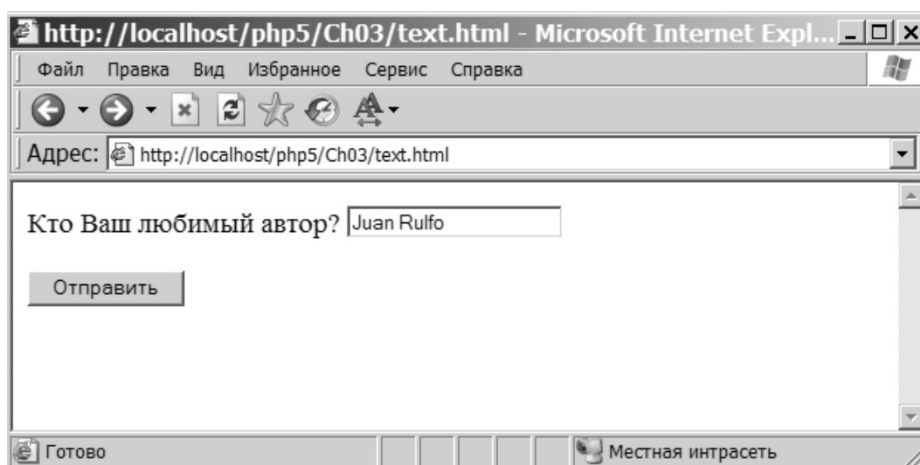


Рис. 3.3.

6. Нажмите кнопку Отправить и в браузере отобразится введенное имя (рис. 3.4).



Рис. 3.4.

Как это работает

Прежде всего следует обратить внимание на URL вверху экрана на рис. 3.4. К концу URL страницы `text.php` была добавлена строка запроса. Строка запроса была добавлена Web-браузером согласно соответствующей инструкции в файле `text.html`:

```
<html>
<head><title></title></head>
<body>
<form method="GET" action="text.php">
Кто Ваш любимый автор?
...
```

Присвоение атрибуту `method` значения `GET` вынуждает браузер отправлять информацию формы в виде строки запроса, а не скрывать ее в теле HTTP-запроса. Рассмотрим строку запроса вверху страницы на рис. 3.4:

```
?Author=Juan+Rulfo
```

Вы уже знаете, что строки запроса состояются из пар имя/значение. В данном случае значением является имя автора `Juan Rulfo`. Имя данной пары (`Author`) попадает в строку запроса согласно следующей выделенной строке в коде `text.html`:

```
Кто Ваш любимый автор?
<input name="Author" type="text">
<br>
```

Атрибут `name` тега `<input>` устанавливает имя для данного текстового поля — `Author`. Значение добавляется, когда пользователь вводит в текстовое поле имя автора.

Вторая программа, `text.php`, фактически состоит из одной строки PHP-кода:

```
Ваш любимый автор:
<?php echo $_GET['Author'];
?>
```

Строка PHP-кода отображает содержимое переменной `$_GET["Author"]`. Данная переменная физически не создается в коде. Она автоматически создается как часть массива `$_GET`. В HTML-файле было создано текстовое поле с именем `Author`. Когда

форма передается Web-серверу и PHP-машине, PHP-машина создает массив `$_GET` с элементом `Author`. Если бы текстовое окно было названо `Name`, то переменная имела бы имя `$_GET['Name']`. Это все, что выполняет данная программа.

Почему этот пример может не работать

Если строка запроса передается, но имя автора не возвращается (см. рис. 3.5) или возвращается с предупреждением, то, скорее всего, в PHP-коде не соблюдается регистр символов в имени переменной.

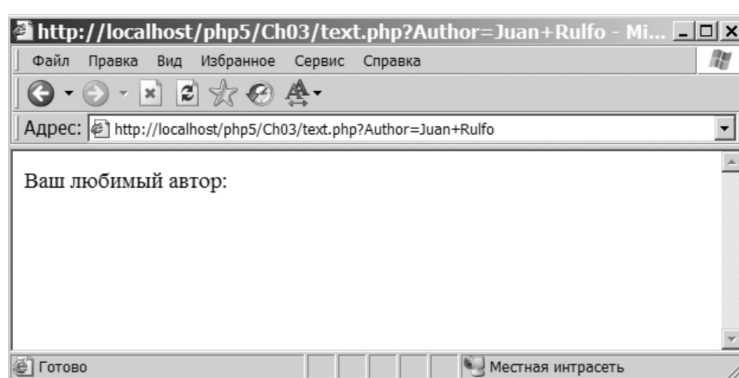


Рис. 3.5.

Предположим, что в файле `text.html` имеется следующая строка (`author` в нижнем регистре):

```
<input name="author" type="text">
```

тогда как в файле `text.php` используется переменная `$_GET['Author']` (первая буква прописная):

```
echo $_GET['Author'];
```

Программа будет работать неправильно, так как имена переменных в PHP чувствительны к регистру символов. Хотя HTML нечувствителен к регистру, PHP принимает имя переменной таким, каким оно задано для текстового поля в HTML-коде, поэтому создается PHP-переменная `$_GET['author']`, а не `$_GET['Author']`. Необходимо убедиться, что имя текстового поля в HTML и имя, используемое в PHP-сценарии, абсолютно идентичны.

Текстовая область

Для того чтобы создать текстовое поле, позволяющее вводить несколько строк, следует использовать другой HTML-дескриптор: `<textarea>`. Для изменения размера поля, количества строк и столбцов, а также других свойств используются атрибуты этого дескриптора. Например, код

```
<textarea name="WebSites" rows="30" cols="50">
```

создает текстовую область (text area) с именем `WebSites` размером 30 строк на 50 столбцов. Текстовые области предназначены для того, чтобы принимать от пользователя целые предложения. Их преимущество заключается в том, что можно задавать раз-

мер, а кроме того, они позволяют принимать несколько строк текста. Дескриптор `<textarea>` требует использования закрывающего тега, а между тегами можно поместить текст, который будет использоваться по умолчанию.

Рассмотрим использование текстовой области на примере.

Практика Использование текстовой области

1. Запустите редактор Web-страниц и введите следующий код:

```
<html>
<head><title></title></head>
<body>
<form method="POST" action="textarea.php">
  Перечислите Ваши любимые Web-сайты
  <textarea name="WebSites" cols="50" rows="5">
    http://
    http://
    http://
    http://
  </textarea>
  <br>
  <br>
  <br>
  <input type="submit" value="Отправить">
</form>
</body>
</html>
```

2. Сохраните данный файл как `textarea.html` и закройте его.
3. Создайте новый файл и введите следующий код:

```
<html>
<head><title></title></head>
<body>
  Ваши любимые Web-сайты:
  <?php
    echo $_POST['WebSites'];
  ?>
</body>
</html>
```

4. Сохраните этот файл под именем `textarea.php`.
5. Откройте файл `textarea.html` в браузере (рис. 3.6) и введите URL-адреса нескольких Web-сайтов.
6. По окончании ввода (вводить все четыре сайта необязательно) нажмите кнопку **Отправить**. На рис. 3.7 приведен примерный результат работы программы.

Как это работает

Вывод выглядит не настолько аккуратным и компактным, как в предыдущем примере. Однако это не должно отвлекать внимание от одного важного момента, а именно — URL:

`http://localhost/beginning_php5/ch03/textarea.php`

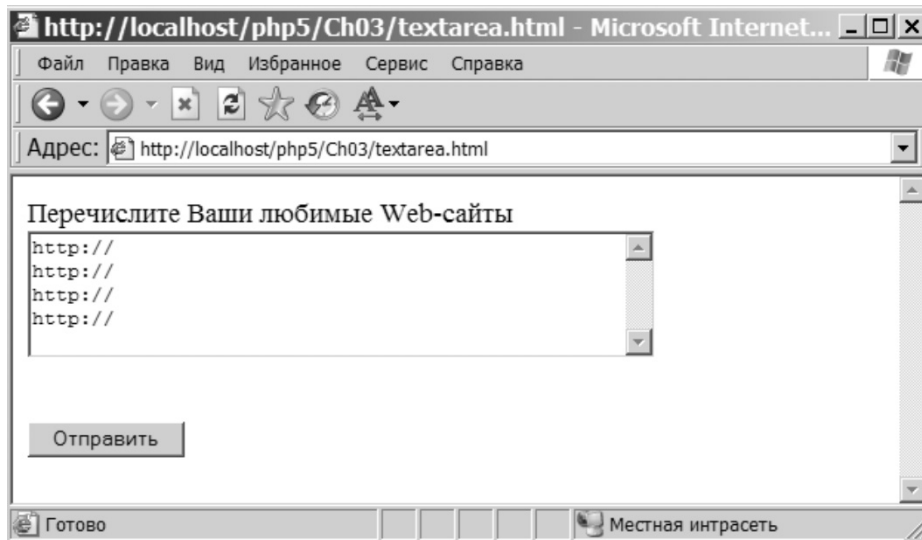


Рис. 3.6.

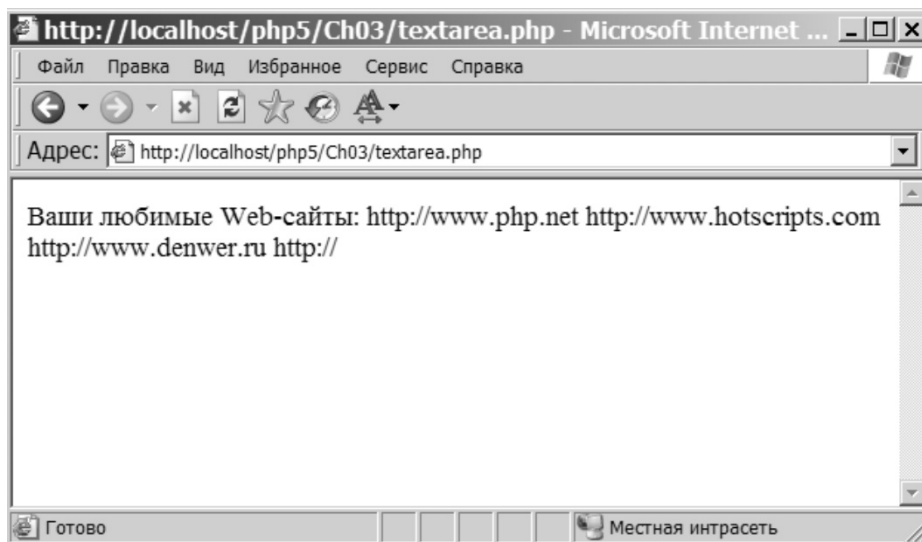


Рис. 3.7.

В данном случае строка запроса не присоединяется, потому что в первой программе задан метод POST.

```
<html>
<head><title></title></head>
<body>
<form method="POST" action="textarea.php">
Перечислите Ваши любимые Web-сайты
...
```

Это изменение необходимо для того, чтобы скрыть информацию формы. Однако данная страница имеет еще один интересный момент — дескриптор `<textarea>`:

```
<textarea name="WebSites" cols="50" rows="5">
http://
http://
http://
http://
</textarea>
```

В данном случае создается текстовая область на пять строк по 50 символов в каждой. В отличие от обычного HTML текст внутри дескриптора `<textarea>` не требует явного указания разрывов строк (тега `
`); достаточно начать вводить текст с новой строки и тогда он будет отображаться на новой строке. Этому элементу управления присваивается имя `WebSites`. Затем во второй программе (`textarea.php`) PHP-переменная получает имя `$WebSites` (регистр символов в обоих случаях должен быть одинаковым):

```
...Ваши любимые Web-сайты:
<?php
echo $_POST['WebSites'];
?>
...
```

Все содержимое дескриптора `<textarea>` отображается на экране, но символы возврата каретки, разделяющие адреса сайтов в HTML-странице, автоматически удаляются браузером наряду со всеми лишними пробелами, чтобы текст помещался в окно.

Флажки

Флажок (check box) — еще один элемент управления, который, как и текстовое поле, создается в HTML с помощью дескриптора `<input>`. Флажок представляет собой небольшой квадрат с “галочкой” или без нее, в зависимости от выбора пользователя, и не требует от пользователя ввода каких-либо данных кроме установки или снятия “галочки”. Поэтому этот элемент управления содержит данные, которые весьма отличаются от данных текстового поля. В HTML процесс создания флажка аналогичен созданию текстового поля; единственным отличием является значение атрибута `type`:

```
<input name="Choice" type="checkbox">
```

Флажки применимы в ситуациях, когда пользователю необходимо ответить на вопрос, требующий строгого однозначного ответа да/нет. Флажки также имеют атрибут `checked`, который не принимает никакого значения. Если этот атрибут используется, то данный флажок будет отмечен по умолчанию:

```
<input name="Choice" type="checkbox" checked>
```

Кроме того, данный элемент управления имеет атрибут `value`; его значение по умолчанию — `on`.

Преимущества флажков по сравнению с другими типами дескриптора `input` будут более очевидными, если использовать флажки в практическом примере. В следующем упражнении используется только один флажок, а его значение возвращается на экран.

Практика Использование флажков

1. Введите следующий код в редакторе Web-страниц:

```
<html>
<head><title></title></head>
```

```
<body>
<form method="POST" action="checkbox.php">
Вы когда-нибудь ели телячий рубец?
<input name="Choice" type="checkbox">
<br>
<br>
<input type="submit" value="Отправить">
</form>
</body>
</html>
```

2. Сохраните данный файл как `checkbox.html` и закройте его.
3. Создайте новый файл и введите в него следующий код:

```
<html>
<head><title></title></head>
<body>
<?php
echo $_POST['Choice'];
?>
</body>
</html>
```

4. Сохраните файл с именем `checkbox.php` и закройте его.
5. Откройте страницу `checkbox.html` в браузере и поставьте флажок, как показано на рис. 3.8.

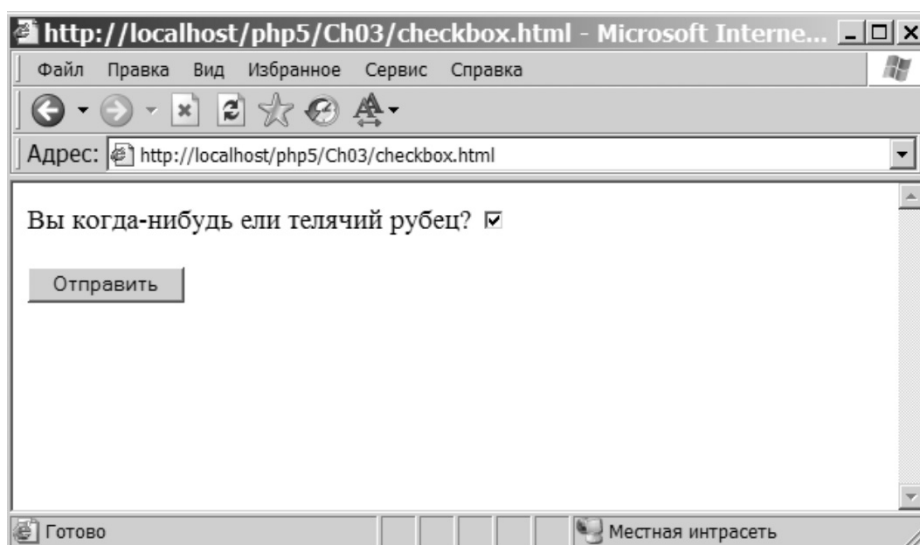


Рис. 3.8.

6. Нажмите кнопку **Отправить**. На странице будет выведено значение флажка `on` (рис. 3.9).
7. Нажмите кнопку **Назад** браузера, снимите флажок и снова нажмите кнопку **Отправить**. Так как флажок в данном случае не был отмечен, в окне браузера отображается пустая страница.

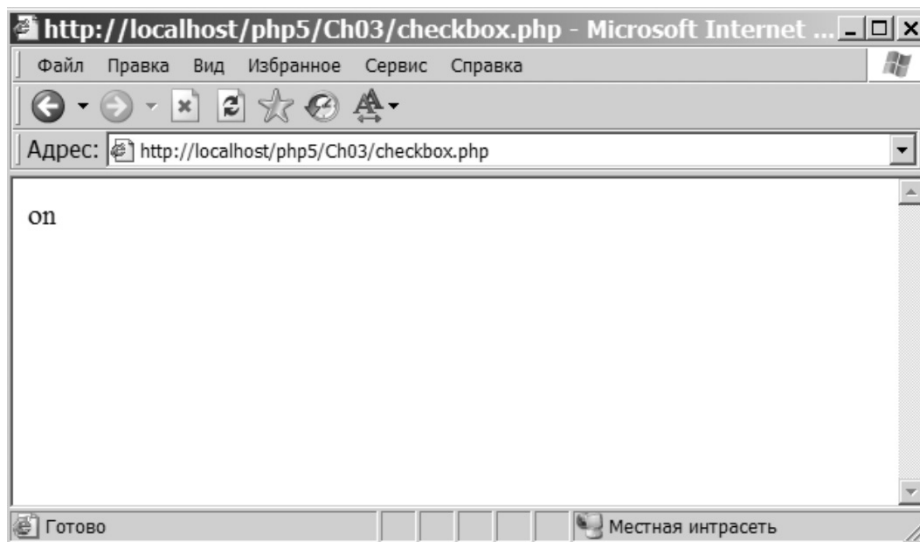


Рис. 3.9.

Как это работает

В данном случае в конце URL строка запроса отсутствует, так как для передачи Web-серверу данных формы используется метод POST. Метод передачи определяется в первом из двух созданных файлов, `checkbox.html`:

```
<html>
<head><title></title></head>
<body>
<form method="POST" action="checkbox.php">
```

Флажок создается с помощью дескриптора `<input>`:

```
Вы когда-нибудь ели телячий рубец?
<input name="Choice" type="checkbox">
```

В сценарии `checkbox.php` вызывается PHP-переменная, которая имеет в точности то же имя, что и элемент управления в файле `checkbox.html`:

```
<?php
echo $_POST['Choice'];
?>
```

Единственное отличие заключается в том, что переменная теперь создается и получает значение автоматически. Если флажок был установлен пользователем, то значением данной переменной будет "on". В противном случае переменная вообще не содержит данных.

Использование нескольких флажков

Что произойдет, если использовать на форме несколько флажков? Читатели, которые знакомы с работой переключателей (radio buttons), знают, что можно выбрать только один переключатель в группе. Флажки работают иначе и их преимущество заключается в том, что каждый из них является отдельным элементом — можно выбрать один или несколько флажков или вообще не выбрать ни одного.

Модифицируем предыдущий пример, добавив в него несколько флажков.

Практика Использование нескольких флажков

1. Откройте редактор Web-страниц и сохраните следующий код как `checkboxes.html`:

```
<html>
<head><title></title></head>
<body>
<form method="POST" action="checkboxes.php">
Вы когда-нибудь ели телячий рубец?
<input name="Choice1" type="checkbox" value="телячий рубец">
<br>
Вы когда-нибудь ели улиток?
<input name="Choice2" type="checkbox" value="улиток">
<br>
Вы когда-нибудь ели саранчу?
<input name="Choice3" type="checkbox" value="саранчу">
<br>
<br>
<input type="submit" value="Отправить">
</form>
</body>
</html>
```

2. Закройте данный файл и создайте новый файл. Введите в файл следующий код:

```
<html>
<head><title></title></head>
<body>
<?php
echo "$_POST[Choice1]<br>";
echo "$_POST[Choice2]<br>";
echo "$_POST[Choice3]<br>";
?>
</body>
</html>
```

3. Сохраните данный файл как `checkboxes.php`.
4. В браузере откройте страницу `checkboxes.html` (рис. 3.10).

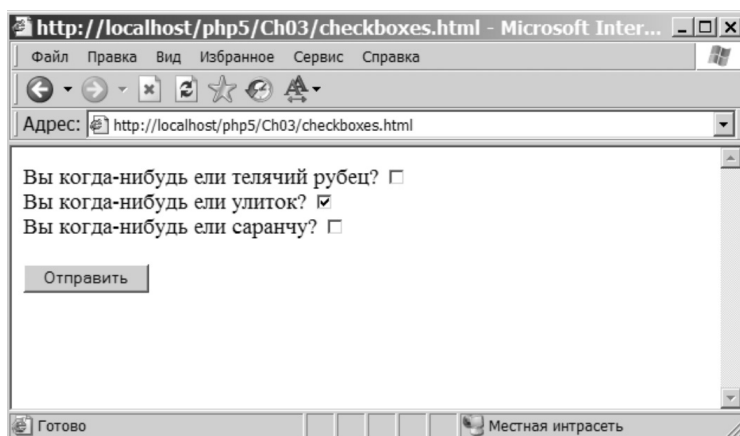


Рис. 3.10.

5. Установите один или несколько флажков и нажмите кнопку Отправить. Программа должна отобразить пользовательский выбор (рис. 3.11).

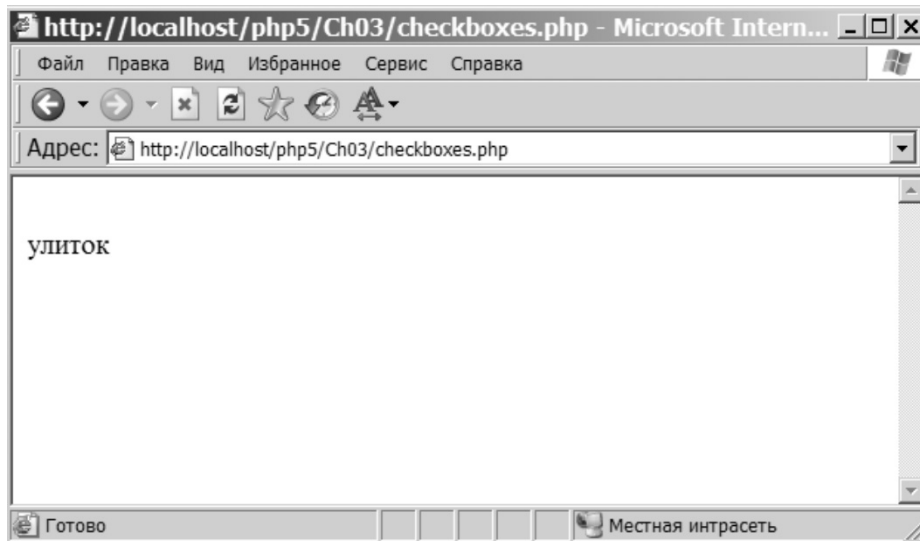


Рис. 3.11.

Как это работает

Атрибут `value` для каждого флажка устанавливается в первой программе:

```
Вы когда-нибудь ели телячий рубец?
☐
<br>
Вы когда-нибудь ели улиток?
☐
<br>
Вы когда-нибудь ели саранчу?
☐
```

В результате этого для каждого отмеченного флажка устанавливается значение. Например, если выбран флажок `Choice1`, то его значением будет `телячий рубец` (а не стандартное `on`), и это значение передается в переменную `$Choice1` сценария `checkboxes.php`. Если пользователь не устанавливает этот флажок, то в соответствующую РНР-переменную данные не передаются. Сценарий `checkboxes.php` отображает значения трех переменных, установленных независимо на странице `checkboxes.html`:

```
echo "$_POST[Choice1]<br>";
echo "$_POST[Choice2]<br>";
echo "$_POST[Choice3]<br>";
```

Каждому флажку назначено уникальное имя и присвоено значение, независимое от других флажков. Существует возможность задать всем трем элементам управления одно и то же имя, но это может привести к неожиданным результатам. Например, изменим следующий код в файле `checkboxes.html`:

```
Вы когда-нибудь ели телячий рубец?
```



```
<input name="Choice" type="checkbox" value="телячий рубец">
```

```
<br>
Вы когда-нибудь ели улиток?
```

```
<input name="Choice" type="checkbox" value="улиток">
```

```
<br>
Вы когда-нибудь ели саранчу?
```

```
<input name="Choice" type="checkbox" value="саранчу">
```

Теперь, если запустить данную программу снова и выбрать несколько вариантов, то результат будет совсем не таким, как ожидалось. Программа выдаст только один ответ, значение последнего флажка в списке. PHP перезаписывает значение переменной каждый раз, когда встречается имя этой переменной. Поэтому переменной присваивается значение последнего флажка в списке. Однако к имени каждого элемента управления в HTML можно добавить пару квадратных скобок:

```
Вы когда-нибудь ели телячий рубец?
<input name="Choice[]" type="checkbox" value="телячий рубец">
<br>
Вы когда-нибудь ели улиток?
<input name="Choice[]" type="checkbox" value="улиток">
<br>
Вы когда-нибудь ели саранчу?
<input name="Choice[]" type="checkbox" value="саранчу">
```

Это приводит к созданию массива переменных внутри массива \$_POST. Для того чтобы различать переменные, PHP добавляет к их именам номер, действующий как уникальный идентификатор. В конце имени первой версии такой переменной содержится ноль в квадратных скобках ([0]), в конце имени второй версии — единица в квадратных скобках ([1]), а в конце имени третьей версии — двойка ([2]).

Чтобы заставить PHP отображать содержимое данных переменных, необходимо обращаться к переменным, явно указывая их полные имена, например, \$_POST[Choice][0]. Переменная \$_POST[Choice][0] имеет значение телячий рубец, если был установлен одноименный флажок. В переменной \$_POST[Choice][1] содержится значение улиток, если был выбран флажок улиток, и т.д. для всех остальных элементов управления, использующих одно и то же имя. Создать массив из HTML-формы, не используя квадратные скобки, невозможно, хотя, как было сказано в главе 2, в PHP-программе можно создавать массивы любого размера.

Переключатели

Переключатели (radio buttons) — “эгоистичные” родственники флажков. Переключатели используются, когда имеется набор возможных ответов или вариантов, но выбрать можно только один из них. Переключатели также создаются с помощью дескриптора <input>, а атрибут type имеет значение radio.

```
<input name="Question1" type="radio" value="Порту">
```

Переключатели, как и флажки, имеют в HTML-коде атрибут checked, который не принимает никакого значения. Если в коде элемента управления используется этот атрибут, то на Web-странице переключатель будет выбран по умолчанию:

```
<input name="Question1" type="radio" checked>
```

Если значение атрибута `value` не задано, то по умолчанию оно устанавливается равным `"on"`.

В отличие от флажков, чтобы связать набор переключателей друг с другом, необходимо задать всем переключателям группы одинаковое имя. Например:

```
<input name="Question1" type="radio" value="Порту">
<input name="Question1" type="radio" value="Лиссабон">
<input name="Question1" type="radio" value="Мадрид">
```

Данный метод сообщает Web-серверу, что все эти переключатели соединены. Если задать каждому переключателю уникальное имя, то пользователь сможет выбирать каждый вариант независимо, как в случае флажков.

Практика Использование переключателей

1. Откройте редактор Web-страниц и введите следующий код:

```
<html>
<head><title></title></head>
<body>
<form method="GET" action="radio.php">
  Выберите название столицы Португалии
  <br>
  <br>
  <input name="Question1" type="radio" value="Порту">
  Порту
  <br>
  <input name="Question1" type="radio" value="Лиссабон">
  Лиссабон
  <br>
  <input name="Question1" type="radio" value="Мадрид">
  Мадрид
  <br>
  <br>
  <input type="submit" value="Отправить">
</form>
</body>
</html>
```

2. Сохраните данный файл как `radio.html` и закройте его.
3. Создайте новый файл и введите в него следующий код:

```
<html>
<head><title></title></head>
<body>
<?php
echo "Вы выбрали ответ: $_GET[Question1]";
?>
</body>
</html>
```

4. Сохраните файл как `radio.php`.
5. Откройте в браузере страницу `radio.html` и выберите ответ (рис. 3.12).
6. Нажмите кнопку **Отправить** и посмотрите результат своего выбора (рис. 3.13).

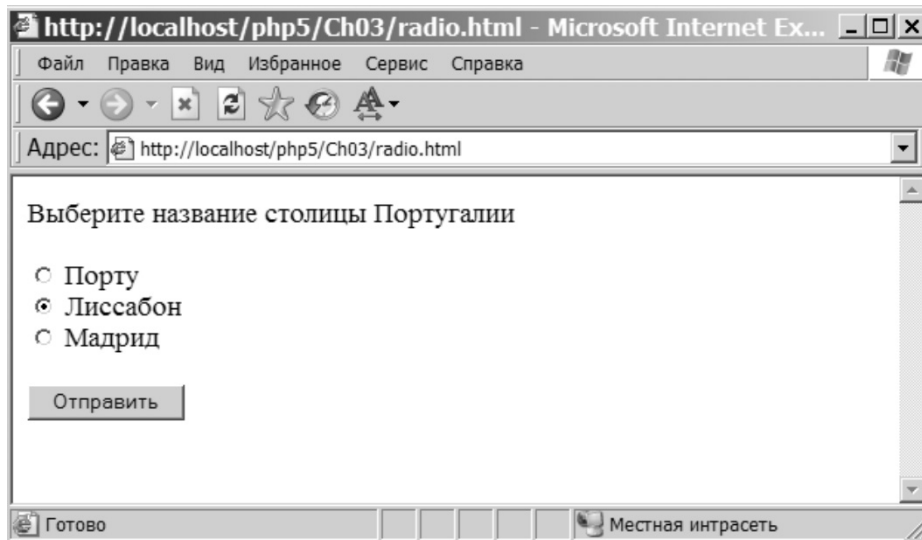


Рис. 3.12.

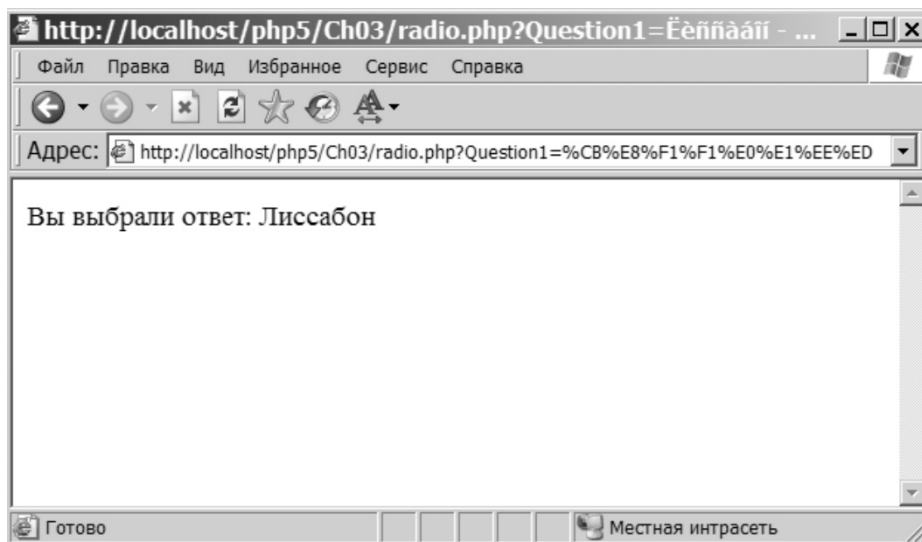


Рис. 3.13.

Как это работает

Для транспортировки данных выбран метод GET, поэтому снова видна строка запроса. Страница с перечнем вопросов — фактически единственное место, где такой подход может оказаться полезным. Страница подтверждает выбранный ответ, хотя, вероятно, было бы полезнее, если бы ответ не отображался в теле страницы. Рассмотрим работу программ. Код первого файла, `radio.html`, создает три переключателя

теля. Все они имеют одинаковое имя, Question1, и уникальные значения, отражающие различные ответы:

```
<input name="Question1" type="radio" value="Порту">
Порту
<br>
<input name="Question1" type="radio" value="Лиссабон">
Лиссабон
<br>
<input name="Question1" type="radio" value="Мадрид">
Мадрид
```

Затем в сценарии radio.php необходимо только отобразить содержимое одной переменной, так как может быть только один правильный ответ:

```
<?php
echo "Вы выбрали ответ: $_GET[Question1]";
?>
```

Списки

Списки (list boxes) или выпадающие списки (drop-down list boxes) представляют собой элементы управления, которые обычно отображают несколько объектов в списке. Иногда они имеют стрелку, которая позволяет пользователю перемещаться вниз к дополнительным объектам. В HTML они работают несколько иначе, чем предыдущие элементы управления, так как они создаются с помощью двух дескрипторов: `<select>` и `<option>`. По сути, они обеспечивают ту же функциональность, что и переключатели, при условии, что обычно можно выбрать только один объект из предопределенного списка вариантов.

Дескриптор `<select>`, создающий список, охватывает несколько дескрипторов `<option>`, каждый из которых содержит текст, соответствующий объекту в списке:

```
<select name="Price">
  <option>До $5000</option>
  <option>$5000-$10000</option>
  <option>$10,000-$25,000</option>
  <option>Свыше $25,000</option>
</select>
```

Однако возможны ситуации, когда необходимо выбрать несколько объектов. Чтобы предоставить пользователю такую возможность, следует добавить в дескриптор `<select>` атрибут `multiple`. Оба варианта рассматриваются в следующем примере, где программа получает от пользователя информацию о желаемой цене автомобиля и желаемом объеме двигателя. Первый вопрос допускает только один ответ, а второй допускает выбор нескольких объектов, которые можно выбрать, удерживая нажатой клавишу `<Shift>`.

Практика Использование списков

1. Откройте HTML-редактор и введите следующий код:

```
<html>
<head><title></title></head>
<body>
  <form method="POST" action="listbox.php">
    Выберите желаемую цену автомобиля
    <br>
    <br>
    <select name="Price">
```

```

        <option>До $5000</option>
        <option>$5000-$10000</option>
        <option>$10,000-$25,000</option>
        <option>Свыше $25,000</option>
    </select>
    <br>
    <br>
    Выберите желаемый объем двигателя?
    <br>
    <br>
    <select name="EngineSize[]" multiple>
        <option>1.0L</option>
        <option>1.4L</option>
        <option>1.6L</option>
        <option>2.0L</option>
    </select>
    <br>
    <br>
    <input type="submit" value="Отправить">
</form>
</body>
</html>

```

2. Сохраните данный файл как `listbox.html` и закройте его.
3. Создайте новый файл и введите в него следующий код:

```

<html>
<head><title></title></head>
<body>
<?php
echo "Диапазон цен: $_POST[Price]";
$Choice0 = $_POST['EngineSize'][0];
$Choice1 = $_POST['EngineSize'][1];
$Choice2 = $_POST['EngineSize'][2];
$Choice3 = $_POST['EngineSize'][3];
echo "<br>Объем двигателя: $Choice0";
echo "$Choice1";
echo "$Choice2";
echo "$Choice3";
?>
</body>
</html>

```

4. Сохраните данный файл как `listbox.php`.
5. Откройте в браузере страницу `listbox.html` и выберите один вариант из верхнего списка и один или несколько вариантов из нижнего (рис. 3.14).
6. Нажмите кнопку **Отправить**. Пример результата показан на рис. 3.15.

Как это работает

Код страницы `listbox.html` создает список с четырьмя пунктами, допускающий множественный выбор. Атрибуту `name` дескриптора `<select>` присваивается значение `Price`:

```

<select name="Price">
    <option>До $5000</option>
    <option>$5000-$10000</option>
    <option>$10,000-$25,000</option>
    <option>Свыше $25,000</option>
</select>

```

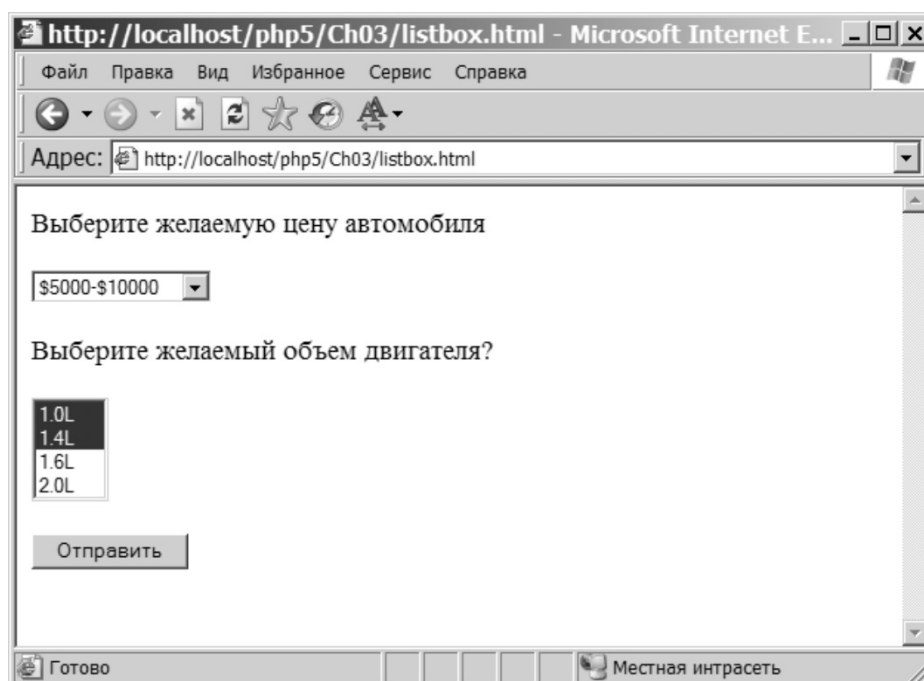


Рис. 3.14.

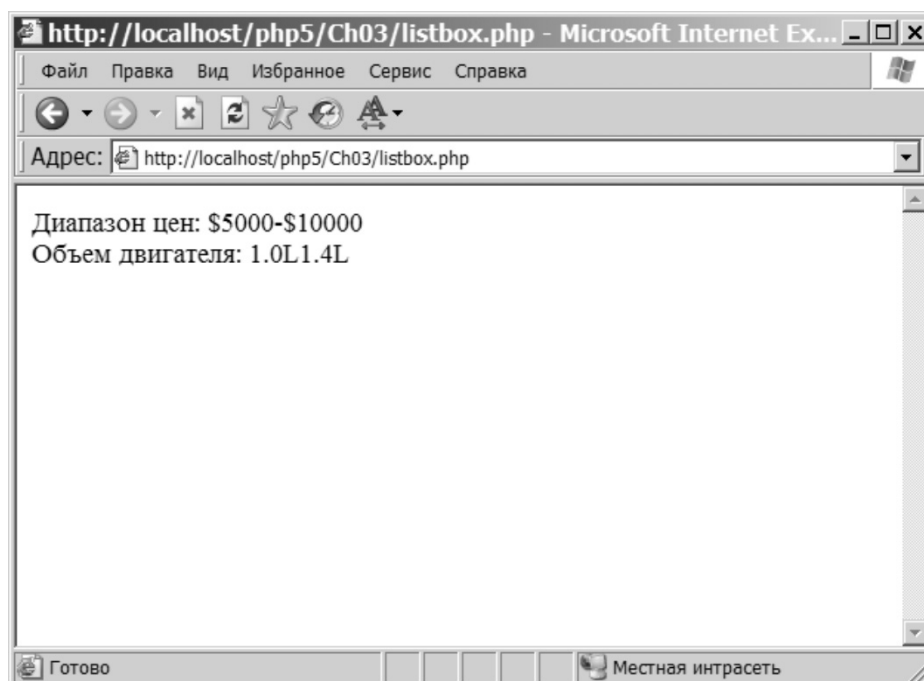


Рис. 3.15.

В сценарии `listbox.php` атрибут `name` связан с PHP-переменной `$_POST[Price]`:

```
<?php
echo "Диапазон цен: $_POST[Price]";
...
```

Здесь пока нет абсолютно ничего нового, однако второй список в `listbox.html` отличается:

```
<select name="EngineSize[]" multiple>
  <option>1.0L</option>
  <option>1.4L</option>
  <option>1.6L</option>
  <option>2.0L</option>
</select>
```

По сути, в данном случае все выглядит так же, как и в предыдущем, кроме первой строки, в которой атрибуту `name` присваивается значение `EngineSize[]`. Добавление квадратных скобок (`[]`) в конце имени элемента управления — инструкция для PHP интерпретировать соответствующую переменную как массив.

```
$Choice0 = $_POST['EngineSize'][0];
$Choice1 = $_POST['EngineSize'][1];
$Choice2 = $_POST['EngineSize'][2];
$Choice3 = $_POST['EngineSize'][3];
echo "<br>Объем двигателя: $Choice0,";
echo "$Choice1,";
echo "$Choice2,";
echo "$Choice3,";
```

Как уже отмечалось, во время создания массива PHP создает новую переменную с тем же именем и присоединенным к имени индексным номером. В списке содержится четыре пункта, поэтому в массиве будет четыре элемента. Содержимое каждого из них необходимо отобразить. Поскольку индекс ссылается на элемент массива, а нумерация элементов в массивах, как правило, начинается с нуля, `$EngineSize[0]` ссылается на первый пункт в списке, 1.0L, если этот пункт был выбран пользователем. Этот элемент всегда содержит первый пункт списка, выбранный пользователем на странице.

В рассматриваемом примере выбран первый пункт, поэтому `$EngineSize[0]` действительно содержит значение 1.0L. То же касается и элемента `$EngineSize[1]`, связанного со вторым пунктом. Элементы `$EngineSize[2]` и `$EngineSize[3]` не содержат ничего, так как пользователь выбрал только два первых пункта списка. Если бы пользователь выбрал только один пункт, то значение хранилось бы только в элементе `$EngineSize[0]`. Элементы `$EngineSize[2]` и `$EngineSize[3]` будут содержать значения только в том случае, когда пользователь выберет все четыре пункта списка. Если бы были выбраны только два последних пункта, то переменные `$EngineSize[0]` и `$EngineSize[1]` содержали бы значения 1.6L и 2.0L соответственно, а переменные `$EngineSize[2]` и `$EngineSize[3]` вновь не имели бы значений. Для простоты отображения значений с помощью оператора `echo` для каждого пункта были созданы переменные и им были присвоены строковые значения переменных массива.

Необходимо помнить о том, что если отображение ошибок включено и ошибки видны в окне браузера (в файле `php.ini` должна быть строка `"display_errors = On"`), то будут также отображаться некоторые предупреждения о невыбранных значениях `EngineSize`, так как значения присваиваются не всем переменным `$_POST['EngineSize'] []`.

Скрытые поля форм

Часто возникает необходимость получить информацию, содержащуюся на одной Web-странице, и передать ее другой Web-странице без какого-либо ввода данных пользователем. В дескрипторе `<input>` существует еще один параметр, который позволяет передавать информацию в поле, как если бы оно было текстовым, скрывая при этом сам элемент управления и его значение от пользователя. Такое поле называется *скрытым полем формы* (или *скрытым элементом управления*).

Действие скрытых полей несколько отличается от действия уже рассмотренных элементов управления — они более полезны в РНР-страницах, содержащих формы, так как их можно использовать для отправки информации, хранящейся в РНР-переменных. Ниже приведен пример обычного скрытого поля формы:

```
<input type="hidden" name="Hidden1" value="Секретное сообщение">
```

Для этого примера нет снимка с экрана, так как данный элемент управления не отображается на странице. Любая форма, отправляющая такое скрытое поле, будет иметь переменную с именем `$Hidden1`, содержащую текст `Секретное сообщение`. Чтобы использовать данное скрытое поле в РНР-странице, можно написать всю HTML-форму в операторах `echo()`, передающих содержимое РНР-переменных через HTML-дескрипторы, как показано ниже:

```
<?php
$Message1="Этот текст невидимый";
echo "<form>";
echo "<input type='hidden' name='Hidden2' value='$Message1'>";
echo "<input type='submit' value='Отправить'>";
echo "</form>";
?>
```

Выше приведен пример целой HTML-формы, написанной в РНР-операторах, которая позволяет создать переменную `$Hidden2` и передать ей значение переменной `$Message1`. Естественно, это не единственный способ сохранять данные между отправками форм или запросами страниц; другие способы, такие как cookie-файлы, сесансы и т.д., рассматриваются в настоящей главе далее.

Рассмотрим пример, в котором на одной странице принимается содержимое списка, а на следующей странице отображается пользовательский выбор. Для создания HTML-формы в РНР также используется описанный выше код с `echo`-операторами.

Практика Использование скрытых полей форм

1. Откройте HTML-редактор и введите следующий код:

```
<html>
<head></head>
<body>
<?php
$Message1="Барс Банни";
$Message2="Гомер Симпсон";
$Message3="Рен и Стиппи";
echo "<form method='GET' action='hidden2.php'>";
echo "Кто из следующих персонажей победит в перестрелке?";
echo "<select name='ListBox'>";
echo "<option>$Message1</option>";
echo "<option>$Message2</option>";
echo "<option>$Message3</option>";
echo "</select><br><br>";
```



```

echo "<input type='hidden' name='Hidden1' value='$Message1'>";
echo "<input type='hidden' name='Hidden2' value='$Message2'>";
echo "<input type='hidden' name='Hidden3' value='$Message3'>";
echo "<input type='submit' value='Отправить'>";
echo "</form>";
?>
</body>
</html>

```

2. Сохраните файл как `hidden.php` и закройте его.
3. Создайте новый файл и введите в него следующий код:

```

<html>
<head><title></title></head>
<body>
<?php
echo "Было три варианта:<br>";
echo "$_GET[Hidden1]<br>";
echo "$_GET[Hidden2]<br>";
echo "$_GET[Hidden3]<br>";
echo "<br>Вы выбрали:<br>";
echo "$_GET[ListBox] ";
?>
</body>
</html>

```

4. Сохраните файл как `hidden2.php` и закройте его.
5. Откройте страницу `hidden.php` в браузере и выберите имя персонажа (рис. 3.16).

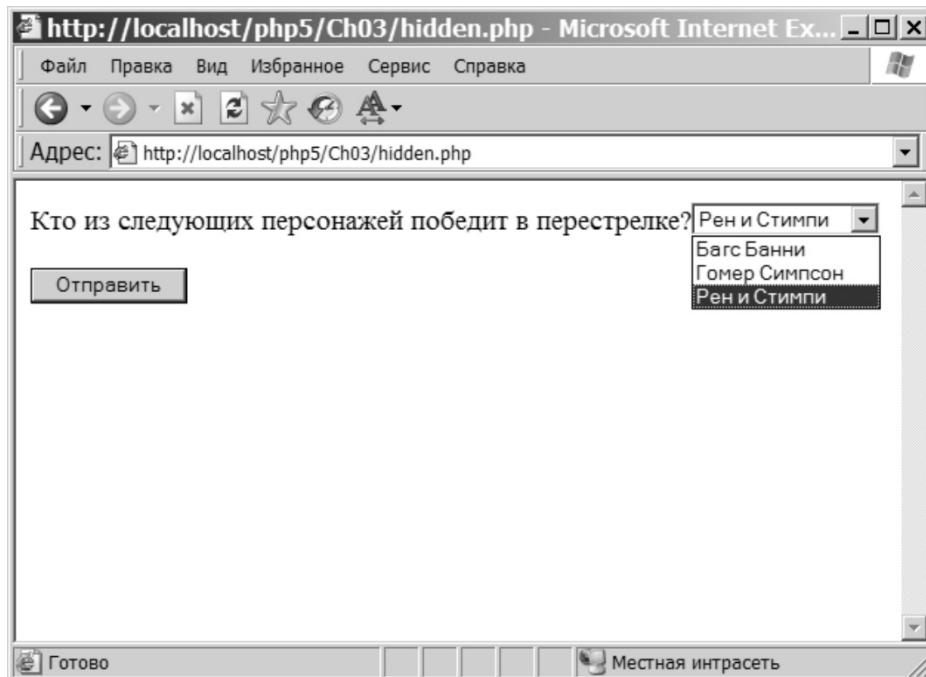


Рис. 3.16.

6. Нажмите кнопку Отправить, чтобы просмотреть результат (рис. 3.17).

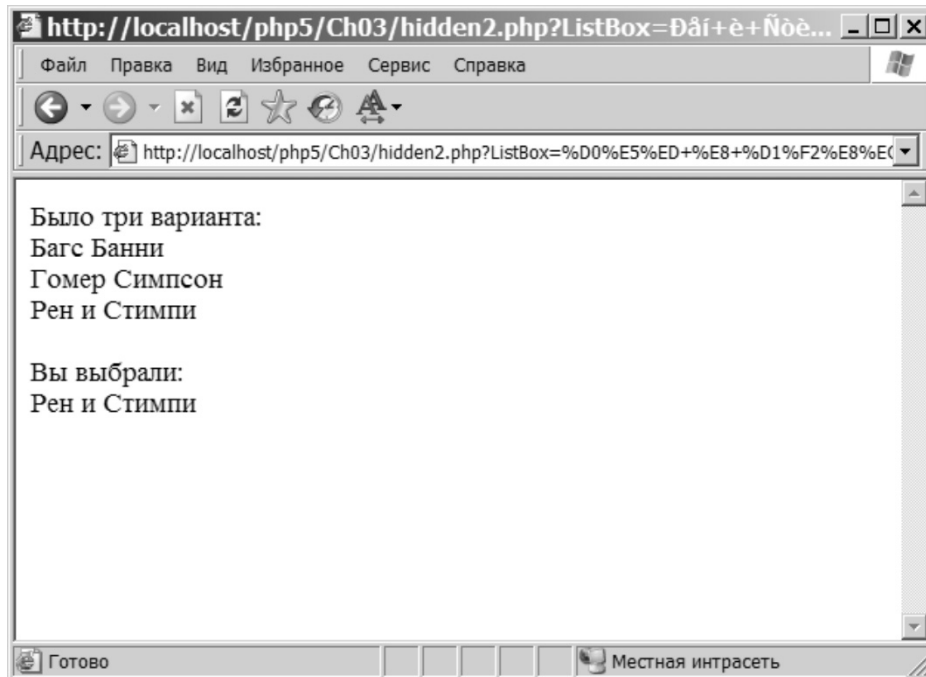


Рис. 3.17.

Как это работает

Процесс создания HTML-форм с помощью echo-операторов очень прост. Главное отличие этого способа от использования чистого HTML-кода в том, что можно опустить апострофы в именах переменных массива. Для формирования списка создается три переменных:

```
$Message1="Багс Банни";
$Message2="Гомер Симпсон";
$Message3="Рен и Стимпи";
```

Затем с помощью echo-операторов создается форма. Она ничем не отличается от обычной HTML-формы за исключением того, что там, где нужны кавычки, следует писать не двойные, а одинарные кавычки, иначе echo-оператор будет работать неправильно. Первая строка говорит только о том, что содержимое формы необходимо передавать сценарию hidden2.php методом GET:

```
echo "<form method='GET' action='hidden2.php'>";
```

Далее отображается вопрос, а затем начинается список:

```
echo "Кто из следующих персонажей победит в перестрелке?";
echo "<select name='ListBox'>";
```

В список включаются три пункта, значения переменных \$Message1, \$Message2 и \$Message3 соответственно.

```
echo "<option>$Message1</option>";
echo "<option>$Message2</option>";
echo "<option>$Message3</option>";
```

После этого записывается закрывающий тег `<select>` и добавляется два разрыва строки:

```
echo "</select><br><br>";
```

Затем три уже используемые переменные передаются в форму как скрытые поля:

```
echo "<input type='hidden' name='Hidden1' value='$Message1'>";
echo "<input type='hidden' name='Hidden2' value='$Message2'>";
echo "<input type='hidden' name='Hidden3' value='$Message3'>";
```

Три переменные превращаются в форме в переменные `$Hidden1`, `$Hidden2` и `$Hidden3` соответственно. В форму добавляется кнопка **Отправить**, после чего форма закрывается:

```
echo "<input type='submit' value='Отправить'>";
echo "</form>";
```

Вторая PHP-страница отображает содержимое элементов управления, созданных на первой странице. Сначала выводится содержимое трех скрытых полей формы:

```
echo "Было три варианта:<br>";
echo "$_GET[Hidden1]<br>";
echo "$_GET[Hidden2]<br>";
echo "$_GET[Hidden3]<br>";
```

Это полезно, потому что обычно содержимое всего списка не передается. На следующую PHP-страницу передается только выбранный пользователем пункт. Однако иногда в PHP-странице требуется доступ ко всему содержимому списка. Это один из эффективных методов передачи информации такого типа.

Последние строки отображают выбранный пользователем пункт.

```
echo "<br>Вы выбрали:<br>";
echo "$_GET[ListBox]";
```

Далее в книге скрытые поля используются для решения задач такого типа.

Поля ввода паролей

Поля ввода паролей, по сути, представляют собой текстовые поля, в которых вводимые символы заменяются звездочками. Они хранят и передают информацию так же, как и текстовые поля:

```
Введите пароль
<input name="Password" type="password">
```

Обработка полей ввода паролей и текстовых полей ничем не отличается, поэтому данный раздел не содержит примеров кода. Чтобы посмотреть поле ввода пароля в действии, можно вернуться к одному из предыдущих примеров (`text.html`) и исправить значение атрибута `type` на `password`. Следует, однако, отметить, что если для передачи данных из поля такого типа использовать метод GET, то пароль в строке запроса не шифруется и будет видимым для всех и каждого. Это не означает, что метод POST является безопасным методом передачи данных. Речь идет только о том, что информация при передаче методом POST непосредственно не показывается пользователю. Чтобы действительно обеспечить безопасность передаваемых данных, необходимо применять какой-либо специальный протокол, например, SSL (Secure Sockets Layer — протокол безопасных сокетов) для реального шифрования данных.

Кнопки submit и reset

Кнопка типа submit (отправить) много раз использовалась в данной главе, поэтому нет необходимости приводить отдельный пример для демонстрации ее работы. Однако следует отметить два момента. Во-первых, что произойдет, если в форму поместить несколько кнопок данного типа? В таком случае также необходимо задать значения атрибутов name и value для кнопок. Например:

```
<input value="Нажата кнопка 1" type="submit" name="Submit1">
<input value="Нажата кнопка 2" type="submit" name="Submit2">
```

Данный код, как и следовало ожидать, создает в PHP доступные переменные. Фактически этот код создает одну PHP-переменную в зависимости от того, какая кнопка нажата. Если нажать кнопку Submit1, то в массиве \$_GET или \$_POST будет создана переменная Submit1. Если нажать вторую кнопку, то будет создана переменная Submit2. Содержимым Submit1 является текст Нажата кнопка 1, тогда как содержимым Submit2 будет текст Нажата кнопка 2. Фактически с помощью данного кода нельзя выполнить что-нибудь полезное, поэтому примеры здесь не приводятся.

Во-вторых, кнопка типа submit не способна отменить ввод информации. Фактически невозможно отменить информацию, переданную с помощью кнопки типа submit. Однако существует такой элемент управления, как кнопка типа reset, которая отчасти решает эту проблему — с ее помощью можно установить все элементы управления на форме в их первоначальное состояние.

```
<input type="reset">
```

Использование в PHP-сценариях значений, возвращаемых формами

Выше были рассмотрены все разновидности элементов управления и обработка их содержимого в PHP, однако приводимые до сих пор в данной главе программы не делают ничего полезного с практической точки зрения, а лишь выводят содержимое элементов управления на другой Web-странице. Вероятно, без какой-либо из функций, которые рассматриваются в последующих главах, будет сложно манипулировать значением переменных. Однако из материала предыдущей главы читателю уже известны математические и строковые операторы, и эти знания можно объединить с понятиями, представленными в данной главе.

В последнем примере этой главы создается форма для подачи заявки на получение кредита, запрашивающая у пользователя размер денежной суммы, которую он хочет занять. После этого приложение вычисляет сумму, которую вымышленный банк NAMLLU может предложить заявителю на основании данных о возрасте и заработной плате. В конце расчетов приложение отвечает пользователю лишь “да” или “нет”.

Хотя формула расчета кредита может показаться сложной, на самом деле она весьма проста (и не основывается на формуле какой-либо реальной компании). Размер кредита для заявителя вычисляется по трем числам:

- ❑ переменная нормы зарплаты: годовая зарплата пользователя, разделенная на 5;
- ❑ переменная возрастного ценза: возраст пользователя, разделенный на 10; результат округляется в меньшую сторону до ближайшего целого числа, а затем уменьшается на единицу;
- ❑ переменная кредитной нормы: норма зарплаты, умноженная на возрастным цензом.

Приложение автоматически исключает всех пользователей моложе 20 лет, так как формула всегда возвращает нуль, а умножение любого числа на нуль в результате дает нуль. Ниже приведен пример вычисления кредитной нормы для пользователя 19 лет, где First figure — норма зарплаты:

```
First figure * (19/10 - (19 Modulus 10)/10) - 1
```

Оператор Modulus возвращает остаток от деления. В данном случае выражение можно упростить:

```
First figure * (1.9 - 0.9) - 1
```

что в результате дает:

```
First figure * 0
```

Для любого пользователя моложе 20 лет кредитная норма всегда равна нулю: так как один из множителей равен нулю, то размер зарплаты не имеет значения.

Рассмотрим другой пример: заявку на кредит подает пользователь 57 лет с годовой зарплатой 50 000 долларов.

- ☐ Норма зарплаты для этого пользователя равна $50000/5 = 10000$.
- ☐ Возрастной ценз равен $(57/10 - (57 \text{ Modulus } 10)/10) - 1 = 4$.
- ☐ Кредитная норма равна $10000 * 4 = 40000$ (норма зарплаты, умноженная на возрастной ценз).

Если кредитная норма больше, чем сумма займа, необходимая пользователю, то приложение выдает ответ “да”.

Практика Форма заявки на кредит

Программа состоит из двух страниц (как и почти все программы в этой главе). Первая страница принимает данные, из которых можно получить имя заявителя, его фамилию, возраст, адрес, заработную плату и желаемую сумму кредита. Вторая страница выполняет все вычисления и возвращает вердикт.

1. Откройте HTML-редактор и введите следующий код:

```
<html>
<head><title></title></head>
<body>
<b>Заявка на получение кредита в Namllu Credit Bank</b>
<form method="POST" action="loan.php">
Имя:
<input name="FirstName" type="text">
Фамилия:
<input name="LastName" type="text">
Возраст:
<input name="Age" type="text" size="3">
<br>
<br>
Адрес:
<textarea name="Address" rows="4" cols="40">
</textarea>
<br>
<br>
Выберите размер Вашей текущей зарплаты
<select name="Salary">
<option value=0>До $10000</option>
<option value=10000>$10000 - $25000</option>
<option value=25000>$25000 - $50000</option>
```

```

<option value=50000>Свыше $50000</option>
</select>
<br>
<br>
Выберите сумму необходимого кредита<br><br>
<input name="Loan" type="radio" value='1000'>$1000 под 8,0% годовых
<br>
<input name="Loan" type="radio" value='5000'>$5000 под 11,5% годовых
<br>
<input name="Loan" type="radio" value='10000'>$10000 под 15,0% годовых
<br>
<br>
<input type="submit" value="Подать заявку">
<input type="reset" value="Очистить">
</form>
</body>
</html>

```

2. Сохраните файл как `loan.html` и закройте его.
3. Создайте новый файл и введите следующий код:

```

<html>
<head><title></title></head>
<body>
<b>Заявка на получение кредита в Namllu Credit Bank</b>
<br>
<br>

<?php
$SalaryAllowance = $_POST['Salary']/5;
$AgeAllowance = ($_POST['Age']/10 - ($_POST['Age']%10)/10)-1;
$LoanAllowance = $SalaryAllowance * $AgeAllowance;
echo "Запрашиваемый кредит:$_POST[Loan]<br>";
echo "Допустимая сумма кредита:$LoanAllowance<br><br>";
if ($_POST['Loan'] <= $LoanAllowance) echo "Да, $_POST[FirstName]
$_POST[LastName], мы удовлетворим Вашу заявку";
if ($_POST['Loan'] > $LoanAllowance) echo "Извините,
$_POST[FirstName] $_POST[LastName], в настоящее время мы не
можем принять Вашу заявку";
?>
</body>
</html>

```

4. Сохраните данный файл как `loan.php`.
5. Откройте в браузере страницу `loan.html` (рис. 3.18) и введите необходимые сведения.
6. Нажмите кнопку **Подать заявку**, в результате чего должна появиться страница, показанная на рис. 3.19.

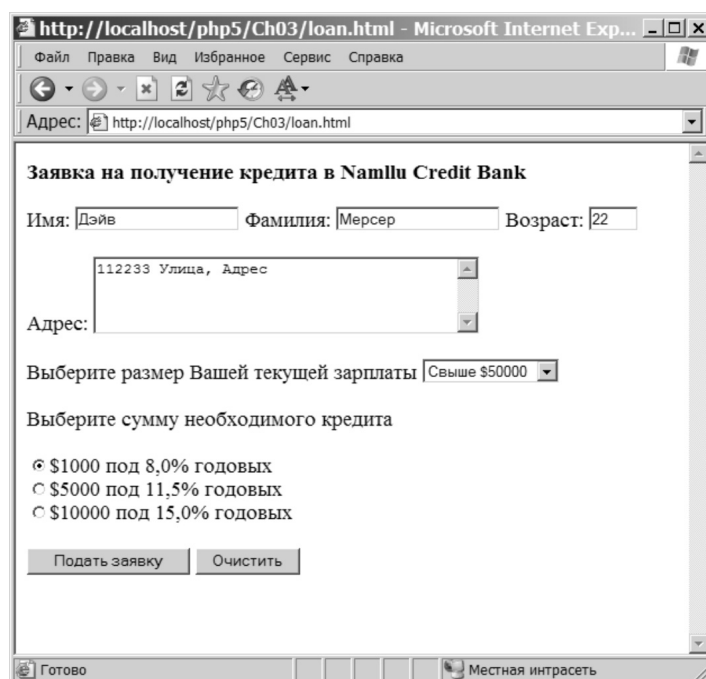
Как это работает

Первая страница состоит из довольно длинного кода, однако она не делает ничего экстраординарного и определенно не содержит ничего нового для изучения. В форме (`loan.html`) содержится восемь элементов управления. Первые три — текстовые поля, используемые для получения имени, фамилии и возраста заявителя:

```

<input name="FirstName" type="text">
Фамилия:
<input name="LastName" type="text">
Возраст:
<input name="Age" type="text" size="3">

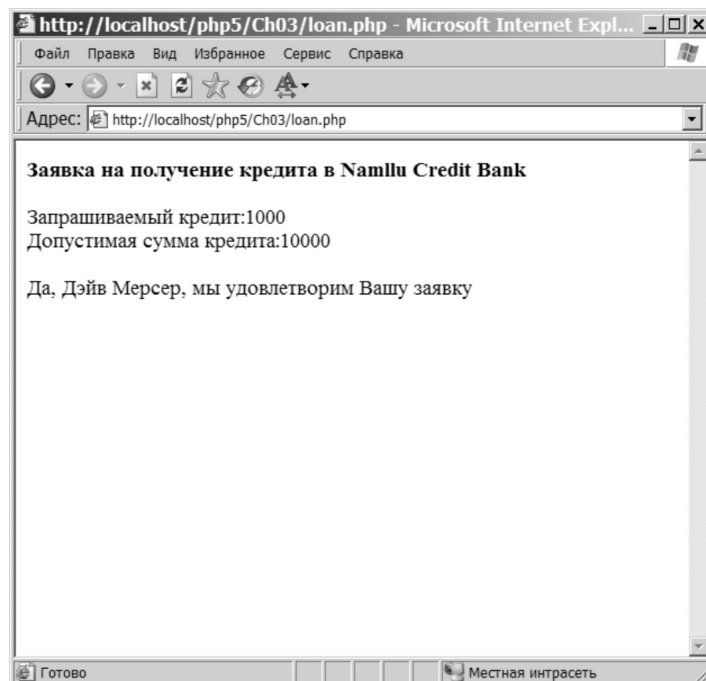
```



The screenshot shows a Microsoft Internet Explorer window with the address bar displaying `http://localhost/php5/Ch03/loan.html`. The page title is "Заявка на получение кредита в Namllu Credit Bank". The form contains the following fields and controls:

- Имя: Фамилия: Возраст:
- Адрес:
- Выберите размер Вашей текущей зарплаты:
- Выберите сумму необходимого кредита:
 - ☒ \$1000 под 8,0% годовых
 - ☐ \$5000 под 11,5% годовых
 - ☐ \$10000 под 15,0% годовых
- Buttons:

Рис. 3.18.



The screenshot shows the same Microsoft Internet Explorer window, but the address bar now displays `http://localhost/php5/Ch03/loan.php`. The page title remains "Заявка на получение кредита в Namllu Credit Bank". The form displays the following information:

- Запрашиваемый кредит: 1000
- Допустимая сумма кредита: 10000
- Message: Да, Дэйв Мерсер, мы удовлетворим Вашу заявку

Рис. 3.19.

Очевидно, что они создают на PHP-странице переменные `$_POST[FirstName]`, `$_POST[LastName]` и `$_POST[Age]` соответственно.

Адрес заявителя вводится в текстовую область:

```
<textarea name="Address" rows="4" cols="40">
</textarea>
```

Текстовая область в свою очередь создает PHP-переменную `$_POST[Address]`. В этом примере используются не все PHP-переменные, созданные данной формой, но в последующих главах рассматриваются аналогичные примеры, где применяются другие переменные.

Следующим элементом управления является выпадающий список, содержащий несколько диапазонов зарплаты:

```
<select name="Salary">
<option value=0>До $10000</option>
<option value=10000>$10000 - $25000</option>
<option value=25000>$25000 - $50000</option>
<option value=50000>Свыше $50000</option>
</select>
```

На практике сохранить весь диапазон как значение нельзя. Вместо этого в качестве значения определенного пункта списка сохраняется наименьшая граница соответствующего диапазона. При этом создается только одна PHP-переменная `$_POST[Salary]`, в которой хранится значение, связанное с диапазоном, выбранным пользователем. Если пользователь не выбрал диапазон, то список вообще не возвращает значение. Следует отметить, что первое значение равно нулю, и согласно приведенной выше формуле приложение автоматически возвращает пользователю, зарплата которого ниже 10 000 долларов, отрицательный ответ.

Следующий элемент управления — группа связанных друг с другом переключателей:

```
Выберите сумму необходимого кредита<br><br>
<input name="Loan" type="radio" value='1000'>$1000 под 8,0% годовых
<br>
<input name="Loan" type="radio" value='5000'>$5000 под 11,5% годовых
<br>
<input name="Loan" type="radio" value='10000'>$10000 под 15,0% годовых
```

Все переключатели имеют одинаковые имена, так как в переменной должно содержаться только одно значение в зависимости от выбора пользователя. Группа, состоящая из трех переключателей, создает только одну PHP-переменную, `$Loan`.

Два последних элемента управления — кнопки типа `submit` и `reset`:

```
<input type="submit" value="Подать заявку">
<input type="reset" value="Очистить">
```

Кнопка типа `submit` использует значение атрибута `action`, установленного в начале формы, и поэтому “знает”, куда следует отправлять информацию формы:

```
<form method="POST" action="loan.php">
```

Очевидно, что первая программа сохраняет и передает информацию формы, а вторая программа, `loan.php`, принимает переданные значения и выполняет некоторые простые операции над ними, чтобы принять или отклонить заявку на кредит. В первой строке создается новая переменная `$SalaryAllowance`, значение которой равно зарплате пользователя, разделенной на 5:

```
$SalaryAllowance = $_POST['Salary']/5;
```

Во второй строке вычисляется более сложная формула возрастного ценза, результатом которой должно быть целое число, зависящее от разделенного на 10 возраста

пользователя. Остаток от деления (если он есть) возраста пользователя на 10 удаляется путем округления в меньшую сторону до ближайшего целого числа. Для вычисления остатка используется оператор Modulus (%). После этого из полученного числа вычитается единица, и результат присваивается переменной. Последняя строка возвращает 0, если пользователь ввел возраст от 0 до 19 лет; 1, если введенный возраст попадает в промежуток 20–29 лет; 2, если возраст попадает в промежуток 30–39 лет, и т.д. Результат вычислений хранится в новой переменной \$AgeAllowance:

```
$AgeAllowance = ($_POST['Age']/10 - ($_POST['Age']/10)%10)/10-1;
```

Следующая строка значительно короче. В ней два только что вычисленных значения перемножаются и результат сохраняется в новой переменной \$LoanAllowance, которая представляет собой окончательное число, ограничивающее размер возможного займа.

```
$LoanAllowance = $SalaryAllowance * $AgeAllowance;
```

Следующие две строки выводят на Web-странице подтверждение введенного пользователем размера займа и сумму допустимого займа:

```
echo "Запрашиваемая ссуда:$_POST[Loan]<br>";  
echo "Допустимая ссуда:$LoanAllowance<br><br>";
```

В двух последующих строках используется оператор <= (меньше или равно), который позволяет принять решение на основе полученной информации. Данный оператор определяет, не превышает ли необходимый пользователю размер ссуды сумму, допустимую банком, и если не превышает, то на Web-странице отображается сообщение, подтверждающее принятие заявки. Данная структура детально обсуждается в следующей главе, поэтому здесь она рассмотрена очень кратко.

Отображаемое сообщение содержит персональную информацию (имя и фамилию), введенную пользователем в форму:

```
if ($_POST['Loan'] <= $LoanAllowance) echo "Да, $_POST[FirstName]  
$_POST[LastName], мы удовлетворим Вашу заявку";
```

В последней строке PHP-сценарий обрабатывает ситуацию отклонения заявки — когда сумма, которую пользователь хочет занять в банке, превышает допустимую банком сумму. В этом случае на Web-странице отображается сообщение об отклонении заявки:

```
if ($_POST['Loan'] > $LoanAllowance) echo "Извините, $_POST[FirstName]  
$_POST[LastName], в настоящее время мы не можем принять Вашу заявку";
```

Это практически все, что касается данных программ. Есть еще одна деталь: информация в реальном приложении такого рода конфиденциальна, поэтому для передачи данных формы следует использовать метод POST. При этом следует помнить, что этот метод скрывает информацию от рядового пользователя, однако хакеры легко могут похитить информацию, передаваемую с помощью данного метода. Чтобы обеспечить реальную безопасность, необходимо применять SSL-сертификат для шифрования обмена данными между пользователем и Web-сервером.

Возможные усовершенствования приложения

Рассмотренное выше приложение нельзя называть идеальным. Если приложить некоторые усилия, его работу можно нарушить или заставить его отображать нелогичные значения. Это возможно, потому что корректность значений, полученных от пользователя, никак *не проверяется*. Как можно предотвратить ввод абсолютно неправильных значений, например, 965 в поле возраста? Очевидно, что такое значение не

может быть верным, но помешать вводу такого числа невозможно. В последующих главах рассматриваются способы доработки подобных приложений путем проверки значений и передачи на обработку только тех значений, которые находятся в пределах определенного диапазона.

Понятие состояния

В общих чертах многие пользователи компьютеров представляют себе работу таких настольных приложений, как Excel, Word, Dreamweaver и др. Пользователь запускает приложение, оно появляется на экране и дает возможность работать с документами посредством пунктов меню, диалоговых окон, элементов управления и полей форм. Может показаться, что нет никакого разделения между логикой обработки данных в приложении и пользовательским интерфейсом. На самом деле всякий раз, когда пользователь выбирает какой-либо пункт меню, нажимает кнопку или вводит какие-либо данные в поле формы, он фактически использует пользовательский интерфейс, а пользовательский интерфейс обменивается данными с программной логикой внутри приложения. Затем приложение отвечает, выполняя обработку и выводя какой-либо ответ в пользовательском интерфейсе. Некоторые действия пользователя могут не иметь отклика, тогда как другие, очевидно, приводят к выполнению огромного объема скрытой от пользователя обработки данных.

Как только пользователь посредством пользовательского интерфейса вводит какие-либо данные в приложение, изменяется общее *состояние* данного приложения. Состояние приложения можно описать как точное мгновенное состояние всех данных и переменных в приложении. Приложение должно отмечать изменения состояния, поскольку эти изменения могут требовать какого-либо отклика (который в свою очередь вызывает дальнейшие изменения состояния приложения).

Кроме взаимодействия пользователя с приложением, существует множество обстоятельств, которые могут изменить состояние приложения. Например, предположим, что пользователь играет в игру и у него в распоряжении 15 секунд для перемещения. Приложение отслеживает системное время и может выполнять действия, абсолютно независимые от действий пользователя (кроме самого перемещения), в зависимости от истекшего периода времени (еще один случай изменения состояния). Какое отношение это имеет к Web-приложениям?

Программист может не задумываться о состоянии при программировании настольных приложений, потому что обычно взаимодействие между пользовательским интерфейсом и программной логикой незаметно. Однако программирование для Web вынуждает программиста учитывать эту проблему, поскольку протокол HTTP не фиксирует состояние соединений. HTTP — протокол без фиксации состояний, так как он не имеет встроенного механизма, отслеживающего состояние каждого объекта в пользовательском интерфейсе и информирующего программную логику об изменении этого состояния.

Вместо этого обмен данными между браузером и сервером принимает форму нажатия на ссылки или кнопки отправки форм. Этот процесс зависит не только от пользователя (иначе его можно было бы автоматизировать) — на медленных каналах или прерывистых соединениях это может занимать длительное время. Поэтому создание Internet-приложений на PHP требует использования других механизмов для контроля состояния, а программисту приходится принимать меры, чтобы справиться с тем, что пользовательский интерфейс отключился или вообще не отвечает так, как это ожидается.

Контроль состояния

Часто программист сталкивается с проблемой сохранения состояния в PHP-программах, когда требуется сохранять значения переменных между запросами страниц, поскольку каждая страница является отдельной программой, а как только программа выполнила обработку, все переменные и значения теряются — программа завершила работу, и все данные стерлись из памяти. В отличие от настольных приложений, которые хранят данные в памяти до тех пор, пока пользователь не закроет приложение, PHP-программа запускается только после запроса, активизирующего данную страницу, и работает до тех пор, пока не будет завершена обработка и/или HTML или текстовое содержимое не будет возвращено пользователю.

PHP способен заполнить пустоту между циклами запрос-ответ, обеспечивая постоянство данных, продлевая, таким образом, интерактивность. Этот процесс непрерывного обновления серверной информации обеспечивает работу большинства интерактивных сайтов и жизненно необходим для поддержки сеансов. Поддержка сеансов — возможность обращаться к последовательности интерактивных операций, которая начинается при посещении или регистрации пользователя на сайте и заканчивается, когда пользователь уходит с сайта или долгое время не проявляет активности.

Ключевым компонентом данного процесса обычно является *cookie-файл* (небольшой блок данных, хранящийся на клиентском компьютере) или некоторая переменная, пересылаемая между клиентом и сервером, которая часто называется *идентификатором сеанса*. Cookie-файлы и идентификаторы сеансов используются для того, чтобы сервер “знал”, с каким клиентом он взаимодействует в процессе того или иного запроса. Идентификатор сеанса (или соответствующий cookie-файл) идентифицирует клиента на Web-сайте так же, как имя пользователя идентифицирует данного пользователя в операционной системе.

Сеансы также поддерживают состояние между запросами страниц (они подробно обсуждаются далее), однако сперва рассмотрим некоторые другие доступные методы, включая использование скрытых полей форм, строк запроса, баз данных и cookie-файлов.

Скрытые поля форм

Ранее уже освещалась работа скрытых полей форм при отправке серверу данных (жестко определенных разработчиком). Чтобы поддерживать состояние с помощью скрытых полей форм (например, сохраняя уникальное значение, представляющее продукт, с которым работает пользователь, такое как идентификатор продукта), можно просто поместить значение текущего идентификатора продукта в скрытое поле формы. Идея заключается в том, чтобы дать пользователю возможность выбрать продукт из выпадающего списка (дескриптора select), а затем отправить форму:

```
<form action="myform.php" method="post">
<select name="selected_product_id">
<option value="121">Продукт 121</option>
<option value="122">Продукт 122</option>
</select>
<input type="submit" name="button" value="Выбрать продукт">
</form>
```

После этого пользователю можно вернуть страницу со следующим кодом:

```
<input type="hidden" name="chosen_product_id" value="<?php echo
$selected_product_id; ?>">
```

До того как страница будет возвращена пользователю, его выбор будет обработан как такой код (предположим, пользователь выбрал продукт 122):

```
<form action="myform.php" method="post">
<input type="hidden" name="chosen_product_id" value="122">
```

Таким образом, когда пользователь в следующий раз отправит форму, в отправленных данных будет поле `chosen_product_id`, которое будет преобразовано PHP в переменную `$chosen_product_id`. Эту переменную затем можно будет использовать для обработки информации по выбранному продукту независимо от того, на какой странице пользователь выбрал данный продукт. Однако нельзя забывать, что каждый раз, когда пользователь запрашивает другую страницу, в коде этой страницы должно быть то же скрытое поле с тем же значением. В противном случае это значение будет потеряно.

Строки запроса

Чтобы поддерживать значения состояния между запросами страниц, можно использовать строки запроса так же, как скрытые поля форм. Однако, как уже отмечалось, пары имя/значение строк запроса будут отображаться в адресной строке браузера вместе с URL. Такой способ передачи данных очень ненадежен, а, кроме того, он предоставляет злоумышленнику возможность прервать, перехватить или иным образом вмешаться в передачу информации.

Базы данных

Механизм работы баз данных и доступа к ним обсуждается в главах 9, 10 и 11, однако общеизвестно, что базы данных используются для постоянного (т.е. данные сохраняются даже после выключения сервера) хранения структурированной информации, поэтому очевидно, что базы данных можно применять для хранения данных между запросами страниц. По существу, если разработчик не против издержек создания запросов к базе данных при каждом обращении к странице, то для хранения всех постоянных данных между запросами страниц можно использовать таблицу базы данных. Такой подход, безусловно, имеет свои недостатки — издержки в соединениях с базой данных и дополнительные усилия на создание самой базы данных.

Cookie-файлы

Cookie-файлы представляют собой быстрый (и, по мнению некоторых программистов, неорганизованный) метод хранения (на клиентском компьютере) небольших блоков данных, которые должны быть постоянными между отдельными визитами пользователя на Web-сайт. Этот метод нельзя назвать ни очень мощным, ни очень надежным средством — cookie-файлы определено не следует использовать для постоянного хранения данных, так как простая смена браузера клиентом приведет к полному уничтожению сохраненных ранее cookie-файлов. Вместе с тем во многих ситуациях cookie-файлы могут оказаться очень удобными. Ниже перечислены некоторые наиболее распространенные примеры использования cookie-файлов.

- ❑ Хранение пользовательских настроек для определенного сайта.
- ❑ Хранение ключа (или ключей), которые можно использовать для связи пользователей с их персональными данными — как это делается в бесчисленных приложениях типа покупательских тележек.

- ❑ Обеспечение квазипостоянного идентификатора сеанса, позволяющего пользователю оставаться зарегистрированным на сайте до тех пор, пока он явно не покинет сайт или не закроет браузер.

Cookie-файлы являются наилучшим способом хранения небольших, полезных, но не критичных блоков данных. Один из самых лучших примеров использования cookie-файлов — хранение настроек, позволяющих изменять внешний вид сайта согласно потребностям пользователя. Хотя возможность настраивать под себя цветовую схему сайта очень удобна для пользователей, те пользователи, которые не могут (или не хотят) использовать cookie-файлы, теряют не многое.

Задуманные вначале как безвредные “помощники” Web-разработчиков, cookie-файлы в последние годы заслужили плохую репутацию. Часто ими злоупотребляют (например, хранят в них большие объемы данных, которые было бы лучше хранить на стороне сервера), а иногда используют не по назначению (например, для накопления информации о клиентах без их на то согласия). Однако при умеренном и ответственном использовании cookie-файлы могут оказаться весьма полезными, в частности в таких ситуациях:

- ❑ когда известно, что поддержка cookie-файлов включена у всех посетителей сайта, например, в корпоративных и образовательных интрасетях;
- ❑ для добавления сайту “бантиков” — функций, которые делают внешний вид сайта более привлекательным, но для использования сайта фактически не требуются.

Использование cookie-файлов

Процесс реализации cookie-файлов отличается от браузера к браузеру, но можно выделить некоторые важные моменты, общие для всех cookie-файлов:

- ❑ cookie-файл представляет собой небольшой блок данных, который можно использовать для хранения имени и значения переменной наряду с информацией о сайте, с которого поступил данный cookie-файл, и сроке его действия;
- ❑ cookie-файлы обеспечивают хранилище данных на клиентской стороне в виде файлов, сохраненных на жестком диске клиентской машины;
- ❑ Web-сайт обычно может модифицировать только созданные им самим cookie-файлы;
- ❑ доступ к cookie-файлам и их изменение (при удовлетворении соответствующих критериев безопасности) может быть предоставлен только Web-серверу, с которого они первоначально были отправлены.

Во время визита клиента на Web-сайт, использующий cookie-файлы, Web-сервер инструктирует клиента (т.е. Web-браузер) сохранить у себя определенную информацию для последующего использования. Ответственность за сохранение этой информации ложится на клиента. Поддерживающие cookie браузеры решают эту задачу путем сохранения данных в файле, названном по имени сайта, которому принадлежит cookie, в специально зарезервированном для этой цели каталоге. При последующих обращениях к этому же сайту клиент отправляет серверу копию данных — данные на клиентской стороне остаются постоянными в течение заданного срока действия cookie, по истечении которого файл удаляется из системы. Срок действия задается сервером, когда сервер отправляет клиенту инструкцию о сохранении cookie-файла, и, по существу, представляет собой число секунд, в течение которых клиент должен хранить этот файл. Если сервер назначает срок действия, равный нулю секунд, то браузер должен хранить такой файл только до тех пор, пока пользователь не закроет браузер.

Поскольку cookie-файлы хранятся на стороне клиента, они выходят из-под контроля сервера сразу после создания. Пользователи могут удалять cookie-файлы самостоятельно путем нажатия определенной кнопки в Web-браузере или вручную, они также могут редактировать содержимое данных файлов, поэтому невозможно быть уверенным в том, что браузер вернет правильные данные.

По существу, cookie представляют собой сообщения сервера вроде “эти данные нужно запомнить; напомни мне, когда вернешься в следующий раз”. “Следующий раз” может наступить когда угодно: когда пользователь нажмет на ссылку двумя секундами позже или когда вернется на сайт через неделю. Это и есть постоянство данных. Cookie-файлы подобны именным табличкам, которые идентифицируют делегатов конференции до тех пор, пока делегаты их носят.

Web-серверы отправляют клиентам cookie-данные в HTTP-заголовках, которые передаются раньше любого HTML-текста. Аналогично браузеры возвращают cookie-данные, также используя HTTP-заголовки. Клиент определяет, какие cookie-файлы отправлять Web-сайту, на основании имени сервера и запрашиваемой в текущий момент страницы. Поэтому если пользователь заходит на сайт `www.php.net`, то браузер не отправляет этому сайту cookie, полученные от сайта `www.wrox.com`.

При установке cookie может устанавливаться имя сервера и путь (хотя это необязательно) — это разрешает доступ к данному cookie-файлу только с заданного сервера и/или пути на этом сервере. Клиенты используют данную информацию для того чтобы узнать, следует ли отправлять серверу определенный cookie-файл. Браузер с включенной поддержкой cookie обычно отправляет в заголовке все cookie-файлы, которые (по мнению браузера) принадлежат этому сайту.

Запись и считывание cookie-файлов

PHP, как и подобает современному языку Web-сценариев, полностью поддерживает cookie, а отправка cookie осуществляется с помощью вызова функции `setcookie()`. Подобно функции `header()`, функцию `setcookie()` необходимо вызывать до того, как в браузер будет выведен какой-либо HTML-текст, поскольку cookie отправляются в HTTP-заголовках, которые должны отсылаться раньше HTML-кода.

Функция `setcookie()` принимает шесть параметров, самыми важными из которых являются первые три:

1. Строка, используемая в качестве имени переменной.
2. Строка, используемая в качестве значения этой переменной.
3. Временная метка Unix, означающая срок действия cookie-файла.

Временная метка Unix — это длинное целое число, которое представляет количество секунд, прошедших с полуночи 1 января 1970 года. Текущее время в форме временной метки Unix можно получить с помощью функции `time()`. Чтобы установить cookie, срок действия которого заканчивается через один час после установки, можно вызвать функцию `setcookie()`, передав ей в качестве третьего параметра выражение `time()+3600`.

Три последние параметра функции `setcookie()` используются не так часто; их описания приведены ниже.

- Путь, к которому относятся cookie-файлы; браузер не возвращает серверу cookie-файлы, установленные с несоответствующих путей. Например, если значением этого параметра при установке cookie был путь `/my/path/number/one`, то при

переходе на страницу `/my/path/number/two` браузер не сможет отправить этот cookie-файл — cookie-файл будет отправлен только при возвращении посетителя на страницу `/my/path/number/one`.

- ❑ Домен, к которому относится cookie-файл; здесь действуют те же правила, что и в предыдущем случае. Параметр может оказаться полезным, когда Web-сервер поддерживает несколько доменов.
- ❑ Параметр `secure` — целое число. Если оно равно 1, то cookie-файл будет передаваться только через SSL-шифрованное соединение. (Cookie-файл на жестком диске клиента будет храниться в незашифрованном виде; параметр `secure` гарантирует лишь то, что cookie будет шифроваться для передачи через Internet.)

В простейшем случае последние три параметра можно опустить, поэтому типичный вызов функции `setcookie()` может выглядеть так:

```
setcookie("fontprefs", "", time()+3600);
```

Доступ к установленным cookie-файлам осуществляется еще проще — для этого не требуется вызывать какую-либо функцию вообще. Так же как и в случае POST-переменных, PHP автоматически помещает информацию cookie в глобальную область, поэтому использовать значения, сохраненные в cookie, так же просто, как использовать любые другие переменные. Например, значение полученного cookie с именем `fontprefs` автоматически будет доступно во всем сценарии как значение глобальной переменной `$_COOKIE['fontprefs']`.

Существует несколько способов удаления cookie-файлов. Конечно, если пользователь знает, где искать cookie-файлы на своей машине, то он всегда может изменять или удалять эти файлы. Однако иногда необходимо заставить сервер удалить свой cookie-файл и для этого существует две возможности:

- ❑ заменить дату окончания действия cookie датой в прошлом, например, `setcookie("num", "0", time()-9999);`
- ❑ переустановить cookie-файл, указав только его имя, например, `setcookie("fontprefs");`.

Практика Использование cookie-файлов для хранения пользовательских настроек

Ниже приведен сценарий, который сохраняет в cookie-файле выбранные пользователем размер и гарнитуру шрифта. При последующих визитах пользователя на эту страницу сценарий проверяет cookie-файл и восстанавливает сохраненные в нем настройки. Сохраните следующий код как `cookies.php`:

```
<?php
//cookies.php
if ($_POST[type_sel]) {
    setcookie("font[type]", $_POST[type_sel], time()+3600);
}

if ($_POST[size_sel]) {
    setcookie("font[size]", $_POST[size_sel], time()+3600);
}
//Определим размер и гарнитуру шрифта, и, поскольку
//пока еще можно добавить HTML-заголовок, добавим его:
$type = array("arial", "helvetica", "sans-serif", "courier");
$size = array("1", "2", "3", "4", "5", "6", "7");
```

```

echo "<html><head><title>Тест cookie</title></head><body><div align='center'>";

//В данной форме содержится два списка, которые можно
//использовать для определения пользовательских предпочтений:
echo "<form method='POST'>";
echo "Какой шрифт Вы хотели бы использовать? ";
echo "<select name='type_sel'>";
echo "<option selected value=''>по умолчанию</option>";
foreach ($type as $var) {
    echo "<option>$var</option>";
}
echo "</select><br><br>";
echo "Какой размер шрифта Вы хотели бы использовать? ";
echo "<select name='size_sel'>";
echo "<option selected value=''>по умолчанию</option>";

foreach ($size as $var) {
    echo "<option>$var</option>";
}
echo "</select><br><br>";
echo "<input type='submit' value='Получить cookie'>";
echo "</form>";

//Наконец, выведем некоторую полезную информацию и отформатируем ее в
//соответствии с выбранными пользователем настройками:
echo "<b> Cookie-информация:</b><br>";
echo "<font ";
if ($_COOKIE[font][type]) {
    $cookie_font_type = $_COOKIE[font][type];
    echo "гарнитура='$cookie_font_type' ";
}

if ($_COOKIE[font][size]) {
    $cookie_font_size = $_COOKIE[font][size];
    echo "размер='$cookie_font_size' ";
}
echo ">";
echo "\$font[type] = $cookie_font_type<br>";
echo "\$font[size] = $cookie_font_size<br>";
echo "</font><br>";
echo "<b> Информация переменных формы:</b><br>";
echo "<font ";

if ($_POST[type_sel]) {
    $post_type_sel = $_POST[type_sel];
    echo "гарнитура='$post_type_sel' ";
}

if ($_POST[size_sel]) {
    $post_size_sel = $_POST[size_sel];
    echo "размер='$post_size_sel' ";
}
echo ">";
echo "\$type_sel = $post_type_sel<br>";
echo "\$size_sel = $post_size_sel<br>";
echo "</font>";

echo "</div></body></html>";
?>

```

Откройте страницу `cookies.php` в браузере и поэкспериментируйте со списками (рис. 3.20).

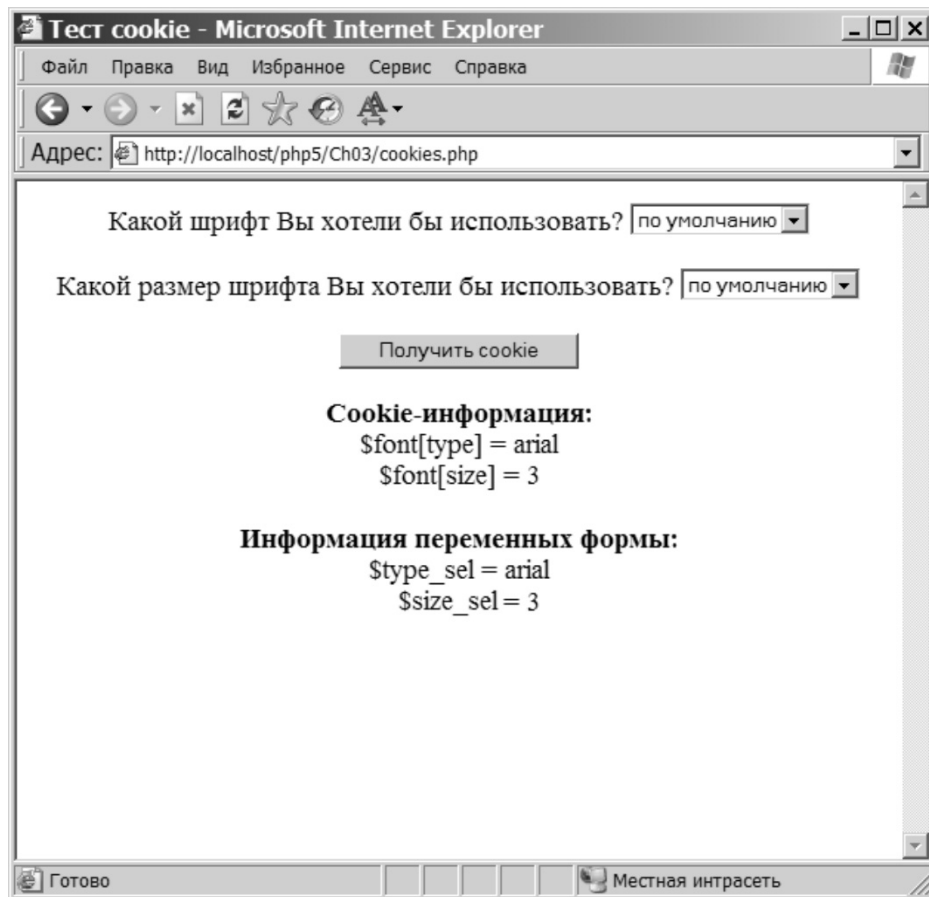


Рис. 3.20.

Как это работает

Наиболее интересными являются первые две функциональные строки данного сценария. Напомним, что cookie отправляются в HTTP-заголовках, поэтому вызовы необходимо поместить до вывода какого-либо HTML-кода:

```
<?php
//cookies.php
if ($_POST[type_sel]) {
    setcookie("font[type]", $_POST[type_sel], time()+3600);
}

if ($_POST[size_sel]) {
    setcookie("font[size]", $_POST[size_sel], time()+3600);
}
```

Для каждого cookie-файла устанавливается срок действия в течение одного часа.

Затем определяются массивы, содержащие доступные гарнитуры и размеры шрифта, после чего для определения всех возможных вариантов в форму помещаются выпадающие списки, использующие данные массивы. После отправки формы выбранные пользователем значения отправляются методом POST обратно сценарию:

```
$type = array("arial", "helvetica", "sans-serif", "courier");
$size = array("1", "2", "3", "4", "5", "6", "7");
echo "<html><head><title>Тест cookie</title></head><body><div align='center'>";

//В данной форме содержится два списка, которые можно использовать для
//определения пользовательских предпочтений:
echo "<form method='POST'>";
echo "Какой шрифт Вы хотели бы использовать? ";
echo "<select name='type_sel'>";
echo "<option selected value='>по умолчанию</option>";
foreach ($type as $var) {
    echo "<option>$var</option>";
}
echo "</select><br><br>";
echo "Какой размер шрифта Вы хотели бы использовать? ";
echo "<select name='size_sel'>";
echo "<option selected value='>по умолчанию</option>";

foreach ($size as $var) {
    echo "<option>$var</option>";
}
echo "</select><br><br>";
echo "<input type='submit' value='Получить cookie'>";
echo "</form>";

    Наконец, отображаются cookie- и POST-переменные, показывая, как изменяются
    их значения в зависимости от выбора пользователя. Отображаемые переменные
    форматируются согласно соответствующим значениям:

//Наконец, выведем некоторую полезную информацию и отформатируем ее в
//соответствии с выбранными пользователем настройками:
echo "<b> Cookie-информация:</b><br>";
echo "<font ";
if ($_COOKIE[font][type]) {
    $cookie_font_type = $_COOKIE[font][type];
    echo "гарнитура='$_cookie_font_type' ";
}

if ($_COOKIE[font][size]) {
    $cookie_font_size = $_COOKIE[font][size];
    echo "размер='$_cookie_font_size' ";
}
echo ">";
echo "\$font[type] = $_cookie_font_type<br>";
echo "\$font[size] = $_cookie_font_size<br>";
echo "</font><br>";
echo "<b> Информация переменных формы:</b><br>";
echo "<font ";

if ($_POST[type_sel]) {
    $post_type_sel = $_POST[type_sel];
    echo "гарнитура='$_post_type_sel' ";
}

if ($_POST[size_sel]) {
    $post_size_sel = $_POST[size_sel];
    echo "размер='$_post_size_sel' ";
}
echo ">";
echo "\$type_sel = $_post_type_sel<br>";
echo "\$size_sel = $_post_size_sel<br>";
echo "</font>";

    echo "</div></body></html>";
?>
```

В сценарии нет ничего сложного. Когда страница `cookies.php` вызывается пользователем впервые, как `cookie`-переменные, так и переменные формы пусты. Если выбрать какие-либо параметры в списках и нажать кнопку **Получить cookie**, то переменным формы будут присвоены значения, которые тут же будут выведены на экран.

Это говорит о том, что переменные формы `type_sel` и `size_sel` получают значения, соответствующие пользовательскому выбору при первом посещении страницы. Следует отметить, что обе `cookie`-переменные все равно пусты. Они присутствуют, но для того чтобы вывести их, необходимо обновить страницу.

Собственные сеансы в PHP

Сеансом можно назвать последовательность взаимосвязанных действий между одним клиентом и Web-сервером, которая имеет место в течение длительного периода времени. Это может быть последовательность транзакций, которые осуществляются пользователем во время обновления портфеля ценных бумаг, или множество запросов, которые выполняются при проверке почтового ящика через Web-интерфейс e-mail-службы. Сеанс может состоять из нескольких запросов к одному сценарию или из запросов к различным ресурсам на одном и том же Web-сайте.

В частности, когда возникает необходимость обрабатывать секретные или объемные данные, имеет смысл переслать их однажды и сохранить на сервере, а не хранить на клиентской машине и пересылать каждый раз между клиентом и сервером. Гораздо практичнее хранить данные на сервере, а клиенту выдать “ключ”, позволяющий ему уникально идентифицировать себя на этом сервере и, следовательно, использовать любые связанные с этим ключом серверные данные. Такой ключ называется *идентификатором сеанса* (*session identifier*); он однозначно связывает клиента с сеансом и, следовательно, с его данными. В PHP идентификатор сеанса называется `SID` (`Session ID`) и представляет собой специальную переменную, которая определена как регистрационный номер конкретного сеанса.

В этой главе уже было показано, как установить сеанс, используя `cookie` для сохранения данных на клиентской машине. Такой метод управления сеансами небезопасен, однако PHP имеет встроенную поддержку управления сеансами, поэтому разбираться в точных деталях реализации нет необходимости. PHP создает `SID` каждый раз, когда в сценарии вызывается функция `session_start()`, а также по умолчанию, когда используются некоторые другие связанные с сеансами функции, такие как `session_register()`. Значение `SID` хранится в глобальной переменной с именем `PHPSESSID`.

`SID` можно рассматривать как идентификационный номер в счете за электроэнергию: клиенту выдается номер, под которым данные клиента хранятся в энергетической компании. В дальнейшем клиенту больше не нужно сообщать компании все свои данные каждый раз, когда ему требуется уточнить сумму выставленного счета. `SID` автоматически создается и передается между клиентом и сервером каждый раз, когда пользователь щелкает по ссылке или на кнопке формы.

В момент инициализации PHP-сеанса сервер назначает данному сеансу идентификатор или `SID`. Любые переменные, зарегистрированные как переменные сеанса (как это делается, будет показано далее), хранятся на сервере в очень похожем на `cookie` файле. Именем данного файла, как правило, является значение `SID`. Все, что требуется сделать клиенту, чтобы получить доступ к своей информации на сервере, — включить `SID` в свой запрос к данному серверу. Для этого можно использовать скрытое поле формы, строку запроса или `cookie`-файл, если это определено `SID` в HTTP-запросе.

Сервер находит данные соответствующего сеанса и предоставляет к ним доступ любому исполняемому впоследствии сценарию.

Если поддержка cookie в браузере включена, то диспетчер сеансов автоматически отправляет клиенту cookie для данного SID-значения. Если клиент не может (или не хочет) использовать cookie, простейшим решением является добавление SID в каждую строку запроса, которая ссылается на данный Web-сайт.

С помощью PHP-сеансов все это происходит очень точно. Сеансы весьма способствуют быстрому созданию интерактивных сайтов, так как позволяют разработчику не беспокоиться о важных деталях реализации постоянства данных и полностью сосредоточиться на разработке сайта. Программисты, внимательно изучающие механизм обеспечения постоянства в PHP (так как этот механизм является частью самого PHP), гарантируют, что поддерживающий этот механизм код работает настолько хорошо, насколько это возможно. Кроме того, сеансы очень хорошо документированы в основной документации по PHP, доступной на официальном Web-сайте PHP (www.php.net/). Итак, сеансы просты в освоении, надежны и почти всегда доступны для использования. Теперь следует разобраться, как их использовать.

В большинстве случаев использование PHP-сеансов сводится к примерно следующей инструкции, переданной PHP: “Сделать переменные X, Y и Z постоянными для данного приложения”. Вся работа выполняется самим PHP. Программист должен позаботиться лишь о том, как дать PHP команду зарегистрировать переменную сеанса — т.е. сделать данную переменную постоянной — а затем как получить доступ к этой переменной.

Для регистрации постоянных переменных с помощью PHP-сеансов предназначена функция `session_register()`. При условии передачи этой функции имени переменной (без знака доллара) функция делает указанную переменную и ее содержимое постоянным в течение всего текущего сеанса. Если текущий сеанс еще не определен, то автоматически создается новый сеанс. Длительность сеанса (период времени, в течение которого сеанс сохраняется в PHP, даже если клиент не использует данный сайт) определяется настройками в файле `php.ini`. По умолчанию такой период равен 1440 секундам (24 минутам).

Например, если требуется сделать постоянными переменные `myvar1` и `myvar2`, то сценарий следует начать так:

```
<?php
session-register("myvar1");
session-register("myvar2");
?>
```

Приведенный выше код необходимо поместить в начало сценария. Cookie используются незаметно. Еще раз следует подчеркнуть, что все переменные сеанса необходимо зарегистрировать до того, как будет отправлен какой-либо HTML-код. Хорошей практикой является использование подобного самодостаточного фрагмента PHP-кода в самом начале сценария.

Как только переменная зарегистрирована, получить доступ к ее содержимому очень просто — следует только обратиться к переменной сеанса так же, как к любой другой глобальной переменной. Если для регистрации переменных `$myvar1` и `$myvar2` использовался приведенный выше код, то эти переменные можно использовать как любые другие переменные. Единственное отличие состоит в том, что эти переменные будут оставаться постоянными, пока продолжается данный сеанс, поэтому в следующий раз, когда внутри данного сеанса страница будет вызвана тем же пользователем, переменные будут содержать те же значения, которые они имели, когда страница была вызвана в предыдущий раз.

Чтобы использовать сеансы на Windows-сервере, вероятно, придется изменить параметр `session.save_path` в файле `php.ini`, так чтобы он указывал на корректный каталог в Windows (например, `D:\WinNT\Temp`, где `D` — имя логического диска).

Практика Счетчик посещений

Рассмотрим практический пример. Требуется подсчитать количество визитов пользователя на страницы Web-сайта с момента начала текущего сеанса. Эту задачу легко решить с помощью PHP-сеансов, сделав с помощью функции `session_register()` постоянными различные счетчики (по одному для каждой страницы сайта):

```
<?php
session_register('view1count');
session_register('view2count');
session_register('view3count');
session_register('view4count');
?>
<?php

//Остальная часть сценария иллюстрирует, как с помощью гиперссылок
//передавать PHP информацию, необходимую для доступа к данным сеанса,
//буквально - SID-идентификатор.
echo "<html><head><title>Счетчик посещений Web-страниц</title></head><body>";
if (isset($_GET['whichpage'])) {
    echo "<b>В данный момент Вы просматриваете страницу
        $_GET[whichpage].</b><br><br>\n";
    $_SESSION["view".$_GET['whichpage']."count"]++;
}

for ($i = 1; $i <= 4; $i++) {
    if (isset($_GET['whichpage']) == $i) {
        echo "<b><a href=\"sessions.php?\".session_id().\"&whichpage=$i\">
            Страница $i</a></b>";
    } else {
        echo "<a href=\"sessions.php?\".session_id().\"&whichpage=$i\">Страница $i</a>";
    }
}
if (!isset($_SESSION["view".$_i."count"])) $_SESSION["view".$_i."count"] = 0;
echo ", которую Вы смотрели ".$_SESSION["view".$_i."count"]." раз.<BR>\n";
}
echo "\n\n<br><br>\n\n";
echo "</body></html>";
?>
```

Сохраните данный файл как `sessions.php`. Откройте файл в браузере и смените страницу несколько раз. На рис. 3.21 показан примерный результат.

Теперь перейдите на другие страницы, а затем снова вернитесь к данному сценарию. Количество посещений страниц сохранилось. Данный сеанс закончится, только когда окно браузера будет закрыто.

Как это работает

Эта простая программа начинается с регистрации четырех переменных сеанса с помощью функции `session_register()`. Вызовы функции помещены в отдельный блок PHP-кода (обрамленный тегами `<?php` и `?>`), для того чтобы было понятно, что они должны располагаться до всего остального кода и не должны перемешиваться с HTML-заголовками:

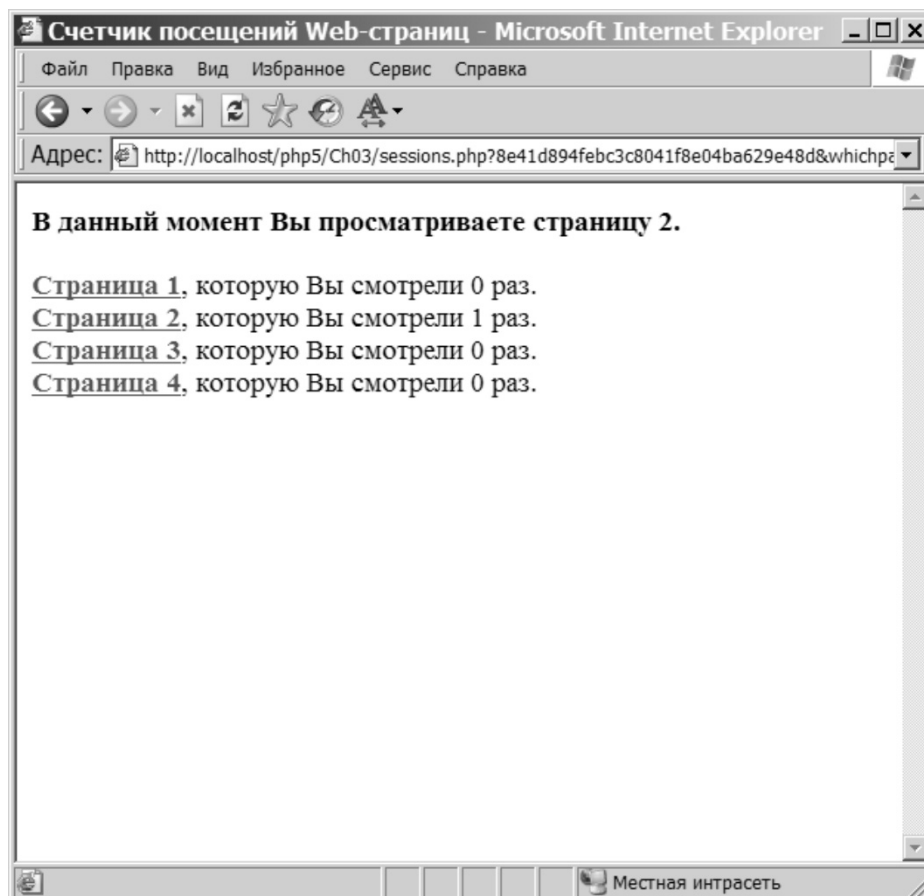


Рис. 3.21.

```
<?php
session_register('view1count');
session_register('view2count');
session_register('view3count');
session_register('view4count');
?>
```

Затем выводится HTML-заголовок и проверяется, определена ли переменная `$_GET['whichpage']` (обозначающая текущую страницу). Если эта переменная определена, то ее значение используется для отображения соответствующего сообщения и определения страницы, счетчик посещений которой необходимо инкрементировать:

```
if (isset($_GET['whichpage'])) {
    echo "<b>В данный момент Вы просматриваете страницу
$_GET[whichpage].</b><br><br>\n";
    $_SESSION["view".$_GET['whichpage']."count"]++;
}
```

В цикле `for` создается ссылка на каждую из четырех используемых страниц. Ссылка на каждую страницу отображается наряду с сообщением, которое говорит пользо-

вателю о том, сколько раз данная страница посещалась во время текущего сеанса. Ссылка на текущую страницу выделяется жирным шрифтом:

```
for ($i = 1; $i <= 4; $i++) {
    if (isset($_GET['whichpage']) == $i) {
        echo "<b><a href=\"sessions.php?\".session_id().\"&whichpage=$i\">Страница
$i</a></b>";
    } else {
        echo "<a href=\"sessions.php?\".session_id().\"&whichpage=$i\">Страница
$i</a>";
    }
}
if (!isset($_SESSION["view".$i."count"])) $_SESSION["view".$i."count"] = 0;
echo ", которую Вы смотрели ".$_SESSION["view".$i."count"]." раз.<BR>\n";
}
echo "\n\n<br><br>\n\n";
echo "</body></html>";
?>
```

В каждой ссылке указывается три элемента: текущий сценарий (`sessions.php`), текущий сеанс (идентифицируется с помощью функции `session_id()`) и страница, с которой связана ссылка (это то, для чего предназначена переменная `$_GET['whichpage']`). Это все. Весь необходимый код обработки сеансов содержится в первых четырех строках. Едва ли он мог бы быть еще проще.

Резюме

В данной главе были описаны многие предопределенные переменные, доступные PHP-программам в то время, когда между клиентом и Web-сервером происходит обмен данными (запросы и ответы). Также рассматривалось отображение и использование отдельных предопределенных переменных, таких как `$_SERVER`, `$_REQUEST`.

В главе были описаны основные элементы для построения интерактивных PHP-программ: строки запроса в гиперссылках (использование дескриптора `<a>`) и HTML-формы (дескрипторы `<input>`, `<textarea>` и `<select>`). Кроме того, обсуждалось различие между методами GET и POST, а также обстоятельства, в которых предпочтительно применять тот или иной метод.

Наконец, в главе было рассмотрено понятие состояния, а вернее отсутствие фиксации состояния в протоколе HTTP и способы преодоления связанных с этим ограничений: использование скрытых полей форм, строк запросов, cookie-файлов, сеансов и даже баз данных для хранения и передачи информации о состоянии между обращениями к страницам.

Упражнение

В PHP имеется очень полезная функция `isset()`, которая сообщает PHP-программе, установлена ли определенная переменная. Например, предположим, что существует страница с формой, содержащей submit-кнопки `login` и `logout`. С помощью функции `isset()` можно узнать, какая кнопка была нажата:

```
if (isset($login)) {
    //делаем что-то
} elseif (isset($logout)) {
    //делаем что-то другое
}
```

В этом упражнении требуется создать Web-страницу с формой, которая отправляет данные самой себе, и заставить PHP-программу сообщать об отправке формы, используя функцию `isset()`. Если форма не была отправлена, то программа должна отобразить форму (без сообщения), запрашивающую имя и фамилию пользователя. Иначе программа должна вместо формы отобразить сообщение (короткое предложение, например: “Ваше имя XX, а фамилия YY”).

Совет: чтобы определить, была ли отправлена форма, следует использовать скрытое поле, а для того чтобы заставить форму отправлять данные самой себе, можно использовать переменную `$PHP_SELF`.

4

Логические операторы, циклы и массивы

PHP, как любой хороший язык программирования, содержит структуры, которые позволяют в работающем приложении принимать решения в зависимости от текущих данных. Такие структуры являются основой программирования и разработки приложений, поскольку способность приложений отвечать на изменение входных данных — это то, что делает компьютеры столь эффективными. Примечательно, что такие структуры весьма просты и очевидны, а, кроме того, вместе с современными компьютерами, поражающими быстротой обработки данных, они открывают пользователям широчайший спектр возможностей.

Блоки кода, отвечающие за принятие решений, в программировании называются *структурами управляющей логики (control-flow structures)* или *структурами ветвления (branching structures)*. Они выполняют наборы базовых инструкций по условию в зависимости от значений или выражений, которые могут быть постоянными или изменяться в каждом шаге цикла. По форме все они представляют собой простые структуры, как, например, оператор `if`, но их можно комбинировать и реализовывать сложную логику принятия решений.

В данной главе рассматривается практическая постановка задач на простом языке и запись псевдокода, отражающего формулировку задачи. Кроме того, в главе описана Булева логика, несколько PHP-структур, оператор `break` и работа с массивами, в том числе и с многомерными массивами.

Проектирование логики PHP-программы

Существует много способов начать разработку приложения, но поэтапный подход помогает сэкономить время и уменьшает риск неудачи. Один из хороших методов заключается в том, чтобы начинать разработку с постановки задачи (формулировки проблемы). Затем следует написание псевдокода (инструкций компьютеру, подобных

реальному языку программирования, но написанных на обычном языке, например, на английском или на русском) на основе поставленной задачи и, наконец, написание реального РНР-кода на основе псевдокода. Псевдокод часто может послужить в качестве основы для комментариев (эта тема подробнее рассматривается в главе 5).

Постановка задачи

Предположим, например, что планируется создать приложение для клиентов ипотечного банка, и одной из функций этого приложения является ипотечный калькулятор. Скорее всего, выходными данными калькулятора будет сумма ежемесячной платы. Вычисление ежемесячной платы зависит от процентной ставки, срока и первоначального баланса по ссуде. Предположим, что приложение должно получать эти данные из некоторого источника (возможно, от пользователя или с какого-либо Web-сайта, предоставляющего информацию по текущим процентным ставкам).

Если не использовать готовую программу или Web-службу для генерации суммы взноса и даты оплаты, то приложение должно будет генерировать эти данные самостоятельно. Методологию обработки в данном случае можно выразить в логических понятиях, абсолютно не связанных с каким-либо языком программирования, например: “Принять в качестве входных данных баланс по ссуде, процентную ставку и срок предоставления ссуды. Предположить равные помесечные выплаты начиная с первого месяца и в течение всего срока ссуды. Вычислить каждую сумму и дату выплаты, а затем сгенерировать HTML-код для отображения этих данных пользователю. Повторять вычисления до тех пор, пока не будут рассчитаны все выплаты, а затем вернуть HTML-код пользователю в форме Web-страницы”.

Такая формулировка проблемы не является псевдокодом в чистом виде, однако она представляет собой хорошее начало для описания входных данных, обработки и выходных данных, необходимых для разрабатываемого приложения. Как только этот этап будет выполнен, можно будет легко написать псевдокод, исходя из поставленной задачи, а затем и код реальной программы.

Написание псевдокода

Что такое псевдокод? На самом деле нет строгого и точного определения. Здесь предлагается общее определение, которое облегчает понимание терминов: псевдокод представляет собой последовательность операторов на обычном языке, которые логически структурированы почти так же, как будет структурирован код в реальной программе.

Конечно, для эффективного написания псевдокода необходимо иметь представление о том, как работает язык программирования, который предполагается использовать, потому что псевдокод должен быть достаточно похожим на код реальной программы, чтобы его можно было очень быстро преобразовать непосредственно в программный код.

Для приведенной выше формулировки проблемы можно написать следующий псевдокод (с учетом того, что его впоследствии придется преобразовать в код РНР5):

```
//Проверить входящие значения баланса ссуды, процентной ставки и
//срока ссуды, используя проверочные функции
//Присвоить переменной для суммы платежа результат вычислений суммы
//платежей на основании входящих значений
//Создать массив дат выплаты, и последовательно выполнить столько
//вычислений даты платежа, сколько требуется выплат
//Создать строковую переменную, содержащую HTML-код для отображения
//сумм и дат платежей, и вставлять даты и суммы платежей на каждом
//шаге описанного выше цикла. Добавлять следующий HTML-код, дату
```

```
//платежа и сумму к строке на каждом шаге цикла
//Вычислить общее число платежей и общую сумму выплат, присваивая
//переменным данные значения на каждом шаге цикла и инкрементировать их
//Присвоить переменной строковое значение, содержащее HTML-код для
//отображения итоговых сумм
//Отправить пользователю HTML-код и сгенерированные значения.
```

В дополнение к демонстрации некоторых базовых методик написания псевдокода следует отметить методику повторяющихся вычислений (или другого вида обработки данных) в цикле. Многие используемые структуры языка программирования являются структурами управляющей логики, в которых решения принимаются в зависимости от значений, вычисленных или сгенерированных динамически (во время работы приложения). Нередко структуры принятия решений объединяются с циклами, чтобы можно было повторять вычисления снова и снова до тех пор, пока не будет найден окончательный ответ. Независимо от языка программирования почти все виды обработки данных основаны на одних и тех же структурах управляющей логики и циклах.

Булева логика

Весьма важной для управляющих структур является булева логика, так как булевы значения часто используются для выражения “условий”, определяющих то, какой набор инструкций будет обрабатываться дальше (по существу, принимаемое решение). Булева логика названа в честь Джорджа Буля (George Boole), создавшего булеву алгебру.

Булевы термы

Читатели, скорее всего, уже сталкивались с булевой логикой при работе с поисковыми машинами. Например, если слово “and” (“и”) между словами в поисковом запросе означает “получить документы, содержащие слово A и слово B”. “И” — булев терм, как и “или” (“or”) и “не” (“not”). Эти термы постоянно используются в повседневной жизни, и каждый из нас интуитивно понимает, что они означают. Существует также специальный терм — *xor* — он означает “если любое подвыражение (но не оба) истинно”. Существуют также специальные операторы, имитирующие термы “или” и “и” (|| и &&), но они имеют более высокий приоритет (в главе 2 тема приоритетов рассматривается подробнее).

Если рассматривать эти операторы в терминах результатов поиска, то можно отметить, что включение or в поисковый запрос дает большее количество возможных результатов, а xor, not и and последовательно все более сужают результаты поиска. Это можно проверить на любой поисковой машине (если она поддерживает булеву логику).

Конструируя блок логики обработки данных, можно использовать булевы термы для создания условий, при которых будет выполняться обработка данных. Например, в управляющей структуре if...then... else...end if за оператором if следует выражение, которое либо истинно, либо ложно (истина (true) и ложь (false) называются булевыми значениями). Если оно истинно, то выполняется код, следующий сразу за оператором if. Если выражение ложно, то данные операторы пропускаются и управление передается блоку else, а если данного блока нет, то пропускается вся управляющая структура.

Булевы значения

Булевыми значениями являются “истина” и “ложь”, представленные в PHP ключевыми словами TRUE и FALSE. И хотя булевы значения можно преобразовать, в большинстве случаев для выполнения соответствующих действий внутри логической

структуры этого делать не требуется. Следующие значения интерпретируются как булево значение FALSE:

- ☐ ноль как целое число (0);
- ☐ ноль как число с плавающей точкой (0.0);
- ☐ пустая строка ("");
- ☐ строковый ноль ("0");
- ☐ массив с нулевым количеством элементов;
- ☐ объект с нулевым количеством внутренних переменных;
- ☐ специальный тип NULL (включая любые незадаанные переменные).

Все остальные значения интерпретируются как TRUE. Следует помнить о том, что истинность или ложность выражения зависит от результата вычислений, а не от сравниваемых значений. Например, если значением А является 20, а В — 30, и есть выражение $A < B$ (А меньше В), то результатом будет TRUE. Для структур принятия решений возможно только два результата: истина или ложь. В таких структурах не допускается результат *может быть*.

Ранее рассматривались переменные, которые могли содержать числа или строки. Булевы значения содержатся в переменной третьего типа, которая может содержать одно из двух абсолютных значений: true или false. Любой из переменных можно присвоить одно из этих значений:

```
$Variable = true;
```

Однако если впоследствии отобразить значение данной переменной на экране, то будет выведено числовое значение:

```
1
```

Очевидно, что булевы значения могут иметь как числовую, так и литеральную форму. Само по себе это не вызывает особого интереса, но как только потребуется принять решение в зависимости от результата и определить, является ли заданное выражение истинным или ложным (либо, что то же самое, выражение равно 1 или 0), то выяснится, что булевы значения используются очень часто.

Использование булевых термов и значений

Чтобы определить истинность или ложность заданного выражения, можно определить его значение и сравнить это значение с некоторым другим значением. В результате вычисления выражения может получаться несколько значений (в форме: это *и* то, это *или* то, это *не* то), каждое из которых проверяется. Эти утверждения соединяются булевыми операторами and, or и xor. В PHP терм “не” представлен оператором !.

Когда в качестве условия используется два значения, первое из которых истинно, а второе ложно, и они соединены оператором and, то все условие будет ложным, и первый набор инструкций будет пропущен. Напротив, если оба выражения истинны и они соединены оператором and, то все условие будет истинным и будут выполняться операторы в первом блоке кода. Остальные группы операторов также приводят к тому, что решение принимается в зависимости от булева значения, которое возвращается после обработки всех операторов.

Проиллюстрировать работу данных условий может таблица результатов для простого блока управляющей логики if...then...else/elseif...end if. (Следует отметить, что then и end if в PHP представлены фигурными скобками; оператор if

подробнее рассматривается далее в настоящей главе.) Рассмотрим код (термы AND и OR записаны прописными буквами, чтобы они выделялись среди остального кода, хотя вообще они не чувствительны к регистру символов):

```
$my_var = 27; $my_var02 = 30;
if ($my_var == $my_var02) {
    //выполнить эти строки кода
} elseif ($my_var + 3 == $my_var02) {
    //выполнить эти строки кода
} elseif ($my_var + 3 == $my_var02 AND $my_var == $my_var02) {
    //выполнить эти строки кода
} elseif ($my_var == $my_var02 OR $my_var == $my_var02-3) {
    //выполнить эти строки кода
} elseif ($my_var == $my_var02 AND !$my_var == $my_var02-3) {
    //выполнить эти строки кода
}
```

В приведенной ниже таблице перечислены возможные результаты.

Условие	Результат
<code>\$my_var == \$my_var02</code>	Данное выражение будет истинным, только если обе переменные содержат в точности равные значения. Во всех остальных случаях результат будет ложным
<code>\$my_var + 3 == \$my_var02</code>	Сначала вычисляется выражение <code>\$my_var + 3</code> , затем полученный результат сравнивается со значением переменной <code>\$my_var02</code>
<code>\$my_var + 3 == \$my_var02 AND \$my_var == \$my_var02</code>	Результат определяется после того, как будут вычислены оба выражения. Результат будет истинным только в случае, если оба выражения истинны
<code>\$my_var == \$my_var02 OR \$my_var == \$my_var02 - 3</code>	Вычисляются оба выражения, но в данном случае результат будет истинным, если хотя бы одно из выражений истинно. Очевидно, это более свободное условие по сравнению с условием, где оба выражения должны быть истинными
<code>\$my_var == \$my_var02 - 3 AND !(\$my_var == \$my_var02)</code>	Чтобы соблюдался весь набор условий, первое выражение должно быть истинным, а второе ложным. При этом общий результат будет истинным. Следует отметить, что второе выражение должно быть ложным, так как это единственный случай, когда оператор <code>!</code> возвращает значение <code>true</code>

Очевидно, что можно выбрать условия, которые удовлетворяют почти любым обстоятельствам. Например, можно написать управляющую структуру, которая ищет только конкретное значение, введенное пользователем. Если написать условие так, что только одно значение активизирует код в структуре, то появится множество возможностей потерять это единственное значение и код никогда не будет выполнен (иногда это как раз то, что нужно). В других ситуациях необходимо убедиться, что код выполняется почти всегда, кроме очень специфических случаев. Можно написать условие, которое запускает код, если пользователь не ввел определенное значение:

```
if ($submitted_value != $my_internal_value) {
    //выполнить код
}
```

Использование структур управляющей логики (как самих по себе, так и вместе с циклами) открывает разработчику множество возможностей заставить программу

делать именно то, что требуется. Вместо того чтобы просто выполнять все строки программы одна за другой, структуры управляющей логики дают разработчику возможность выбирать для выполнения определенную строку кода и позволяют для принятия решений сравнивать различные переменные и значения.

Условные операторы или операторы ветвления

В простейшей своей форме условный блок кода означает либо “выполнить одну строку кода”, либо “не выполнять код вообще” в зависимости от того, выполняется ли заданное условие. Более сложный вариант подразумевает “выполнить данную строку кода” или “выполнить другую строку кода” в зависимости от того, какое условие выполняется. Это можно расширить до “выполнить один завершенный блок кода” или “выполнить другой завершенный блок кода”. Наконец, можно перечислить все возможные итоговые значения для определенного условия. Если результатом условия является итоговое значение номер 1, то следует выполнять блок кода 1; если результатом является итоговое значение 2, то будет выполнен блок 2; если результатом является итоговое значение 3 — блок 3, и т.д.

Пример ветвления

Все изложенное представляет собой, скорее, абстрактную дискуссию, поэтому полезно будет рассмотреть реальный пример из повседневной жизни. Посещение магазинов непосредственно не связано с программированием, но оно позволяет проиллюстрировать методику принятия решения, которая может быть реализована в программе.

Предположим, что требуется составить список продуктов, которые потребуются для приготовления бутерброда с сыром. Необходимо проверить наличие продуктов и сделать бутерброд на завтрак или в случае необходимости отправиться в магазин за недостающими продуктами. Ниже подробно описываются этапы данного процесса.

- ☐ Заглянуть в холодильник и проверить, есть ли там молоко, сыр и масло.
- ☐ Если какого-либо из перечисленных продуктов не хватает, то его следует добавить в список и перейти к следующему этапу; если все есть, то перейти к следующему этапу.
- ☐ Проверить наличие хлеба.
- ☐ Если хлеба нет, то его следует вписать в список и перейти к следующему этапу; если хлеб есть, то перейти к следующему этапу.
- ☐ Если уже есть все продукты, то приготовить бутерброд; если чего-либо не достает, то перейти к следующему этапу.
- ☐ Пойти в супермаркет.
- ☐ Купить недостающие продукты.
- ☐ Сделать бутерброд с сыром.

Данный пример представляет небольшой фрагмент процесса принятия логических решений, который мы используем практически каждый день. Часто мы проверяем определенные условия (есть ли все необходимые продукты), прежде чем предпринять

конкретные шаги (например, сделать бутерброд). В зависимости от этих условий (например, не хватает сыра) могут понадобиться дополнительные шаги (например, посещение магазина и покупка недостающих продуктов), позволяющие добиться определенных целей (таких как приготовление бутербродов с сыром).

Этот пример представляется типичным примером использования операторов принятия решений в РНР. При написании РНР-программ некоторые фрагменты кода пишутся для обработки специфических ситуаций, и если определенные ситуации не возникают, то нет необходимости выполнять относящийся к ним РНР-код.

По сути, процесс, описанный в примере, можно достаточно легко описать в РНР-программе. Фактически утверждения в примере достаточно близки к тому, что требуется, поэтому их можно использовать в качестве псевдокода (они хорошо подходят для большинства языков программирования, так как в большинстве языков подобный процесс принятия решений может быть представлен с помощью операторов `if`).

Например, можно написать на РНР следующий код (предположим, что определены переменные `$fridge` и `$bread_bin`, в которых хранятся строковые значения — названия продуктов):

```
if ($fridge == "молоко, сыр и масло") {
    if ($bread_bin == "хлеб") {
        make_sandwich();
    } else {
        go_to_store();
        make_sandwich();
    }
} else {
    go_to_store();
    make_sandwich();
}
```

Иначе говоря, если в холодильнике есть молоко, сыр и масло, то следует проверить наличие хлеба в хлебнице. Если оба эти условия выполняются, то можно остаться дома и сделать бутерброд. Однако если хлеба нет, то прежде чем сделать бутерброд, придется сходить в магазин и купить хлеба.

А если в холодильнике нет молока, сыра и масла, то все равно придется идти в магазин и покупать их независимо от того, есть хлеб или нет (подразумевается, что в данном случае также проверяется наличие хлеба, и если его нет, то он будет куплен одновременно с остальными продуктами в списке, поэтому снова проверять его наличие необязательно).

Оператор `if`

Оператор `if` вкратце уже рассматривался в главе 2, поэтому читатель должен представлять себе, как он работает. Абстрактно его действие заключается в том, что:

если (условие истинно) { то выполнить данную строку кода; }

Оператор `if` выполняет код (между фигурными скобками), только если условие истинно. Если же условие не соблюдается, то данный код будет проигнорирован и не будет выполняться РНР-процессором вообще.

Рассмотрим другой пример:

```
если (идет дождь) { взять зонт; }
выйти на улицу ;
```

Вторая строка выполняется независимо от истинности условия, но зонт необходимо взять, только если идет дождь.

Если требуется выполнить целый блок кода, то необходимо поместить данный код после условия и между фигурными скобками:

```
если (условие истинно) {  
    выполнить содержимое данных скобок;  
}
```

Чтобы расширить пример с зонтом, можно написать так:

```
если (погода дождливая) {  
    взять зонт;  
    одеть плащ;  
}  
выйти на улицу;
```

Снова оператор “выйти на улицу” выполняется в любом случае, а взять зонт и надеть плащ нужно, только если условие “погода дождливая” истинно.

Использование булевых операторов в структурах управляющей логики

Наиболее широко булевы термы и значения используются в структурах управляющей логики, таких как операторы `if..then..else` и `switch..case`. Данные структуры работают в зависимости от вычисления выражений, результатом которых являются булевы значения (истина или ложь). Если результат “истина”, то выполняется один набор операторов, если “ложь” — другой.

Операторы, результатом которых является одно из булевых значений, называются *булевыми операторами*. Они включают в себя операторы сравнения (такие как “больше чем” и “меньше чем”), операторы “равно” и “не равно” и т.д. Некоторые из таких операторов использовались в главе 3 для реализации логики принятия решения в приложении, обрабатывающем заявки на ссуды. В действительности всякий раз, когда требуется создать условие для принятия какого-либо решения, необходимо применять один из этих операторов. Существует четыре крупных категории таких операторов, рассмотрим примеры использования операторов в каждой из них.

Операторы > и <

Операторы “больше” и “меньше” справедливо считаются фундаментальными в начальной математике и в равной степени важны в программировании. В PHP они могут использоваться для сравнения двух констант, константы с переменной или двух переменных. В зависимости от результата сравнения можно выполнить определенный набор действий. В случае сравнения констант результат очевиден:

```
if (5 < 6) { echo "Пять меньше чем шесть"; }
```

Однако все равно стоит углубиться в детали происходящего. Условная часть оператора `if` — выражение, заключенное в круглые скобки. В результате вычисления этого выражения получается одно из двух булевых значений: истина или ложь. Фактически это единственные возможные результаты, поскольку условие либо соблюдается, либо нет. Поэтому в данном примере результатом данного выражения является истина (`true`). Оператор `if` выполняется, только если его условие истинно.

Предыдущий пример нельзя назвать полезным, потому что и без проверки известно, что пять меньше шести. Однако если сравнивать содержимое переменной с каким-либо числом, например, со счастливым числом, то ответ будет зависеть от значения переменной:

```
If ($lucky_number < 6) { echo "Наше счастливое число меньше шести"; }
```


Можно также сравнить две переменные:

```
If ($lucky_number < $lottery_number) { echo "Наше счастливое число слишком мало"; }
```

И конечно, результаты условия можно использовать не только для отображения какого-либо сообщения, но и для определения необходимого набора действий:

```
If ($lucky_number < $lottery_number){
    echo "Наше счастливое число слишком мало";
    $lucky_number = $lucky_number + 1;
}
```

Рассмотрим простой пример, в котором PHP-программа “загадывает” число между 1 и 10, а пользователю необходимо угадать это число. Чтобы заставить PHP “загадывать” число, необходимо использовать функцию-генератор случайных чисел, которая называется `rand()`. Принцип ее действия описан после примера.

Практика Использование операторов сравнения

1. Запустите HTML-редактор и введите следующий код:

```
<html>
<head><title></title></head>
<body>
<?php
if (isset($_POST['posted'])) {
    $number = rand(1,10);
    if ($_POST['guess'] > $number) {
        echo "<h3>Ваше число
            слишком большое</h3>";
        echo "<br>Загаданное число:
            $number, Вы проиграли, попробуйте еще раз <hr>";
    } else if ($_POST['guess'] < $number) {
        echo "<h3>Ваше число слишком маленькое</h3>";
        echo "<br>Загаданное число:
            $number, Вы проиграли, попробуйте еще раз <hr>";
    } else {
        echo "<br> Загаданное число:
            $number, Вы выиграли!<hr>";
    }
}
?>
<form method="POST" action="guessgame.php">
<input type="hidden" name="posted" value="true">
Угадайте число от 1 до 10
<input name="guess" type="text">
<br>
<br>
<input type="submit" value="Отправить">
</form>
</body>
</html>
```

2. Сохраните данный файл как `guessgame.php` и откройте его в браузере. Введите число в текстовое поле и нажмите кнопку **Отправить**. На рис. 4.1 показан примерный результат.

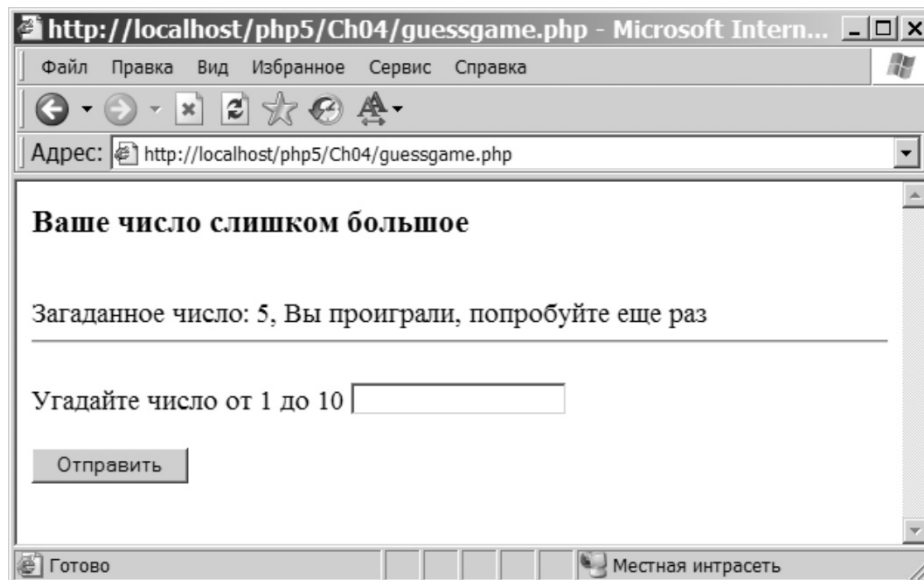


Рис. 4.1.

Как это работает

Данная программа слегка обманывает пользователя, потому что генерирует случайное число только после того, как пользователь отправил свой ответ. Это не влияет на результат, так как случайное число генерируется полностью независимо от переданного ответа пользователя. Ниже приведен код, генерирующий случайное число:

```
$number = rand(1,10);
```

Функция `rand()` чрезвычайно проста в использовании — ей просто необходимо передать минимальное и максимальное значение, разделенное запятой, и она сгенерирует случайное число между двумя заданными значениями. Результат сохраняется в переменной `$number`.

Когда страница открывается впервые, форма запрашивает у пользователя число и сохраняет ответ пользователя в текстовом поле, атрибут `name` которого равен `guess`.

```
Угадайте число от 1 до 10

```

Когда форма отправляется, данные в поле `guess` передаются обратно сценарию; устанавливается переменная `$_POST[posted]`, после чего можно использовать функцию `isset()`, чтобы определить, следует ли выполнять оставшуюся часть обработки (первый оператор `if`).

Затем используются операторы сравнения, позволяющие определить, что введенное пользователем число больше, меньше или равно случайному числу. Введенное пользователем число, которое хранится в переменной `$_POST[guess]`, сравнивается с числом, “загаданным” PHP-программой. Если значение этой переменной больше, чем значение `$number`, то выполняется код между двумя последующими фигурными скобками:

```
if ($_POST['guess'] > $number) {
    echo "<h3>Ваше число
    слишком большое</h3>";
}
```

```
echo "<br>Загаданное число:
$number, Вы проиграли, попробуйте еще раз <hr>";
```

Код после первых фигурных скобок информирует пользователя о том, что введенное число слишком велико, выводит “загаданное” число и приглашает сыграть снова.

Затем используется оператор `else if`. Чтобы он работал, необходимо закрыть предыдущий блок кода с помощью закрывающей фигурной скобки, написать `else if` и начать другой блок кода с открывающей фигурной скобки. Данный оператор работает как другой оператор `if` и проверяет, является ли введенное пользователем число слишком маленьким. Если это так, то выполняется код в следующих за данным оператором фигурных скобках:

```
} else if ($_POST['guess'] < $number) {
    echo "<h3>Ваше число слишком маленькое</h3>";
    echo "<br>Загаданное число:
$number, Вы проиграли, попробуйте еще раз <hr>";
```

На этот раз программа сообщает пользователю, что его ответ слишком мал, выводит “загаданное” число и приглашает сыграть снова.

Последний блок кода завершает РНР-сценарий. Если сценарий дошел до этой точки, то пользовательское число уже проверено и должно быть равно случайному числу, поэтому выполняется блок кода `else`. Программа сообщает пользователю о том, что он выиграл.

```
} else {
    echo "<br> Загаданное число:
$number, Вы выиграли!<hr>";
}
?>
```

Основной эффект от использования скрытого поля формы и функции `isset()` заключается в том, что программа будет работать, только если форма была отправлена.

Операторы `==` и `===`

Внимательные читатели уже заметили, что знак равенства в РНР используется в двух различных вариантах. Один знак равенства является оператором присваивания; двойной знак равенства — оператор равенства. Это различие очень важно. Рассмотрим код:

```
$lucky_number = 5;
$lucky_number = 7;
```

В приведенном выше коде переменной `$lucky_number` присваивается значение 5, а затем независимо от предыдущего значения (в данном случае 5) переменной присваивается новое значение (7). Иными словами, вторая строка аннулирует первую.

В следующем примере оператор равенства не оказывает влияния на значения переменных:

```
if ($lucky_number == 7) echo ("Ваше счастливое число: 7");
```

В других языках программирования, например, в Visual Basic, знак равенства (`=`) не является строго оператором присваивания и в зависимости от контекста может служить в качестве оператора сравнения. Программисты, которые привыкли к такому способу сравнения, часто ошибаются в РНР и присваивают переменной значение, имея в виду сравнение переменной с данным значением. Например:

```
$lucky_number = 5;
If ($lucky_number = 7) {
    echo "Ваше число $lucky_number";
```

```

} else {
    echo "Ваше число $lucky_number";
}

```

В этом примере программа выводит “Ваше число 7” (первый случай), а не “Ваше число 5” (второй случай), так как вместо того чтобы сравнить два значения знак равенства в операторе `if`, фактически переназначает (присваивает) значение 7 переменной `$lucky_number`. В PHP необходимо использовать одинарный знак равенства для присвоения значений и двойной знак равенства для сравнения, в противном случае программа может выдавать непредвиденные результаты. Об этом необходимо помнить.

Существует еще одна версия оператора равенства, она была введена в PHP 4.01. В данном операторе используется три знака равенства и он выдает `true`, только если значения равны и типы данных значений/переменных также равны:

```

if ($lucky_number === $random_number) {
    echo "Ваше счастливое число $random_number";
}

```

Тройной знак равенства очень полезен в ситуациях, когда необходимо проверить не только равенство значений, но и точное соответствие типов данных этих значений. Например, требуется узнать, равно ли данное значение именно `true` или `false`, а не одному из не булевых значений, которые можно преобразовать в `true` или `false` (см. раздел “Булевы значения” в данной главе выше).

Операторы `!=` и `<>`

Противоположностью оператору равенства `==` является оператор неравенства `!=`. Часто, читая чужой PHP-код, программисты пропускают `!` (восклицательный знак), однако он имеет большое значение, так как меняет значение выражения на противоположное, поэтому с ним следует быть особенно внимательным.

```

if ($lucky_number != 7) {
    echo ("Ваше счастливое число определено не 7");
}

```

Символы `!=` буквально означают “не равно”.

Существует другая форма записи для оператора неравенства — использование операторов “меньше” и “больше”. В операторе `if` такая нотация используется следующим образом:

```

if ($lucky_number <> 7) {
    echo ("Ваше счастливое число определено не 7");
}

```

Любой из этих операторов (`!=` или `<>`) возвращает `false` только в одном случае, когда значением переменной `$lucky_number` является число 7. Оператор `<>` означает “не”.

Воспользуемся простым сценарием из главы 3 для выбора ответа с помощью переключателей. На этот раз программа будет не только задавать вопрос, но и сообщит пользователю о том, верен его ответ или нет.

Практика Использование операторов равенства и неравенства

1. Откройте HTML-редактор и введите следующий код:

```

<html>
<head><title></title></head>
<body>
<?php

```

```

if (isset($_POST['posted'])) {
    if ($_POST['question1'] == "Лиссабон") {
        echo "Верно, $_POST[question1] -правильный ответ<hr>";
    }

    if ($_POST['question1'] != "Лиссабон") {
        echo "Неверно, $_POST[question1] - неправильный ответ<hr>";
    }
}
?>
<form method="POST" action="quiz.php">
<input type="hidden" name="posted" value="true">
Назовите столицу Португалии
<br>
<br>
<input name="question1" type="radio" value="Порту">
Порту
<br>
<input name="question1" type="radio" value="Лиссабон">
Лиссабон
<br>
<input name="question1" type="radio" value="Мадрид">
Мадрид
<br>
<br>
<input type="submit" value="Отправить ответ">
</form>
</body>
</html>

```

2. Сохраните данный файл как `quiz.php` и закройте его.
3. Откройте созданный файл в браузере, выберите ответ Мадрид и нажмите кнопку Отправить ответ. На рис. 4.2 показан результат.

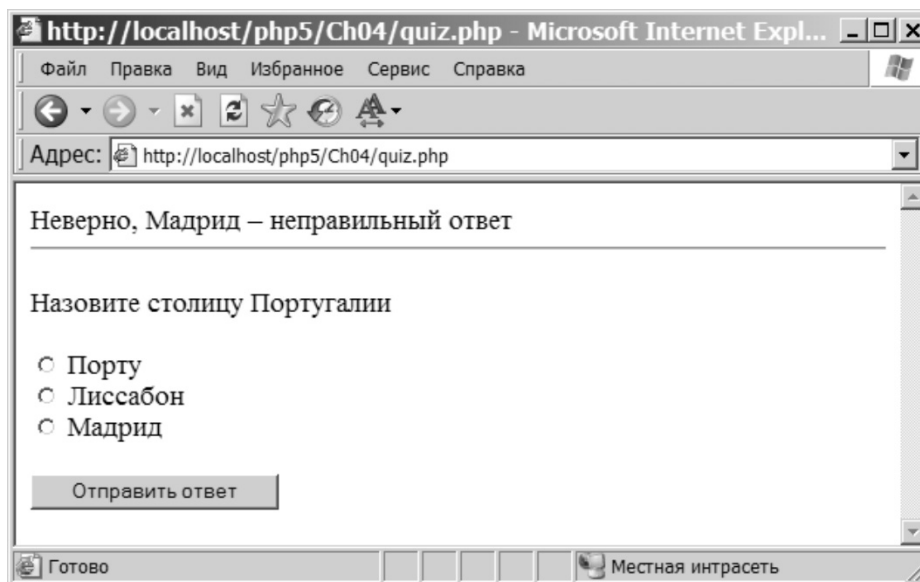


Рис. 4.2.

4. Попробуйте выбрать другие ответы и посмотрите, какой будет результат.

Как это работает

Операторы `if` проверяют, имеет ли переменная `$_POST[question1]` значение Лиссабон. Если это так, то пользователю отправляется положительный ответ; в противном случае возвращается отрицательный ответ.

Логические операторы (AND, OR, !)

Логические операторы уже обсуждались ранее в данной главе, теперь их можно применить на практике. Чтобы понять их работу в программах, можно провести аналогию с их действием в обычном языке, поскольку, например, их использование в английском языке почти аналогично использованию в PHP. Например, если сегодня суббота и погода солнечная, то можно идти на пляж. Подобным образом это реализуется и в PHP:

```
if ($day == "Суббота" AND $weather == "солнце светит") {  
    echo ("Значит можно идти на пляж");  
}
```

Оператор `AND` также можно написать с помощью двойного амперсанда (`&&`):

```
if ($day == "Суббота" && $weather == "солнце светит") {  
    echo ("Значит можно идти на пляж");  
}
```

Операторы `OR` и `!` используются также просто. Можно перефразировать данный пример и использовать в нем оператор `OR` для представления противоположной ситуации.

```
if ($day == "Понедельник" OR $weather == "идет дождь") {  
    echo ("значит сегодня пляж отменяется");  
}
```

Если сегодня понедельник или идет дождь, то идти на пляж нельзя. Данный оператор также может быть представлен с помощью двойных вертикальных черт (двойного символа конвейера) `||`, поэтому предыдущий пример кода можно записать так:

```
if ($day == "Понедельник" || $weather == "идет дождь") {  
    echo ("значит сегодня пляж отменяется");  
}
```

Следует отметить, что (хотя это не всегда влияет на работу кода) операторы `&&` и `AND` имеют несколько различный приоритет. То же можно сказать и об операторах `||` и `OR`. Операторы `&&` и `||` имеют приоритет выше, чем у их текстовых аналогов.

Несмотря на то что оператор `NOT` существует, использовать слово “not” нельзя. Фактически в качестве оператора `NOT` используется восклицательный знак. Если восклицательный знак предшествует круглым скобкам, то он изменяет результат в них на противоположный, поэтому если условие первоначально возвращает `true`, то при использовании восклицательного знака оно будет возвращать `false` и наоборот. Например, если текущий день не суббота, то идти на пляж нельзя:

```
if !($day == "Суббота") {  
    echo ("значит сегодня пляж отменяется");  
}
```

Может показаться, что восклицательный знак работает так же, как и описанный ранее оператор неравенства, однако это не так. Оператор неравенства применяется только к выражению, частью которого он является, тогда как восклицательный знак перед круглыми скобками в операторе `if` применяется ко всему условию (к выражению в скобках). Фактически можно обратить результат условия, в котором даже нет операторов. Можно просто поместить переменную в условную часть оператора `if`, и если данная переменная не имеет значения, то в следующем примере условие будет

возвращать false, но если перед условием будет стоять восклицательный знак, то условие будет возвращать true. Описать данный код можно так: “Если переменная \$answer не содержит значения, то перейти к оператору echo”.

```
if !($answer) {
    echo ("Ответа нет");
}
```

Данный оператор распечатывает сообщение, только если переменной \$answer не присвоено значение или если ей присвоено нулевое значение (что в PHP эквивалентно отсутствию значения). Это происходит из-за того, что оператор ! инвертирует значение \$answer, поэтому если \$answer возвращает false, то !(\$answer) возвращает true, и выполняется оператор в фигурных скобках. Это важный момент, потому что часто условие, используемое в операторе if, просто является результатом вычисления какого-либо выражения или результатом функции, и если выражение возвращает true, то оператор if продолжает обработку. В условной части операторов вообще может не быть.

Рассмотрим практический пример использования логических операторов.

Практика Использование логических операторов

Следующую программу можно использовать в компании, предоставляющей автомобили напрокат, чтобы определять, можно ли доверить тому или иному клиенту автомобиль. Предполагаемый водитель должен иметь водительские права и быть старше 21 года. Программа проверяет эти и другие данные.

1. Откройте HTML-редактор и введите следующий код:

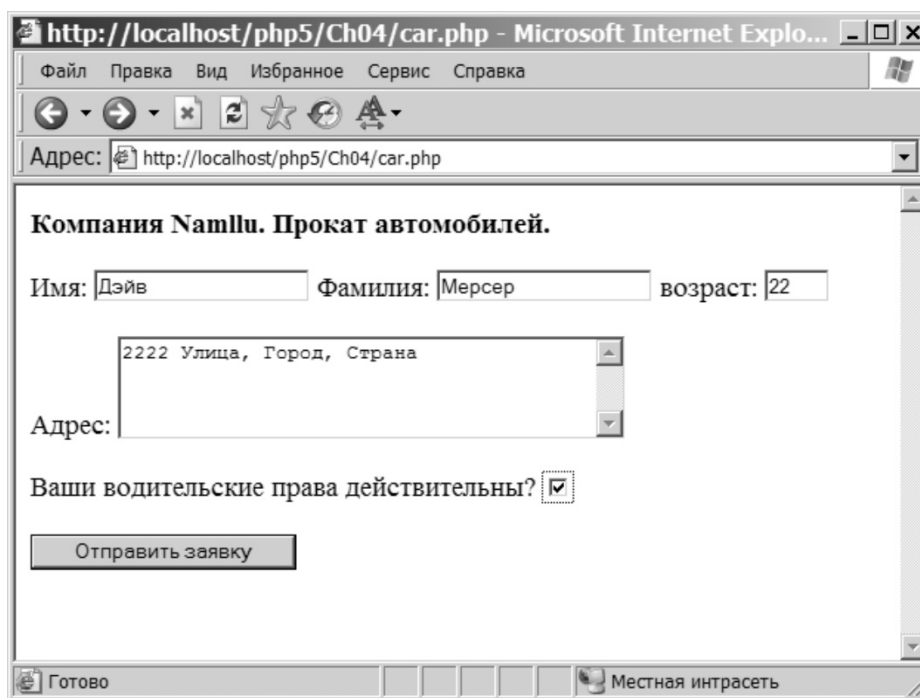
```
<html>
<head><title></title></head>
<body>
<b>Компания Namllu. Прокат автомобилей</b>
<?php
if (isset($_POST['posted'])) {

    if ($_POST['age'] > 20 and $_POST['license'] == "on") {
        echo ("Вам можно предоставить машину напрокат.<br>");
    }

    if ($_POST['age'] < 21 or $_POST['license'] == "") {
        echo ("К сожалению, мы не можем предоставить Вам машину напрокат.<br>");
    }
} else {
?>
<form method="post" action="car.php">
<input type="hidden" name="posted" value="true">
Имя:
<input name="first_name" type="text">
Фамилия:
<input name="last_name" type="text">
возраст:
<input name="age" type="text"size="3">
<br>
<br>
Адрес:
<textarea name="address" rows=4 cols=40>
</textarea>
<br>
<br>
```

```
Ваши водительские права действительны?  
<input name="license" type="checkbox">  
<br>  
<br>  
<input type="submit" value="Отправить заявку">  
</form>  
</body>  
</html>  
<?php  
>
```

2. Сохраните данный файл как `car.php` и закройте его.
3. Откройте созданный файл в браузере и введите необходимые сведения, как показано на рис. 4.3.



http://localhost/php5/Ch04/car.php - Microsoft Internet Explo...

Файл Правка Вид Избранное Сервис Справка

Адрес: http://localhost/php5/Ch04/car.php

Компания Namllu. Прокат автомобилей.

Имя: Фамилия: возраст:

Адрес:

Ваши водительские права действительны? ☒

Готово Местная интрасеть

Рис. 4.3.

4. Нажмите кнопку **Отправить заявку**. Примерный результат показан на рис. 4.4.

Как это работает

Чтобы собрать дополнительную информацию, используется более сложная по сравнению с уже рассмотренными HTML-форма (ее можно было бы сделать лучше с дизайнерской точки зрения, но в данном случае достаточно того, что она просто работает). В коде используется оператор `else`, позволяющий отображать форму, только если она не была отправлена пользователем, потому что как только пользователь отправил заявку, нет необходимости отображать форму снова (не забудьте написать `<? ?>` после завершения кода HTML-формы).

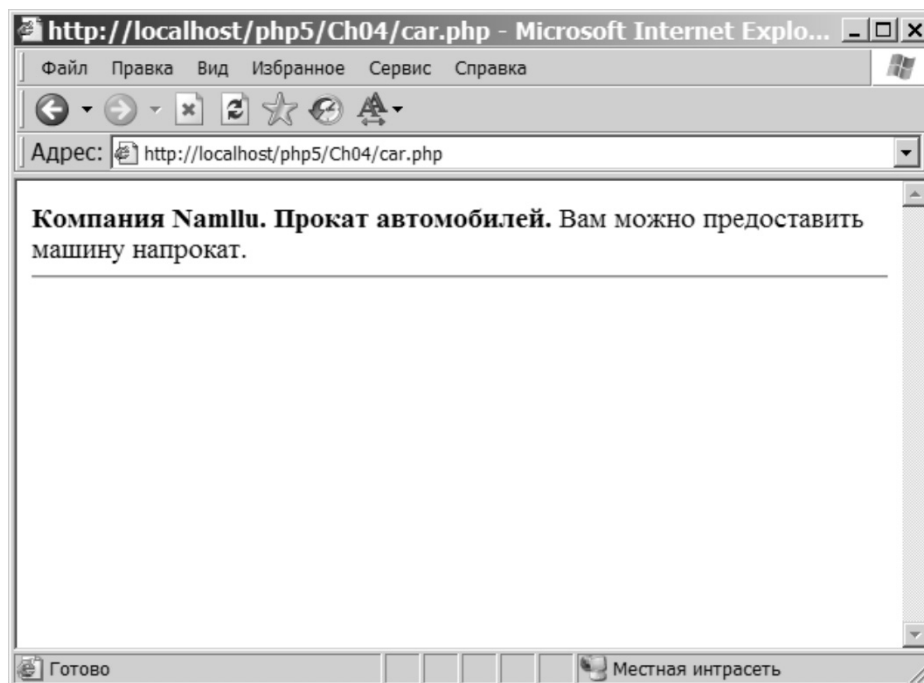


Рис. 4.4.

Правильная работа PHP-кода зависит от выбора данных из HTML-полей (текстового поля `age` и флажка `license`). Данные переменные принимаются после отправки формы как `$_POST[age]` и `$_POST[license]`. Приведенный ниже HTML-код реализует форму и кнопку отправки данных.

```
<form method="post" action="car.php">
<input type="hidden" name="posted" value="true">
Имя:
<input name="first_name" type="text">
Фамилия:
<input name="last_name" type="text">
возраст:
<input name="age" type="text"size="3">
<br>
<br>
Адрес:
<textarea name="address" rows=4 cols=40>
</textarea>
<br>
<br>
Ваши водительские права действительны?
<input name="license" type="checkbox">
<br>
<br>
<input type="submit" value="Отправить заявку">
</form>
```

Текстовое поле имеет атрибут `name`, равный `age`, поэтому для хранения возраста пользователя создается переменная `$_POST[age]`. Флажок (`license`) либо установлен, либо нет, поэтому переменная `$_POST[license]` может либо содержать значение `on`, либо не содержать значения вообще.

Значение `on` фактически зависит от браузера, но браузеры Internet Explorer, Netscape Navigator и Opera используют его без каких-либо проблем. Если используется другой браузер и данный пример не работает, то следует применить оператор `echo()`, чтобы опросить переменную `$license` и соответствующим образом модернизировать код (подробнее использование оператора `echo()` в целях отладки описано в главе 5).

PHP-сценарий использует обе эти переменные. Первый оператор `if` в файле `car.php` проверяет указанный пользователем возраст, и если возраст больше 20 лет и пользователь имеет водительские права, то можно принять заявку на прокат машины:

```
if ($_POST['age'] > 20 and $_POST['license'] == "on") {  
    echo ("Вам можно предоставить машину напрокат.<hr>");  
}
```

Второй блок проверяет обратное условие: если пользователь моложе 21 года или он не имеет водительских прав, то ему будет отказано в аренде автомобиля:

```
if ($_POST['age'] < 21 or $_POST['license'] == "") {  
    echo ("К сожалению, мы не можем предоставить Вам машину напрокат.<hr>");  
}
```

Это весь сценарий.

Есть еще одно непредвиденное обстоятельство: что произойдет, если пользователь введет возраст между 20 и 21 годами, например, 20.5? Это маловероятно, но сценарий должен обрабатывать подобные ситуации. В таком случае любое условие по возрасту будет приемлемым и получится два ответа. В ситуациях подобных данной, возможно, наилучшим решением будет использование операторов `>=` или `<=` (“больше или равно” либо “меньше или равно”). Для того чтобы получить верный результат, можно использовать следующий код:

```
if ($_POST['age'] >= 20 and $_POST['license'] == "on") {  
    echo ("Вам можно предоставить машину напрокат.<hr>");  
} else {  
    echo ("К сожалению, мы не можем предоставить Вам машину напрокат.<hr>");  
}
```

Операторы switch

Иногда при использовании оператора `if` приходится проверять множество условий. В таких случаях можно писать длинные `if`-операторы с блоками `else if`, но часто намного проще использовать оператор `switch`. Данный оператор переключает управление с одного блока на другой в зависимости от определенного входного значения.

Ниже приведен пример использования оператора `switch`:

```
switch ($grade) {  
    case $grade>90:  
        echo ("Оценка A.");  
        break;  
    case $grade>80:  
        echo ("Оценка B.");  
        break;  
    case $grade>70:  
        echo ("Оценка C.");  
        break;  
    case $grade>50:  
        echo ("Оценка D.");  
        break;  
    case $grade<50:  
        echo ("Оценка F. ");  
        break;  
    default:  
        echo ("Не сдано");  
}
```

Вместо операторов `if` и `else if` в данном случае используется только оператор `case` (принимаящий переменную `$grade`), за которым следуют блоки кода. В каждом из этих блоков проверяется значение переменной `$grade`, а затем выполняются некоторые действия. В каждом случае РНР-программа выполняет другие действия. Хорошей практикой является использование случая по умолчанию, так как в нем обрабатываются все не описанные случаи (если случай по умолчанию не используется, то может появиться пустая страница без каких-либо данных и без сообщений об ошибках).

В операторе `switch` используется команда `break`. Когда РНР встречает данную команду, то останавливает текущее действие, выходит из `switch`-структуры и переходит к выполнению программного потока сразу после закрывающей скобки. Совпадение с другими критериями при этом не проверяется, даже если оценка, равная 80%, соответствует всем этим критериям. Это полезно, потому что дает возможность обойтись без многочисленных мелких блоков для обработки всех вероятных ситуаций. Если требуется, чтобы проверялись все критерии, то можно просто не использовать ключевое слово `break`, хотя следует отметить, что `break` работает только совместно с оператором `switch` и не работает с `if`. Также следует заметить, что можно запутаться, если поместить `break` в оператор `if`, который находится внутри оператора `switch` или цикла `for`, так как оператор `switch` и цикл `for` в отличие от `if` реагируют на `break`.

Если опустить ключевое слово `break`, то все выражения будут равны `true`, и оценки А, В, С, D и F получатся с результатом 80.

```
switch ($State) {    case "IL":
    echo ("Illinois");
    break;
    case "GA":
    echo ("Georgia");
    break;
    default:
    echo ("California");
    break;
}
```

Если сразу за ключевым словом `case` просто указать значение (в данном примере IL), то РНР автоматически сравнит с данным значением переменную в круглых скобках оператора `switch` независимо от того, числовая она или текстовая.

После ключевого слова `case` используется двоеточие, а не точка с запятой.

Фактически оба знака работают одинаково (двоеточие используется в документации на официальном сайте РНР), поэтому можно использовать то, что удобнее.

Следует создавать `case`-блоки только для значений, которые ожидаются и с которыми программа способна работать (даже если оператор `switch` используется только для прерывания обработки). Например, можно оставить пустым `case`-блок внутри оператора `switch` и включить в него только ключевое слово `break`. После этого, когда РНР встретит ключевое слово `case`, оператор `switch` будет прерван и обработка внутри него будет прекращена. Это происходит в последующем примере кода, когда пользователь вводит для переменной `$State` значение НН. Тогда активизируется блок для НН и обработка завершается. Когда же вводится неизвестное значение, оно активизирует случай по умолчанию и выводит ответ California:

```
switch ($State) {    case "HH":
    break;
    case "IL":
    echo ("Illinois");
    break;
    case "GA":
```

```
        echo ("Georgia");  
        break;  
    default:  
        echo ("California");  
        break;  
    }
```

Практика Использование операторов switch

В следующем примере создается форма, позволяющая пользователю выбирать пункт назначения и класса гостиницы для оформления путевки на отдых. Эти классы гостиниц используются впоследствии для расчета цены для выбранного пользователем вида отдыха.

1. Откройте текстовый редактор и введите следующий код:

```
<html>  
<head><title></title></head>  
<body>  
<b>Бронирование гостиниц Namllu</b>  
<br>  
<br>  
<?php  
if (isset($_POST['posted'])) {  
  
    $price = 500;  
    $starmodifier = 1;  
    $citymodifier = 1;  
    $destination = $_POST['destination'];  
    $destgrade = $_POST['destination'] . $_POST['grade'];  
    switch($destgrade) {  
        case "Barcelonathree":  
            $citymodifier = 2;  
            $price = $price * $citymodifier;  
            echo "Недельная стоимость проживания в $destination - $price";  
            break;  
        case "Barcelonafour":  
            $citymodifier = 2;  
            $starmodifier = 2;  
            $price = $price * $citymodifier * $starmodifier;  
            echo "Недельная стоимость проживания в $destination - $price";  
            break;  
        case "Viennathree":  
            $citymodifier = 3.5;  
            $price = $price * $citymodifier;  
            echo "Недельная стоимость проживания в $destination - $price";  
            break;  
        case "Viennafour":  
            $citymodifier = 3.5;  
            $starmodifier = 2;  
            $price = $price * $citymodifier * $starmodifier;  
            echo "Недельная стоимость проживания в $destination - $price";  
            break;  
        case "Praguethree":  
            $price = $price * $citymodifier;  
            echo "Недельная стоимость проживания в $destination - $price";  
            break;  
        case "Praguefour":  
            $starmodifier = 2;  
            $price = $price * $citymodifier * $starmodifier;  
            echo "Недельная стоимость проживания в $destination - $price";  
            break;  
    }
```

```

        default:
            echo ("Выберите снова");
            break;
    }
}
?>
<form method="POST" action="holiday.php">
<input type="hidden" name="posted" value="true">
Где Вы хотели бы провести отпуск?
<br>
<br>
<input name="destination" type="radio" value="Prague">
Прага
<br>
<input name="destination" type="radio" value="Barcelona">
Барселона
<br>
<input name="destination" type="radio" value="Vienna">
Вена
<br>
<br>
В гостинице какого класса Вы хотели бы остановиться?
<br>
<br>
<input name="grade" type="radio" value="three">
Три звездочки
<br>
<input name="grade" type="radio" value="four">
Четыре звездочки
<br>
<br>
<input type="submit" value="Заказать">
</form>
</body>
</html>

```

2. Сохраните данный файл как `holiday.php` и закройте его.
3. Откройте файл в браузере, введите в форму какие-либо сведения и нажмите кнопку **Заказать**. Примерный результат показан на рис. 4.5.

Как это работает

Два замечания по поводу оператора `switch`: он легко читается и в нем используется только несколько строк кода. Первое замечание субъективно. Рассмотрим второе: данную программу можно было бы написать как последовательность операторов `if` и в ней пришлось бы использовать большое число строк PHP-кода между тегами `<?php` и `?>`, а в файле `holiday.php` используется всего 41 строка.

Рассмотрим работу программы в целом. При открытии страницы пользователь видит HTML-форму и понимает, что ему требуется сделать какой-либо выбор. Как только пользователь отправляет форму, PHP-код активизируется, так как переменная `$_POST[posted]` установлена, а это определяется в условной части первого (и только) оператора `if` с помощью функции `isset()`. Затем код внутри данного оператора `if` начинает работать и присваивает значения нескольким переменным:

```

$price = 500;
$starmodifier = 1;
$citymodifier = 1;

$destination = $_POST['destination'];

```

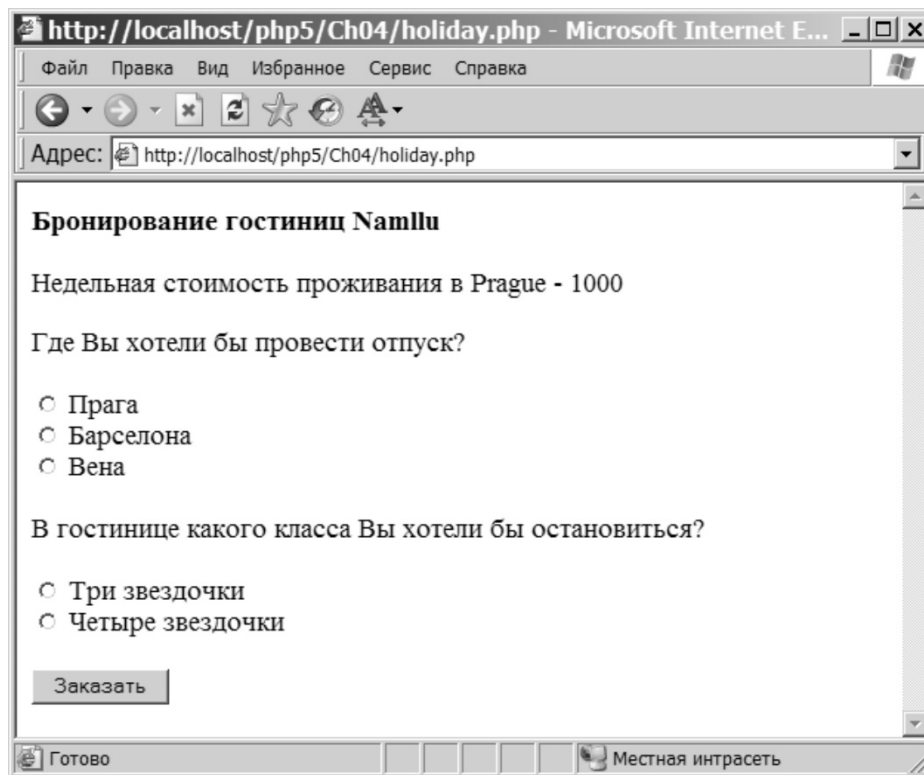


Рис. 4.5.

В следующей строке вводится новая переменная, \$destgrade, значение которой представляет собой конкатенацию \$_POST['destination'] и \$_POST['grade'].

```
$destgrade = $_POST['destination'] . $_POST['grade'];
```

Если пользователь выбирает Барселону и четырехзвездочную гостиницу, то значением переменной \$destgrade будет строка "Barcelonafour". Путем однократного введения условия в оператор switch частично сокращается количество строк кода (по сравнению с использованием нескольких операторов if):

```
switch($destgrade) {
```

Необходимо обработать все семь возможных результатов. (Три варианта гостиницы, умноженные на два класса, плюс нестандартный выбор.) Шестью возможными вариантами являются Барселона и трехзвездочная гостиница, Барселона и четырехзвездочная гостиница, Прага и трехзвездочная гостиница, Прага и четырехзвездочная гостиница, Вена и трехзвездочная гостиница, Вена и четырехзвездочная гостиница. Таким образом, возможными значениями переменной \$destgrade могут быть только строки Barcelonathree, Barcelonafour, Praguethree, Praguefour, Viennathree и Viennafour, а также нестандартный выбор пользователя.

Так как во всех случаях выполняются одинаковые действия, рассмотрим только один пример:

```
case "Barcelonathree":
    $citymodifier = 2;
```

```
$price = $price * $citymodifier;
echo "Стоимость недельного проживания в $destination - $price";
break;
```

В случае `Barcelonathree` переменной `$citymodifier` присваивается значение 2, умноженное на цену. Затем отображается пункт назначения и цена, и оператор `break` переводит управление в конец программы. Если значение `$destgrade` не совпадает ни с одним корректным вариантом, значит что-то было сделано не так, и пользователю отображается сообщение о необходимости вернуться и ответить на вопросы снова.

Хорошей практикой является добавление в конец `case`-блока ключевого слова `break`, даже если этот блок не выполняет никаких действий:

```
default:
    echo ("Выберите снова");
    break; }
```

Это означает, что код, скорее всего, не будет генерировать ошибок, если когда-нибудь после оператора `else` будет добавлен новый `case`-блок.

Циклы и массивы

В PHP имеется три вида циклов: `while`, `do while` и `for`. Все они рассматриваются в данной главе, после чего описывается связанный с циклами элемент языка — массивы, которые уже рассматривались вкратце в начале данной книги. Массив представляет собой набор индексированных переменных, который, особенно совместно с циклом, может оказаться весьма полезным. Циклы обработки массивов позволяют создавать сотни или даже тысячи переменных, используя только три или четыре строки кода.

Циклы

В данной главе уже рассматривался оператор ветвления (условный оператор). Он позволяет вводить в PHP-код логику принятия решений. Циклы похожи на операторы ветвления тем, что выполнение следующей строки кода зависит от того, истинно или ложно условие.

Однако циклы отличаются от условных операторов, потому что содержимое цикла может выполняться снова и снова. Проверяется условие, и если оно удовлетворено, то выполняется код цикла. Затем условие проверяется снова; если оно все еще корректно, то код цикла выполняется снова, и этот процесс может повторяться многократно. Каждый проход по циклу называется *итерацией* (*iteration*).

Как будет показано далее, все типы циклов приспособлены для различных ситуаций.

Цикл `while`

Цикл `while` является простейшим из циклов и отчасти похож на оператор `if`, потому что проверяет результат выполнения некоторого условия. В зависимости от того, истинно или ложно данное условие, выполняется раздел кода, помещенный между фигурными скобками после условного оператора:

```
while (условие истинно){
    выполнить содержимое данного блока;
}
```

После того как команды цикла выполнены, условие вверху цикла проверяется снова, и, возможно, код будет выполняться опять и т.д. Если при проверке условия выяс-

няется, что оно ложно, то код в фигурных скобках игнорируется и PHP продолжает работу с первой строки ниже скобок. Рассмотрим пример псевдокода:

```
while (луна полная){
    койот будет выть;
}
```

Таким образом, если луна не полная, то койоты не будут выть, а во время полнолуния будут. Рассмотрим еще один пример. Предположим, что необходимо проинформировать пользователя, выполняющего покупки на сайте, о том, что его кредит исчерпан. Чтобы отобразить пользователю сообщение, когда его счет превысит лимит кредита, можно использовать такой фрагмент PHP-кода:

```
while ($shopping_total > $credit_limit){
    echo ("Вы превысили лимит своего кредита,
           поэтому последний объект будет удален из Вашей корзины");
    $last_purchase_but_one = $shopping_total - $last_purchase;
    $last_purchase = $last_purchase_but_one;
}
```

Если пользователь превысит свой кредитный лимит `$credit_limit`, то его последняя покупка будет отменена и ее стоимость (`$last_purchase`) будет вычтена из общего счета (`$shopping_total`). Затем следует изменить значение `$last_purchase`, так чтобы оно было равно общей стоимости всех покупок, кроме последней, `$last_purchase_but_one`. Это даст возможность проходить по циклу, удаляя из списка приобретенных товаров один объект за раз до тех пор, пока `$shopping_total` не будет меньше, чем `$credit_limit`.

Нетрудно заметить, что если `$credit_limit` имеет негативное значение, то данный цикл может продолжаться бесконечно. Бесконечные циклы могут возникать, когда используется условие, которое может быть всегда верным. Это означает, что программа не закончится, поскольку проверка условия никогда не возвратит `false`. При этом ошибка не генерируется, просто программа продолжает обрабатывать содержимое цикла снова и снова. Об этом следует помнить при написании программ, использующих циклы. Подробнее бесконечные циклы описываются в настоящей главе далее.

Чтобы посмотреть на цикл `while` в действии, можно расширить приложение, обрабатывающее заявку на ссуду, которое было написано в главе 3. Программа запрашивала у пользователя сумму ссуды, а затем подтверждала принятие заявки или отклоняла ее. Теперь представим, что утверждено несколько возможных вариантов ссуды, из которых пользователь может выбирать. Существует три различных варианта ссуды, в каждом из которых предлагается своя сумма и своя месячная процентная ставка. Новая программа будет сообщать пользователю о том, как долго ему придется возвращать ссуду.

Конечно, чтобы это сделать, от пользователя необходимо получить еще один блок информации: сумму, которую пользователь готов выплачивать ежемесячно. Кроме того, пользователь должен ежемесячно выплачивать проценты по ссуде. Иначе говоря, если клиент взял займы 1000 долларов под 5% в месяц и возвращает 100 долларов в месяц, то в это число также включается 50 долларов по процентам за первый месяц. Таким образом, в действительности в первый месяц клиент возвращает 50 долларов невыплаченной ссуды. Во втором месяце остаток ссуды будет равен 950 долларов, и если вернуть еще 100 долларов, то 5% от этой суммы составят 47,50 долларов. Очевидно, что вычисления довольно скоро могут стать сложными, хотя формула каждый месяц будет оставаться постоянной:

Платеж = Месячный платеж – Проценты
Долг = Долг – Платеж

Все что нужно делать — снова и снова вычислять результат по данной формуле, пока долг не будет равен нулю, а затем посчитать, сколько месячных платежей для этого требуется. Чтобы это сделать, можно заставить PHP использовать цикл `while`.

Практика Использование цикла `while`

1. Откройте HTML-редактор и введите следующий код:

```
<html>
<head><title></title></head>
<body>
<b>Заявка на получение ссуды. Кредитный банк Namllu</b>

<?php
    if (isset($_POST['posted'])) {
        $duration = 0;
        switch ($_POST['loan']) {
            case "1000";
                $interest = 5;
                break;
            case "5000";
                $interest = 6.5;
                break;
            case "10000";
                $interest = 8;
                break;
            default:
                echo "Вы не выбрали вариант ссуды<br>";
                exit;
        }
        while ($_POST['loan'] > 0)
        {
            $duration = $duration + 1;
            $monthly = $_POST['month'] - ($_POST['loan']*$interest/100);
            if ($monthly <= 0)
            {
                echo "Чтобы погасить ссуду, требуются более крупные
                    ежемесячные платежи<br>";
                exit;
            }
            $_POST['loan'] = $_POST['loan'] - $monthly;
        }
        echo "Для погашения ссуды при процентной ставке $interest процентов
            понадобится $duration месяцев.<br>";
    }
?>
<form method="POST" action="loan.php">
<input type="hidden" name="posted" value="true">
<br>
Выберите необходимую сумму ссуды<br><br>
<input name="loan" type="radio" value="1000">1000 долларов под 5,0% в месяц
<br>
<input name="loan" type="radio" value="5000">5000 долларов под 6,5% в месяц
<br>
<input name="loan" type="radio" value="10000">10000 долларов под 8,0% в месяц
<br>
<br>
Введите сумму ежемесячного платежа
<input name="month" type="text" size="5">
<br>
```

```
<br>
<input type="submit" value="Рассчитать">
</form>
</body>
</html>
```

2. Сохраните данный файл как `loan.php` и закройте его.
3. Откройте страницу в браузере. Выберите вариант 1000 долларов, введите 100 как месячный платеж и нажмите кнопку **Рассчитать**. Результат показан на рис. 4.6.

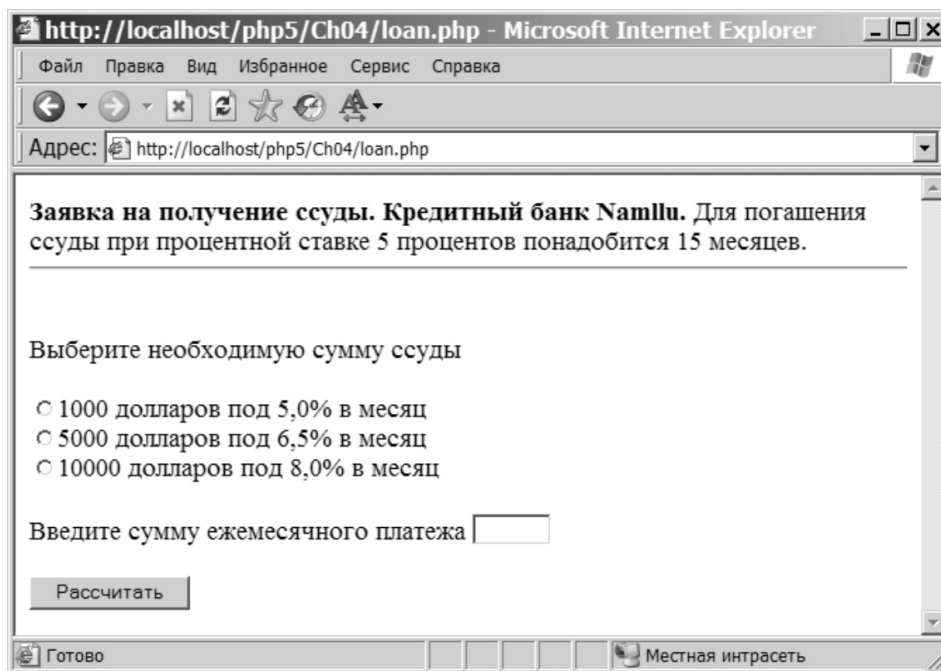


Рис. 4.6.

Как это работает

Форма предлагает пользователю выбрать сумму ссуды, которая затем сохраняется в переменной `$_POST['loan']`, и сумму ежемесячного платежа, которая хранится в переменной `$_POST['month']`. После этого данные переменные используются в сценарии `loan.php`. В представленном сценарии используется только две структуры. Первая структура — оператор `switch`, который описывался в предыдущем разделе. Сначала инициализируется переменная `$duration`, в которой будет храниться количество месячных платежей, необходимых для погашения ссуды. Затем для выбора процентной ставки используется оператор `switch`, поскольку процентная ставка зависит от ссуды, выбранной пользователем в HTML-форме:

```
$duration = 0;
switch ($_POST['loan']) {
case "1000";
    $interest = 5;
    break;
```

```

case "5000";
    $interest = 6.5;
    break;
case "10000";
    $interest = 8;
    break;
default:
    echo "Вы не выбрали вариант ссуды<hr>";
    exit;
}

```

Существует только три возможных значения `$_POST[loan]`: 1000, 5000 и 10000. Значение переменной `$loan` используется для того, чтобы с помощью оператора `switch` присвоить переменной `$interest` соответствующую процентную ставку. Если ни одно из значений не было выбрано, то пользователю выдается соответствующее сообщение, и программа останавливает свою работу.

Прежде чем программа войдет в цикл `while`, необходимо проверить условие. Условие заключается в том, что цикл может повторяться до тех пор, пока значение переменной `$_POST[loan]` не достигнет нуля; иными словами, пока имеется долг:

```
while ($_POST[loan] > 0){
```

Значение переменной `$_POST[loan]` — условие, с которого начинается цикл, так как значение `$loan` должно быть равно 1000, 5000 или 10000, но не 0. Затем счетчик месяцев `$duration`, первоначально равный нулю, увеличивается на единицу, чтобы показать, что это первая итерация цикла:

```
$duration = $duration + 1;
```

В следующей строке вычисляется сумма, которая будет выплачена по ссуде в данном месяце, включая проценты:

```
$monthly = $_POST['month'] - ($_POST['loan'] * $interest / 100);
```

Чтобы получить месячную сумму выплаты по процентам, процентная ставка делится на 100%, а результат умножается на сумму ссуды. Затем, чтобы определить сумму ссуды, оставшуюся в данном месяце, необходимо вычесть платеж по процентам за этот месяц из фиксированного месячного платежа. Результат будет ответом на первую часть выражения, приведенного в начале данного примера:

Платеж = Месячный платеж - Проценты

Однако для чего в этой точке программы создается переменная `$monthly`? Почему вместо этого нельзя использовать следующую строку:

```
$month = $month - ((($_POST['loan'] / 100) * $interest);
```

Причина заключается в том, что значение переменной `$_POST['month']` должно оставаться постоянным во всех циклах, представляя, таким образом, фиксированный ежемесячный платеж 100 долларов. Если использовать данную строку кода, то значение `$_POST['month']` будет изменяться в каждой итерации цикла, так как из него будет вычитаться месячный платеж. Поэтому, чтобы представить общую сумму платежа в данном месяце, используется другая переменная, `$monthly`. После каждой итерации цикла в ней содержится другое значение.

Проигнорируем пока оператор `if`; он не изменяет значений переменных. Теперь, когда имеется значение, которое необходимо вычесть из остатка ссуды за данный месяц, можно получить общий остаток ссуды, `$_POST['loan']`:

```
$_POST['loan'] = $_POST['loan'] - $monthly;
```

Рассмотрим цикл. В начале первой итерации переменные имеют свои первоначальные значения:

```
$_POST['loan'] = 1000
$interest = 5
$duration = 0
$_POST['month'] = 100
```

поэтому первая строка фактически означает

```
while (1000 > 0)
```

Так как 1000 больше 0, программа может безопасно войти в цикл, начиная с первой строки его кода, которая теперь означает

```
$duration = 0 + 1;
```

Теперь `$duration` равно единице. Далее следует строка

```
$monthly = 100 - (1000 * 5 / 100);
```

Вычислив результат в скобках, получаем

```
$monthly = 100 - (50);
```

Значение переменной `$monthly` равно 50. Программа достигла последней строки цикла:

```
$_POST['loan'] = 1000 - 50;
```

Поэтому `$loan` теперь равна 950. Затем следует закрывающая фигурная скобка, которая означает конец итерации. Программа возвращается обратно к проверке условия цикла, и теперь условие таково:

```
while (950 > 0)
```

Условие снова соблюдается и программа снова входит в цикл. `$duration` увеличивается на 1, и

```
$duration = 1 + 1;
```

Теперь переменная `$duration` содержит значение 2. Следующая строка несколько изменяется по сравнению с предыдущей итерацией, так как отличается значение переменной `$loan`:

```
$monthly = 100 - (950 * 5 / 100);
```

В результате этого переменная `$monthly` получает значение 52,5. Это влияет на вычисление остатка ссуды:

```
$_POST[loan] = 950 - 52.5;
```

После этого в переменной `$_POST['loan']` хранится значение 897,5. Фактический платеж по процентам уменьшается со временем, так как уменьшается остаток ссуды. Однако месячный платеж по ссуде фиксированный, поэтому фактическая выплата по ссуде с каждым месяцем возрастает. В первом месяце пользователь выплачивает 50 долларов, во втором — 52,50 и т.д.

Программа снова возвращается к проверке условия цикла:

```
while (897.5>0)
```

Условие верно, поэтому начинается третья итерация цикла. На самом деле цикл должен выполняться 15 раз, прежде чем долг будет равен нулю.

Бесконечные циклы

Известно, что очень медленная выплата ссуды ведет к тому, что проценты по ссуде превышают платежи, и ссуда постепенно возрастает, поэтому ее никогда нельзя будет выплатить. Эти устрашающие для человека перспективы также могут иметь фатальные последствия для программы. Если пользователь введет слишком маленькую сумму ежемесячного платежа, то цикл программы будет продолжаться бесконечно долго, потому что условие `$loan > 0` всегда будет верным. По умолчанию PHP-сценарии могут выполняться 30 секунд независимо от того, если ли в программе бесконечный цикл. Однако если сервер работает под управлением Windows 2000 и IIS 5, то браузер может в конце концов зависнуть, а если не предпринять определенные меры предосторожности, то зависнет также и Web-сервер.

Это означает, что необходимо позаботиться о том, чтобы бесконечные циклы не возникали. В рассматриваемом здесь примере это можно сделать путем проверки минимального значения платежа; однако данное значение полностью зависит от выбранного варианта ссуды. Следовательно, проверку необходимо реализовать внутри цикла. Теперь бесконечный цикл возникает только в том случае, если значение `$loan` каждый месяц увеличивается. Еще раз рассмотрим соответствующую строку в программе:

```
$_POST['loan'] = $_POST['loan'] - $monthly;
```

`$monthly` *вычитается* из `$_POST['loan']`. Сумма ссуды будет увеличиваться только тогда, когда значение переменной `$monthly` будет отрицательным (вычитание отрицательного числа эквивалентно прибавлению положительного). Поэтому необходимо проверять, не является ли значение `$monthly` меньшим или равным нулю. Пока это так, можно быть уверенным, что ссуда будет уменьшаться с каждой итерацией цикла, и число таких итераций конечно. На самом деле неважно, сколько будет итераций: PHP способен выполнять тысячи итераций в секунду.

Если создается условие для бесконечного цикла, то следует выйти из цикла с помощью команды `exit`, предварительно отправив пользователю соответствующее сообщение:

```
if ($monthly <= 0)
{
    echo "Чтобы погасить ссуду, требуются более крупные ежемесячные платежи<br>";
    exit;
}
```

После этого выводить какую-либо информацию не имеет смысла, поэтому используется команда `exit`, предотвращающая дальнейший вывод данных на Web-страницу.

С другой стороны, если итерации заканчиваются корректно, то отображается окончательное значение переменной `$duration`:

```
echo "Для погашения ссуды при процентной ставке $interest процентов
понадобится $duration месяцев.<br>";
```

Таким образом, пользователь сможет узнать, как долго ему придется выплачивать ссуду. Несмотря на то, что цикл состоит всего из 5 строк кода, чтобы рассмотреть его подробно, понадобится некоторое время. Если работа цикла не совсем ясна, вы можете смоделировать его работу на бумаге, используя несколько отличающиеся цифры и записывая результаты для каждой итерации цикла, а затем сверить полученные данные с тем, что возвращает PHP. Данные в обоих случаях должны быть идентичными.

Циклы do while

Цикл `do while` работает аналогично циклу `while` за исключением одной небольшой детали: условие проверяется в конце цикла, а не в начале. В этом кроется малоизвестное, но важное отличие: код в фигурных скобках выполняется, по крайней мере, один раз, даже если условие ложно:

```
do{
    выполнить код в данных скобках;
}
while(условие верно); // вернуться и выполнить код в скобках снова
```

Вернемся к примеру с покупательским приложением, на этот раз используя в нем цикл `do while`. Очевидно, что данный цикл меняет всю работу кода:

```
do{
    echo ("Вы превысили лимит своего кредита,
           поэтому последний объект будет удален из Вашей корзины");
    $last_purchase_but_one = $shopping_total - $last_purchase;
    $last_purchase = $last_purchase_but_one;
} while ($shopping_total > $credit_limit);
```

Предупреждение распечатывается еще до того, как программа проверит, не превысил ли пользователь свой кредитный лимит. Это совсем не то, что нужно.

Рассмотрим другую ситуацию, когда цикл `do while` будет действительно полезным. Предположим, необходимо проехать по междугородной автомагистрали от выезда 14 до выезда 103. Это можно описать с помощью следующего псевдокода:

```
do{
    ехать до следующего выезда;
} while ($exit != 103);
```

Если водитель захочет свернуть с автомагистрали, то ему, как минимум, придется доехать до следующего выезда, поэтому, по крайней мере, одна итерация необходима. А если следующим будет не 103-й выезд, то цикл понадобится выполнить еще раз и т.д. до тех пор, пока водитель не доедет до нужного выезда. Еще один хороший пример использования цикла `do while` в РНР — вычисления, которые необходимо выполнить хотя бы один раз, но, возможно, придется повторять вычисления до достижения определенного результата. Предположим, например, что необходимо создать программу для проверки простых чисел. Чтобы проверить, является ли число простым, его необходимо разделить на каждое число в диапазоне от двух и до самого числа, уменьшенного на единицу. Это можно сделать с помощью цикла `do while`:

```
do{
    $remainder = $possible_prime_number % $number;
    $number=$number + 1;
} while ($remainder != 0 AND $number < $possible_prime_number);
```

Проверяемое число `$possible_prime_number` необходимо разделить нацело (оператор `%`) на каждое число от 2 до самого `$possible_prime_number` минус 1 и проверить, если ли остаток от деления. Деление и проверка остатка выполняются в каждой итерации. Когда остаток равен нулю, программа должна выйти из цикла, потому что такое число может быть простым.

Чтобы определить, делится ли проверяемое число на другие числа, необходимо выполнить, как минимум, одно деление, и поэтому цикл `do while` в данной ситуации является наиболее подходящим. Если цикл доходит до самого проверяемого числа, то, очевидно, что это число делится только на само себя или на 1, поэтому данное число является простым.

Цикл `do while` также оказывается полезным в ситуации, когда программа должна дожидаться ввода данных от пользователя. В качестве примера можно модифицировать РНР-программу, которая “загадывала” числа от 1 до 10. Чтобы дать возможность пользователю угадывать число до тех пор, пока он не введет верный ответ, можно поместить соответствующий код в цикл `do while`, который будет повторяться вплоть до ввода загаданного числа.

Чтобы разобраться в работе цикла `do while`, рассмотрим пример, конкретизирующий фрагмент РНР-кода для проверки простых чисел.

Практика Использование циклов `do while`

1. Запустите редактор Web-страниц и введите следующий код:

```
<html>
<head><title></title></head>
<body>
<?php
if (isset($_POST['posted'])) {
    $count=2;
    do
    {
        $remainder = $_POST['guess'] % $count;
        $count = $count + 1;
    } while ($remainder != 0 and $count < $_POST['guess']);
    if (($count < $_POST['guess']) || ($_POST['guess'] == 0)) {
        echo ("Введенное число не является простым<br>");
    } else {
        echo ("Введенное число является простым<br>");
    }
}
?>
<form method="POST" action="check.php">
<input type="hidden" name="posted" value="true">
Введите число:
<input name="guess" type="text">
<br>
<br>
<input type="submit" value="Проверить число">
<br>
</form>
</body>
</html>
```

2. Сохраните данный файл как `check.php` и закройте его.
 3. Откройте созданный файл в браузере, введите число и нажмите кнопку Проверить число.
- Результат показан на рис 4.7.

Как это работает

Задача HTML-формы состоит исключительно в получении числа от пользователя:

Введите число:

Введенное число передается в переменную `$_POST['guess']`, которая затем устанавливает переменную `$count`:

```
$count=2;
```

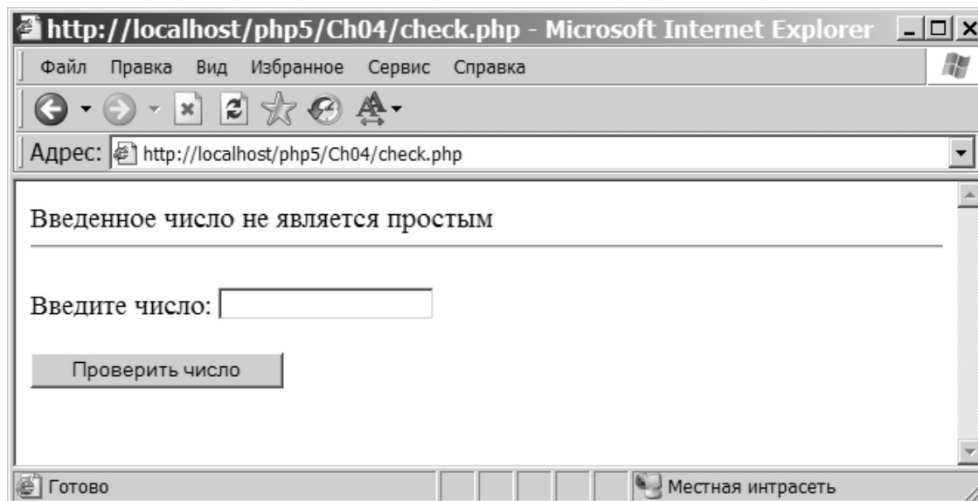


Рис. 4.7.

Данной переменной присваивается значение 2, потому что нет смысла начинать проверку с деления на 1; в любом случае каждое целое число делится на 1. Затем начинается цикл:

```
do{
```

Цикл начинается с проверки остатка от деления `$_POST['guess']` на `$count`. Затем значение переменной `$count` увеличивается на единицу:

```
$remainder = $_POST['guess'] % $count;
$count = $count + 1;
```

Эти две строки формируют все тело цикла. Следует учитывать, что если пользователь ввел число 25634, то в данном цикле оно будет делиться на 2, 3, 4, 5, 6, 7, 8 и т.д. до 25633; т.е. потребуется довольно много итераций. В конце каждой итерации проверяется два момента: есть ли остаток от деления и не достигло ли значение `$count` проверяемого числа `$_POST['guess']`:

```
} while ($remainder != 0 AND $count < $_POST['guess']);
```

Цикл прерывается, главным образом, когда возникает одно из двух указанных условий. Если цикл прервался, но значение `$count` не равно значению `$_POST['guess']`, то, скорее всего, проверяемое число было без остатка разделено на значение `$count`, поэтому `$_POST['guess']` не является простым числом. Необходимо также учитывать, что пользователь может ввести число нуль, которое дает нулевой остаток от деления на любое число, но не является простым. В противном случае, если две данные переменные имеют одинаковые значения, значит, проверяемое число является простым:

```
if (($count < $_POST['guess']) || ($_POST['guess'] == 0)) {
    echo ("Введенное число не является простым<br>");
} else {
    echo ("Введенное число является простым<br>");
}
```

Последний шаг — отобразить результаты пользователю.

Циклы for

Циклы `for` применяются, когда необходимо повторять какой-либо блок кода заданное количество раз. Иными словами, такой цикл дает возможность указывать число итераций. Условная часть в данном цикле более сложная, чем в цикле `while`, так как она состоит из трех частей:

```
for (установить счетчик цикла; проверить счетчик; увеличить или уменьшить счетчик) {
    выполнить код в фигурных скобках;
}
```

Цикл `for` вводит понятие счетчика цикла, т.е. переменной, которая используется для подсчета количества итераций и для останова цикла, после того как достигнуто заданное число итераций. Первые две части условия цикла `for` — установка счетчика цикла, т.е. указание необходимого числа итераций, и проверка счетчика — не превышает ли количество выполненных итераций счетчик цикла. Третья часть условия цикла гарантирует, что в каждой итерации значение счетчика изменяется. Эти три части условия позволяют конструировать сложные условия и циклы. Ни одна из частей условия не является обязательной, но для начала рекомендуется использовать все три части.

Как можно использовать цикл `for`? Например, можно 10 раз распечатать строку текста. Для этого можно 10 раз использовать команду `echo`, но это было бы довольно утомительно. Можно также использовать цикл `while` и создать внутри него переменную для подсчета итераций:

```
$counter=0;
while ($counter<10){
    echo "Привет!";
    $counter=$counter+1;
}
```

Однако сколько в таком случае итераций выполнит цикл? Девять, потому что когда счетчик достигает 10, то цикл останавливается? Или 11, потому что цикл начинается со счетчиком равным 0? Цикл выполнится действительно 10 раз, но придется вернуться назад и логически все проверить в уме. Гораздо проще контролировать количество итераций с помощью цикла `for`:

```
for ($counter = 1; $counter <= 10; $counter ++){
    echo "Привет!";
}
```

В данном случае вообще нет необходимости следить за счетчиком цикла, все делается автоматически.

Практическое применение цикла `for` описано в следующем разделе “Практика”.

Практика Использование циклов for

В следующем примере создается динамическая форма, которая принимает от пользователя некоторое число, использует это число для установки количества элементов управления, отображаемых на второй странице, а затем на третьей странице отображает содержимое этих элементов управления (следует отметить, что HTML-код всех трех страниц расположен в соответствующих операторах `if` внутри одного файла, и отображается только при соблюдении определенных условий, а именно когда переменные `$_POST[posted]` и `$_POST[posted01]` существуют).

1. Введите следующий код в редакторе Web-страниц:

```
<html>
<head><title></title></head>
```

```

<body>
<?php
if (isset($_POST['posted'])) {
    echo "<form method='POST' action='dynamic.php'>";
    for ($counter = 0; $counter < $_POST['number']; $counter++)
    {
        $offset = $counter + 1;
        echo "<br>Введите имя $offset-го ребенка<br>";
        echo "<input name='child[]' type='text'>";
    }
    echo "<br>Для продолжения нажмите кнопку<br>";
    echo "<input type='submit' value='Далее'>";
    echo "<input type='hidden' name='posted01' value='true'></form>";

} else {

    if (isset($_POST['posted01'])) {
        $count=0;
        echo "Имена Ваших детей:";
        do
        {
            $childs_name = $_POST['child'][$count];
            echo "<br><b>$childs_name</b>";
            $checkempty = $childs_name;
            $count = $count + 1;
        } while ($checkempty != "");
        if ($count == 1) {
            echo "Введите другое число";
        }
    }

    ?>
    <hr>
    <form method="POST" action="dynamic.php">
    <input type="hidden" name="posted" value="true">
    Сколько у Вас детей?
    <input name="number" type="text">
    <br>
    <br>
    <input type="submit" value="Отправить число">
    <br>
    </form>
    <?
    }
    ?>
</body>
</html>

```

2. Сохраните данный файл как `dynamic.php` и закройте его.
3. Откройте созданный файл в браузере (на рис. 4.8 показан созданный интерфейс) и введите число (для пробы введите число больше нуля).
4. Нажмите кнопку **Отправить число** и на следующей странице будет выведено несколько текстовых полей, количество которых равно введенному числу. На рис. 4.9 показан примерный результат для пользователя, который ввел число 3 в поле **Сколько у Вас детей?**
5. Введите в текстовые поля имена, нажмите кнопку **Далее** и на следующей странице отобразятся введенные имена (рис. 4.10).

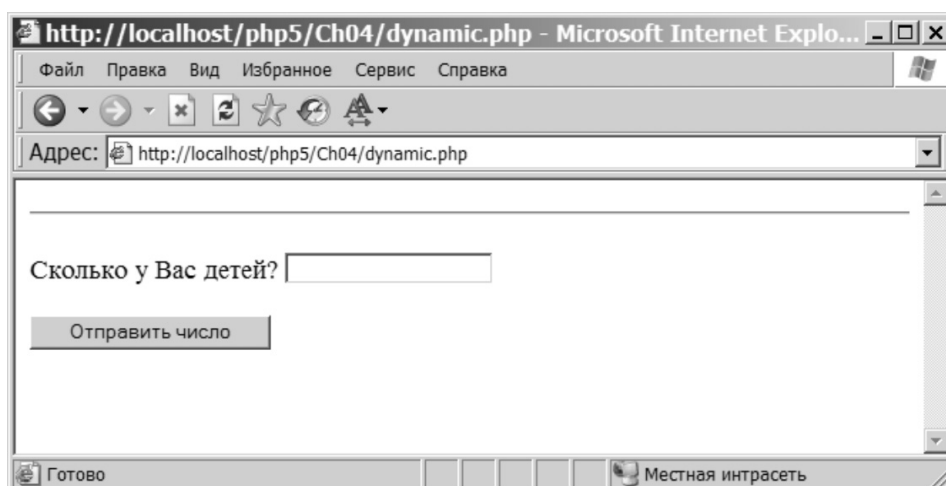


Рис. 4.8.

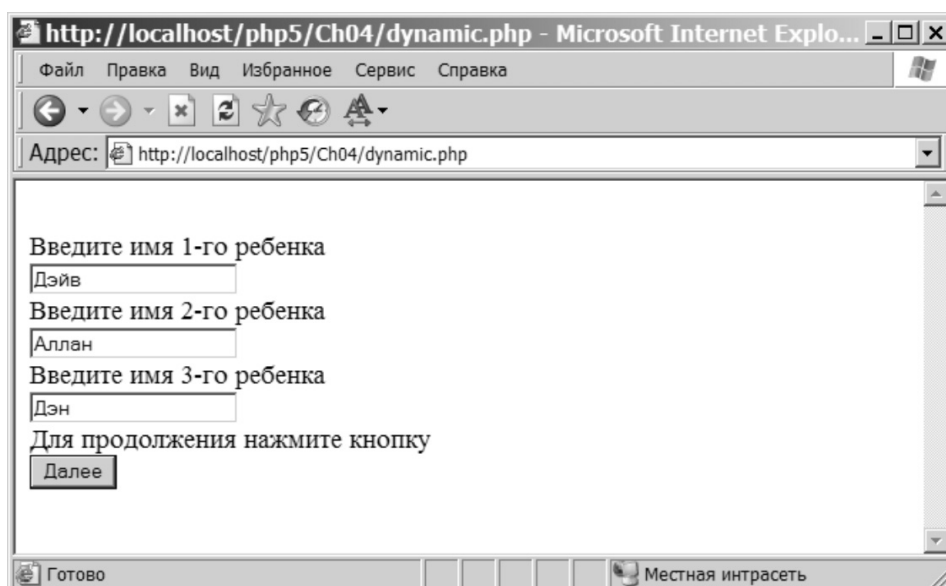


Рис. 4.9.

Как это работает

Описанная задача не настолько проста, как может показаться на первый взгляд. Программа начинается с простой формы, запрашивающей у пользователя число. Данное число передается сценарию как переменная \$number.

Сколько у Вас детей?

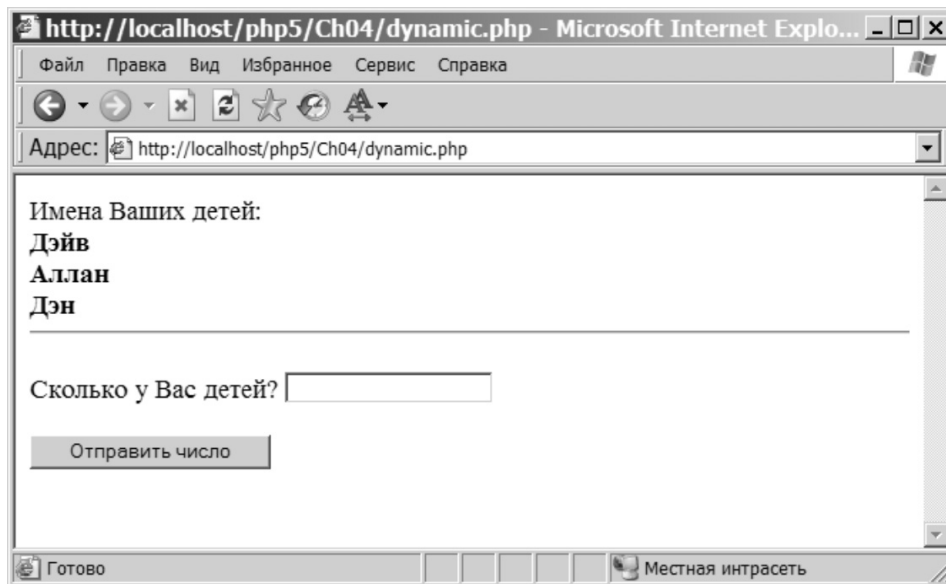


Рис. 4.10.

Затем в PHP-сценарии используется небольшой трюк. После того как программа обнаруживает, что форма была отправлена (проверив с помощью функции `isset()` существование переменной `$_POST['posted']`), браузеру отправляется тег `<form>` как часть оператора `echo`. По сути, программа, таким образом, сообщает браузеру, что необходимо создать форму:

```
if (isset($_POST['posted'])) {
    echo "<form method='POST' action='dynamic.php'>";
```

В форме необходимо создать определенное количество текстовых полей — столько, сколько указал пользователь на предыдущей странице; это число хранится в переменной `$_POST['number']`. Это идеальное время для начала цикла `for`. Необходимо отобразить несколько текстовых полей, количество которых соответствует значению переменной `$_POST['number']`, но если у пользователя нет детей, то отображать текстовые поля вообще не следует. Это означает, что счетчик `$counter` должен начинаться с нуля, и если значение `$_POST['number']` равно нулю, то необходимо прервать цикл. Итак, счетчик начинается с нуля и увеличивается на единицу в конце каждой итерации. Цикл продолжается до тех пор, пока значение счетчика меньше, чем количество детей (`$_POST['number']`):

```
for ($counter = 0; $counter < $_POST['number']; $counter ++){
```

Внутри цикла есть проблема. Необходимо обозначить каждое текстовое поле с помощью какого-либо номера, а, кроме того, если для этого использовать текущий счетчик цикла, то номер текстового поля будет на единицу меньше, чем должен быть. Чтобы обойти это препятствие, создается переменная смещения, значение которой равно `$counter` плюс один:

```
$offset = $counter + 1;
```

Затем отображается сообщение, приглашающее пользователя ввести имя ребенка:

```
echo "<br>Введите имя $offset-го ребенка<br>";
```

Затем добавляется текстовое поле. Здесь примечательно то, что когда для текстового поля устанавливается атрибут `name`, к имени текстового поля добавляются квадратные скобки, что означает создание массива. Поэтому каждый раз при передаче имени в `$_POST['child']` имя сохраняется в новом элементе данного массива: `$_POST['child'][0]`, `$_POST['child'][1]`, `$_POST['child'][2]` и т.д.:

```
    echo "<input name='child[]' type='text'>";
}
```

Здесь цикл фактически заканчивается. В каждой итерации цикла (количество итераций определяется переменной `$number`) просто отображается сопроводительный текст и текстовое поле. После выхода из цикла программа отображает кнопку отправки данных, вставляет еще одно скрытое поле формы с именем `posted01`, для того чтобы можно было определить вторую отправку формы, и, наконец, закрывает форму.

```
echo "<br>Для продолжения нажмите кнопку<br>";
echo "<input type='submit' value='Далее'>";
echo "<input type='hidden' name='posted01' value='true'></form>";
```

Второй блок кода, который активизируется `isset()`-проверкой переменной `$_POST[posted01]`, распечатывает имена из массива. На первый взгляд это просто, однако прежде всего необходимо учитывать, что переменная `$number` не была передана в новый сценарий. Она существовала только при отображении предыдущей страницы. Поэтому возникает ситуация, когда известны имена, переданные в массив `$_POST[child][]`, но неизвестно их количество. Значит можно прибегнуть к уже известному циклу `do while`. Необходимо вручную установить счетчик имен детей `$count`, начиная с нуля, так как имен может не быть.

```
} else {
    if (isset($_POST['posted01'])) {
        $count=0;
        echo "Имена Ваших детей:";
```

Затем начинается цикл `do while`, поскольку код цикла должен быть выполнен, по крайней мере, один раз:

```
do{
```

На входе в цикл отображается первый элемент массива, `$_POST[child][0]`. Если в массиве нет элементов, то выводится только разрыв строки. Иначе отображается имя первого ребенка:

```
    echo "<br><b>$_POST['child'][$count]</b>";
```

Переменной `$checkempty` присваивается значение элемента `$child[0]`. Если данный элемент массива пуст, то `$checkempty` позднее приведет к прерыванию цикла. Если первое имя существует, то цикл повторяется снова.

```
    $checkempty = "$_POST['child'][$count]";
```

Увеличивается счетчик:

```
    $count = $count + 1;
```

Выполняется проверка переменной `$checkempty`. Если она пуста, то цикл прерывается, иначе цикл продолжается:

```
    } while ($checkempty != "");
```

Затем выполняется проверка введенного количества детей. Если число не было введено, то отображается соответствующее сообщение:

```
    if ($count == 1) {
        echo "Введите другое число";
    }
}
```

Циклы являются мощным средством в программировании. Предположим, в данной программе пользователь ввел бы число 20. Без использования циклов в программе пришлось бы использовать 20 строк для считывания имен и 20 строк для их вывода. Кроме того, без циклов теряется динамичность, так как приходится фиксировать число считываемых полей с именами; циклы позволяют создавать более динамический код.

Массивы

Массивы уже встречались в примерах данной книги, теперь рассмотрим их более подробно. Массив представляет собой набор переменных, которые имеют одинаковые имена, но разные индексы. Каждый член массива называется *элементом*. Массивы создаются так же, как и обычные переменные, за исключением того, что имя каждого элемента должно заканчиваться индексом данного элемента в квадратных скобках:

```
$states_of_the_USA[1] = "Washington";
$states_of_the_USA[2] = "California";
```

Так как индексные номера являются числовыми значениями, использовать кавычки для указания индекса не требуется.

Присваивать значения элементам массива можно в любом порядке, указывая или не указывая связанный с элементом индексный номер:

```
$states_of_the_USA [49]="Alaska";
$states_of_the_USA [13]="Alabama";
```

Фактически использовать числовые индексы вообще необязательно — вместо них можно использовать символьные индексы. Символьные индексы указываются в кавычках в квадратных скобках. Массивы, в которых для обозначения элементов используются имена, а не индексные номера, часто называются ассоциативными массивами:

```
$state_capital["CA"] = "Sacramento";
$state_capital["IL"] = "Springfield";
```

Для того чтобы получить доступ к элементам ассоциативного массива, использовать кавычки вокруг имен элементов необязательно. Например, чтобы вывести на Web-странице слово Sacramento, можно использовать такой код:

```
echo $state_capital["CA"];
```

или

```
echo $state_capital[CA];
```

В обоих случаях на Web-странице появится название города. В документации на сайте PHP сказано, что возможность обращаться к элементам массива по именам без использования при этом кавычек, в конце концов, может быть устранена, но пока она очень хорошо работает.

Следует также упомянуть об одной особенности массивов в PHP — разным элементам одного массива можно присваивать различные типы данных и значения переменных, например:

```
$number[1]=24;
$number[2]="двадцать три";
$number[2]=$variable;
$number["CA"]=$variable;
```

Однако при этом возникает ряд вопросов. Как PHP определяет, каким должен быть размер массива? Сколько памяти следует выделить для данного массива? Ответы на эти вопросы вы найдете в последующих разделах.

Инициализация массивов

Инициализацией массива называется установка первоначальных значений переменных массива. В данной книге уже рассматривались два способа инициализации массивов (создание переменной массива путем именования и присвоения ей значения). Первый способ: программисту не приходится беспокоиться об индексации, РНР делает все автоматически. Создается один элемент массива, а затем создается другой элемент с тем же именем:

```
$author[] = "Вильям Шекспир";
$author[] = "Франц Кафка";
```

Без квадратных скобок РНР не сможет определить, что необходимо оперировать массивом, и просто заменит первое значение вторым. Скобки указывают на то, что значения необходимо сохранять в массиве. Пропуск индексного значения позволяет РНР самостоятельно решать, куда помещать переменные. Если массив `$author[]` ранее не использовался, то значения в данном примере будут сохранены в элементах `$author[0]` и `$author[1]`. РНР позаботится о присвоении новых значений следующим элементам массива.

Второй способ инициализации массива заключается в явном указании индексов:

```
$author[0] = "Вильям Шекспир";
$author[1] = "Франц Кафка";
```

В данном случае необязательно подчиняться ограничениям автоматической нумерации, которые в противном случае создает РНР; можно назначать индексы не по порядку. Что касается массивов, то РНР отличается от многих языков программирования в двух аспектах. Во-первых, не требуется предопределять тип данных массива, указывая на то, что массив будет содержать числа или текст. Это соответствует политике РНР в отношении переменных: программисту не обязательно выбирать тип данных переменной; РНР делает это автоматически. Во-вторых, также не требуется указывать размер массива до его создания, потому что РНР определяет максимальный индексный номер массива.

Существует еще два способа наполнения массивов в РНР, и в обоих используется конструкция `array()`. При первом способе значения присваиваются непосредственно без указания индексов или элементов массива. Например, фрагмент кода с созданием массива авторов можно переопределить следующим образом:

```
$author = array("Вильям Шекспир", "Франц Кафка");
```

В этом случае РНР также автоматически генерирует индексные значения.

Ограничений на размер массива не существует, поэтому можно написать следующую строку:

```
$states_of_the_USA = array("Alabama", "Alaska", "Arizona", "Arkansas",
"California", "Colorado", "Connecticut", "Delaware", "Florida", "Georgia",
"Hawaii", "Idaho", "Illinois", "Indiana", "Iowa", "Kansas", "Kentucky",
"Louisiana", "Maine", "Maryland", "Massachusetts", "Michigan", "Minnesota",
"Mississippi", "Missouri", "Montana", "Nebraska", "Nevada", "New Hampshire",
"New Jersey", "New Mexico", "New York", "North Carolina", "North Dakota",
"Ohio", "Oklahoma", "Oregon", "Pennsylvania", "Rhode Island", "South Carolina",
"South Dakota", "Tennessee", "Texas", "Utah", "Vermont", "Virginia",
"Washington", "West Virginia", "Wisconsin", "Wyoming");
```

РНР автоматически сгенерирует индексы. Первый штат при этом получит индекс равный нулю, а штат Вайоминг (Wyoming) будет иметь индекс 49.

Однако это может показаться несколько нелогичным; известно, что в США входит 50 штатов, и если последний штат получает номер 49, то можно ошибочно предположить, что один штат потерян. Второй способ создания переменной массива с помощью

функции `array()` позволяет обойти эту проблему. Для этого используется оператор `=>`, позволяющий выбрать индекс, с которого следует начинать отсчет:

```
$states_of_the_USA = array (1 => "Alabama", "Alaska", "Arizona", "Arkansas",
"California", "Colorado", "Connecticut", "Delaware", "Florida", "Georgia",
"Hawaii", "Idaho", "Illinois", "Indiana", "Iowa", "Kansas", "Kentucky",
"Louisiana", "Maine", "Maryland", "Massachusetts", "Michigan", "Minnesota",
"Mississippi", "Missouri", "Montana", "Nebraska", "Nevada", "New Hampshire",
"New Jersey", "New Mexico", "New York", "North Carolina", "North Dakota",
"Ohio", "Oklahoma", "Oregon", "Pennsylvania", "Rhode Island", "South Carolina",
"South Dakota", "Tennessee", "Texas", "Utah", "Vermont", "Virginia",
"Washington", "West Virginia", "Wisconsin", "Wyoming");
```

Иными словами, необходимо ввести индексное значение, с которого следует начинать индексацию, и оператор `=>`, а затем как обычно перечислить значения массива. Теперь, если вывести значение элемента `$states_of_the_USA[50]`, то на странице появится слово “Wyoming”. Необязательно использовать в качестве начального индекса единицу; можно начинать с индекса 101, и тогда штат Вайоминг получит индекс 150.

Если необходимо проиндексировать крупный ассоциативный массив, то придется устанавливать каждое значение отдельно. В таком случае в коде также используется оператор `=>`. Например:

```
$states_of_the_USA = array ("al" => "Alabama", "ak" => "Alaska", "az" =>
"Arizona", "ar" => "Arkansas", "ca" => "California", "co" => "Colorado", "ct"
=> "Connecticut", "de" => "Delaware", "fl" => "Florida", "ga" => "Georgia",
"hi" => "Hawaii", "id" => "Idaho", "il" => "Illinois", "in" => "Indiana", "ia"
=> "Iowa", "ks" => "Kansas", "ky" => "Kentucky", "la" => "Louisiana", "me" =>
"Maine", "md" => "Maryland", "ma" => "Massachusetts", "mi" => "Michigan", "mn"
=> "Minnesota", "ms" => "Mississippi", "mo" => "Missouri", "mt" => "Montana",
"ne" => "Nebraska", "nv" => "Nevada", "nh" => "New Hampshire", "nj" => "New
Jersey", "nm" => "New Mexico", "ny" => "New York", "nc" => "North Carolina",
"nd" => "North Dakota", "oh" => "Ohio", "ok" => "Oklahoma", "or" => "Oregon",
"pa" => "Pennsylvania", "ri" => "Rhode Island", "sc" => "South Carolina", "sd"
=> "South Dakota", "tn" => "Tennessee", "tx" => "Texas", "ut" => "Utah", "vt"
=> "Vermont", "va" => "Virginia", "wa" => "Washington", "wy" => "West
Virginia", "wi" => "Wisconsin", "wy" => "Wyoming");
```

Медленно, но задача индексации решается.

Итерации в массиве

Если создан массив с большим количеством элементов, то нежелательно возвращаться назад и по отдельности извлекать каждый элемент массива. Для этого требуется большая дополнительная работа. Облегчить эту работу позволяют циклы. Чтобы на Web-странице отобразить названия всех 50 штатов, можно просто использовать цикл `for`. Для отображения всех 50 штатов, проиндексированных от 1 до 50 в массиве `$states_of_the_USA`, в конечном итоге придется использовать три строки кода, не считая строку создания массива:

```
for ($counter=1; $counter<51; $counter++) {
    echo "<BR>$states_of_the_USA[$counter]";
}
```

Код в фигурных скобках выбирает элемент массива `$states_of_the_USA` с помощью переменной `$counter`, которая используется в качестве индекса. Так как `$counter` является переменной, индекс можно изменять, меняя значение данной переменной. Данный цикл `for` можно описать так: “Начать цикл с 1 и продолжать до 50, увеличивая счетчик на единицу”. Таким образом, переменная `$counter` замещается значениями 1, 2, 3 и т.д. для каждой итерации (разрыв строки (`
`) используется для того, чтобы выводить каждый штат в новой строке):


```
echo"<br>$states_of_the_USA[1];
echo"<br>$ states_of_the_USA[2];
echo"<br>$ states_of_the_USA[3];
и т.д.
```

Ничто не мешает использовать циклы `while` или `do while`, хотя для этого придется самостоятельно создать счетчик цикла и присваивать ему значения. Следующий цикл `while` делает то же, что и цикл `for`:

```
$counter = 1;
while ($counter < 51) {
    echo"<br> $states_of_the_USA[$counter]";
    $counter = $counter + 1;
}
```

Для этого требуется всего пара лишних строк.

Рассмотрим практический пример обработки массивов с помощью циклов.

Практика Итерация в массиве

В данном примере создается два массива: один, в котором содержатся названия всех штатов, и другой, в котором хранятся названия столиц всех штатов. Программа предоставит пользователю возможность выбирать штат из выпадающего списка, а затем, просматривая соответствующий массив, будет искать столицу штата. Массивы и циклы используются для того, чтобы не создавать вручную 50 строк HTML-кода для отображения ответов.

1. Откройте HTML-редактор и введите следующий код:

```
<html>
<head><title></title></head>
<body>
<?php
if (isset($_POST['posted'])) {
    $state_capital = array (0 => "Montgomery", "Juneau", "Phoenix",
    "Little Rock", "Sacramento", "Denver", "Hartford", "Dover", "Tallahassee",
    "Atlanta", "Honolulu", "Boise", "Springfield", "Indianapolis", "Des Moines",
    "Topeka", "Frankfort", "Baton Rouge", "Augusta", "Annapolis", "Boston",
    "Lansing", "Saint Paul", "Jackson", "Jefferson City", "Helena", "Lincoln",
    "Carson City", "Concord", "Trenton", "Santa Fe", "Albany", "Raleigh",
    "Bismarck", "Columbus", "Oklahoma City", "Salem", "Harrisburg", "Providence",
    "Columbia", "Pierre", "Nashville", "Austin", "Salt Lake City", "Montpelier",
    "Richmond", "Olympia", "Charleston", "Madison", "Cheyenne");
    for ($counter=0; $counter<50; $counter++)
    {
        if ($_POST['hiddenstate'][$counter]==$_POST['state'])
        {
            echo "Столица штата $_POST[state] - <b>$state_capital[$counter]</b><br>";
        }
    }
}
?>
<form action="capitals.php" method="POST">
<input type="hidden" name="posted" value="true">
Выберите штат, название столицы которого Вы хотели бы узнать
<select name="state">
<?php
$states_of_the_USA = array (1 => "Alabama", "Alaska", "Arizona", "Arkansas",
"California", "Colorado", "Connecticut", "Delaware", "Florida", "Georgia",
"Hawaii", "Idaho", "Illinois", "Indiana", "Iowa", "Kansas", "Kentucky",
"Louisiana", "Maine", "Maryland", "Massachusetts", "Michigan", "Minnesota",
```

```

"Mississippi", "Missouri", "Montana", "Nebraska", "Nevada", "New Hampshire",
"New Jersey", "New Mexico", "New York", "North Carolina", "North Dakota",
"Ohio", "Oklahoma", "Oregon", "Pennsylvania", "Rhode Island", "South
Carolina", "South Dakota", "Tennessee", "Texas", "Utah", "Vermont",
"Virginia", "Washington", "West Virginia", "Wisconsin", "Wyoming");
for ($counter = 1; $counter < 51; $counter ++ ) {
    echo "<option>$states_of_the_USA[$counter]</option>";
}
echo "</select><br><br>";
for ($counter = 1; $counter < 51; $counter++) {
    echo"<input type='hidden' name='hiddenstate[' '
value='$states_of_the_USA[$counter]'>";
}
?>
<input type="submit" value="Найти столицу">
</form>
</body>
</html>

```

2. Сохраните данный файл как `capitals.php` и закройте его.
3. Откройте созданный файл в браузере и выберите какой-либо штат.
4. Нажмите кнопку **Найти столицу**. На рис. 4.11 показан ответ для пользователя, который выбрал штат Гавайи (Hawaii).

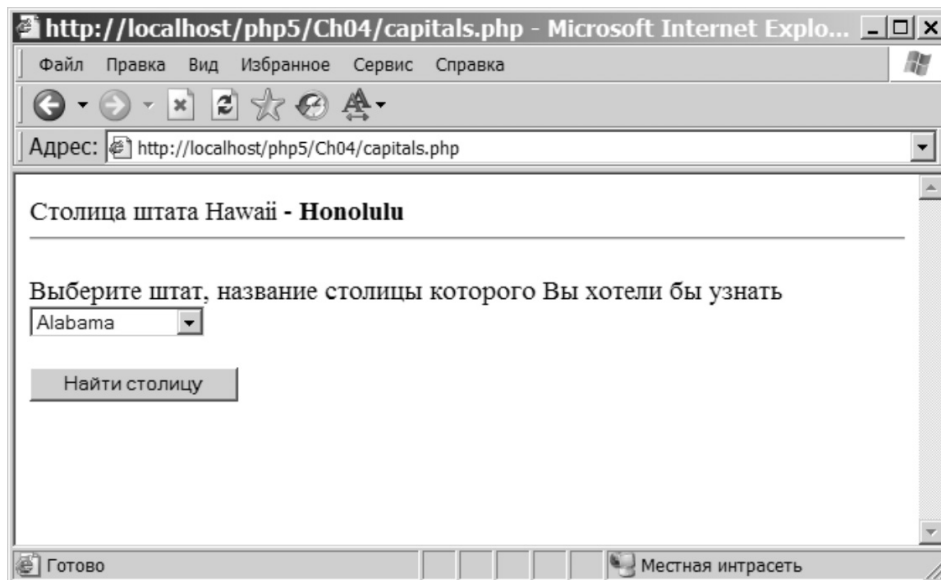


Рис. 4.11.

Как это работает

Данный пример демонстрирует то, насколько компактный код можно писать с помощью циклов и массивов.

В примере первая страница была создана как PHP-сценарий, чтобы можно было использовать функцию `array()` и, таким образом, избежать необходимости вводить 50 строк HTML-кода в списке `<select>`. Страница начинается с создания HTML-формы:

```
<form action="capitals.php" method="POST">
<input type="hidden" name="posted" value="true">
Выберите штат, название столицы которого Вы хотели бы узнать
```

Затем создается выпадающий список:

```
<select name="state">
```

Названия штатов помещены в массив `$states_of_the_USA`, начиная с индекса 1:

```
<?php
$states_of_the_USA = array (1 => "Alabama", "Alaska", "Arizona", "Arkansas",
"California", "Colorado", "Connecticut", "Delaware", "Florida", "Georgia",
"Hawaii", "Idaho", "Illinois", "Indiana", "Iowa", "Kansas", "Kentucky",
"Louisiana", "Maine", "Maryland", "Massachusetts", "Michigan", "Minnesota",
"Mississippi", "Missouri", "Montana", "Nebraska", "Nevada", "New Hampshire",
"New Jersey", "New Mexico", "New York", "North Carolina", "North Dakota",
"Ohio", "Oklahoma", "Oregon", "Pennsylvania", "Rhode Island", "South Carolina",
"South Dakota", "Tennessee", "Texas", "Utah", "Vermont", "Virginia",
"Washington", "West Virginia", "Wisconsin", "Wyoming");
```

Создается цикл для 50 итераций, помещающий название каждого штата в выпадающий список:

```
for ($counter = 1; $counter < 51; $counter++) {
    echo"<option>$states_of_the_USA[$counter]</option>";
}
echo "</select><br><br>";
```

Когда этот цикл заканчивается, список закрывается. Можно было бы предположить, что в следующей строке вставляется кнопка отправки данных и форма закрывается. Однако, забегаая вперед, можно заметить проблему: все штаты должны быть доступны не только на данной странице, но и на следующей. Почему? Рассмотрим поиск столицы определенного штата. Сначала, в массиве `$states_of_the_USA` необходимо найти индекс штата, а затем найти столицу данного штата, название которой хранится с тем же индексом в массиве `$state_capital`. Например, значением элемента `$states_of_the_USA[1]` является строка "Alabama", поэтому значение `$state_capital[1]` — "Montgomery". Это возможно, поскольку массивы были созданы так, чтобы гарантировать, что индекс столицы определенного штата в массиве `$state_capital` соответствует индексу штата в массиве `$states_of_the_USA`. Иными словами, данные в двух массивах согласованы по индексу.

Теперь, если передать следующему сценарию только название штата из списка без его индекса в `$states_of_the_USA`, то невозможно будет найти соответствующий индекс в массиве `$state_capital`, а, следовательно, не удастся найти столицу выбранного штата.

Как можно решить данную проблему? Очевидно, в следующий сценарий необходимо передать некоторую информацию об индексе штата, поэтому в данном примере в следующий сценарий передается весь массив штатов.

Необходимо создать второй цикл и использовать его для создания массива, содержащего 50 скрытых полей формы `$_POST['hiddenstate']`. Затем следует заполнить этот массив названиями штатов из массива `$states_of_the_USA`.

```
for ($counter = 1; $counter < 51; $counter++) {
    echo"<input type='hidden' name='hiddenstate[' '
value='$states_of_the_USA[$counter]'>";
}
```

После этого добавляется кнопка отправки запроса и форма закрывается:

```
<input type="submit" value="Найти столицу">
</form>
```

Теперь в программе имеется два набора данных. Первый получен из выпадающего списка и содержит название одного штата, выбранного пользователем. Эта информация хранится в переменной `$state`. Второй набор данных — список из 50 штатов в алфавитном порядке, сохраненный в массиве `$hiddenstate`. (Поскольку данный массив заполняется РНР автоматически, индексные номера начинаются с 0 и заканчиваются значением 49. Это не проблема; просто это следует иметь в виду. Первая строка сделает все, что нужно.) Теперь следует создать массив с названиями столиц 50 штатов. Чтобы индексы данного массива соответствовали индексам `$hiddenstate`, необходимо начать индексацию с нуля:

```
$state_capital = array (0 => "Montgomery", "Juneau", "Phoenix", "Little Rock",  
"Sacramento", "Denver", "Hartford", "Dover", "Tallahassee", "Atlanta", "Honolulu",  
"Boise", "Springfield", "Indianapolis", "Des Moines", "Topeka", "Frankfort",  
"Baton Rouge", "Augusta", "Annapolis", "Boston", "Lansing", "Saint Paul", "Jackson",  
"Jefferson City", "Helena", "Lincoln", "Carson City", "Concord", "Trenton", "Santa  
Fe", "Albany", "Raleigh", "Bismarck", "Columbus", "Oklahoma City", "Salem",  
"Harrisburg", "Providence", "Columbia", "Pierre", "Nashville", "Austin", "Salt  
Lake City", "Montpelier", "Richmond", "Olympia", "Charleston", "Madison", "Cheyenne");
```

Теперь массивы `$hiddenstate` и `$state_capital` согласованы по индексу, что значительно упрощает остальную часть программы. Фактически потребуются один цикл со счетчиком от 0 до 49:

```
for ($counter=0; $counter<50; $counter++)  
{
```

Внутри данного цикла имеется вложенный условный оператор, который проверяет совпадение текущего штата со штатом, выбранным пользователем. Если они совпадают, то на Web-странице отображается содержимое соответствующего элемента `$state_capital`:

```
if ($_POST['hiddenstate'][$counter]==$_POST['state'])  
{  
    echo "Столица штата $_POST[state] - <b>$state_capital[$counter]</b><br>";  
}  
}
```

Если значения не совпадают, то цикл `for` продолжается. Использование выпадающего списка обеспечивает конечное число вариантов, поэтому известно, что совпадение всегда найдется. Это все что требуется сделать в программе.

Усовершенствование программы

Предыдущая программа получилась не настолько простой, насколько она могла бы быть. Авторы намеренно пошли на это, чтобы показать, как два отдельных массива могут быть связаны по индексу. Гораздо проще было бы использовать ассоциативный массив, в котором элементы массива создаются с помощью следующей строки кода:

```
$state_capital = array ("Alabama" => "Montgomery", "Alaska" => "Juneau",  
"Phoenix" => "Arizona", "Arkansas" => "Little Rock", "California" =>  
"Sacramento", "Colorado" => "Denver", "Connecticut" => "Hartford", "Delaware"  
=> "Dover", "Florida" => "Tallahassee", "Georgia" => "Atlanta", "Hawaii" =>  
"Honolulu", "Idaho" => "Boise", "Illinois" => "Springfield",  
"Indiana" => "Indianapolis", "Iowa" => "Des Moines", "Kansas" => "Topeka",  
"Kentucky" => "Frankfort", "Louisiana" => "Baton Rouge", "Maine" => "Augusta",  
"Maryland" => "Annapolis", "Massachusetts" => "Boston", "Michigan" => "Lansing",  
"Minnesota" => "Saint Paul", "Mississippi" => "Jackson", "Missouri" =>  
"Jefferson City", "Montana" => "Helena", "Nebraska" => "Lincoln", "Nevada" =>  
"Carson City", "New Hampshire" => "Concord", "New Jersey" => "Trenton", "New  
Mexico" => "Santa Fe", "New York" => "Albany", "North Carolina" => "Raleigh",
```

```
"North Dakota" => "Bismarck", "Ohio" => "Columbus", "Oklahoma" => "Oklahoma
City", "Oregon" => "Salem", "Pennsylvania" => "Harrisburg", "Rhode Island" =>
"Providence", "South Carolina" => "Columbia", "South Dakota" => "Pierre", "Tennessee"
=> "Nashville", "Texas" => "Austin", "Utah" => "Salt Lake City", "Vermont" =>
"Montpelier", "Virginia" => "Richmond", "Washington" => "Olympia", "West
Virginia" => "Charleston", "Wisconsin" => "Madison", "Wyoming" => "Cheyenne");
```

Затем можно выводить название столицы, используя в качестве индекса имя штата в переменной `$state`.

Итерации в неупорядоченных массивах

Весьма просто реализовать цикл для работы с массивом, элементы которого создавались последовательно (первый элемент был введен первым, второй — вторым, третий — третьим и т.д.). Однако будет ли также просто обрабатывать массив, в котором элементы создавались не по порядку? Например:

```
$array[56993]="огромное количество";
$array[1]="очень мало";
$array[499]="достаточно много";
```

На самом деле неупорядоченные массивы не являются сложной проблемой, и все описанные ранее методы также будут работать. РНР “не обращает внимания” на то, что значения сохраняются не по порядку, так как он считает элементы массива расположенными в прямом числовом порядке индексных значений. Единственная проблема заключается в том, что при последовательном просмотре всех элементов большого массива (как в примере выше), который содержит только три значения, возможно, придется проверять множество несуществующих элементов.

Функции `current()` и `key()`

Для отслеживания элементов при перемещении по массиву в РНР используется указатель. Указатель ссылается на элемент, который используется сценарием в текущий момент времени. Чтобы просмотреть значение данного элемента, можно использовать функцию `current()`, а для того чтобы определить индекс данного элемента, можно использовать функцию `key()`. (Ключ (key) — еще одно название индекса.)

Ниже приведен небольшой фрагмент кода, иллюстрирующий работу функций `current()` и `key()`. Неясно, какой индекс РНР будет считать первым при первом просмотре РНР-сценария, содержащего такой код.

```
$director[4]="Orson Welles";
$director[1]="Carol Reed";
$director[93]="Fritz Lang";
$director[24]="Jacques Tourneur";
```

Это можно узнать, добавив несколько строк, которые возвращают текущий индекс массива `$director[]` и отображают его:

```
$current_index_value = key($director);
echo ($current_index_value);
```

Функция `key()` возвращает значение 4. Почему? Потому что она возвращает индекс элемента, который был помещен в массив первым (когда данный сценарий выполняется впервые, первый введенный элемент является текущим элементом), т.е. Orson Welles под индексом 4 (ключ, возвращенный функцией `key()`). Функция `current()` вернет значение Orson Welles (значение текущего элемента, найденного функцией `current()`):

```
$current_contents = current($director);
echo ($current_contents);
```

Теперь, если в массив добавить следующую строку:

```
$director[]="Alfred Hitchcock";
```

то строка `Alfred Hitchcock` будет помещена в массив и получит индекс 94, а не 0, потому что наибольшим значением используемых индексных чисел является 93, и PHP назначит следующему незаполненному элементу индекс 94. Как можно определить, какой индекс будет назначен новому элементу при условии, что функции `key()` и `current()` возвращают информацию по тому элементу, который был помещен в массив первым? Ответ на этот вопрос дан в следующем разделе.

Функции `next()` и `prev()`

Для определения индексного значения нового элемента, добавленного в массив, используются функции `next()` и `prev()`. Они позволяют перемещаться по массивам, передвигая указатель на следующий или предыдущий элемент соответственно. В качестве аргумента обе функции принимают имя массива. Например, для созданного в предыдущем примере массива:

```
$director[4]="Orson Welles";  
$director[1]="Carol Reed";  
$director[93]="Fritz Lang";  
$director[24]="Jacques Tourneur";  
$director[]="Alfred Hitchcock";
```

можно вызвать функцию `next()` до проверки текущего индекса и содержимого текущего элемента:

```
next($director);  
$current_index_value = key($director);  
echo ($current_index_value);
```

В результате этого будет отображено значение 1, а функция `current()` вернет строку `Carol Reed`.

Если вызвать функцию `next()` еще три раза

```
next($director);  
next($director);  
next($director);  
next($director);  
$current_index_value = key($director);  
echo ($current_index_value);
```

будет выведено значение 94. Если теперь вызвать функцию `current()`:

```
$current_contents = current($director);  
echo ($current_contents);
```

то в браузере будет выведена строка `"Alfred Hitchcock"`.

Функция `prev()` используется аналогично. Возьмем предыдущий пример и добавим вызов `prev()` после четырех вызовов `next()`:

```
next($director);  
next($director);  
next($director);  
next($director);  
prev($director);  
$current_index_value = key($director);  
echo ($current_index_value);
```

Ввиду того, что указатель был передвинут вперед на четыре элемента и на один назад, будет выведено число 24, которое является индексным значением, связанным со строкой `Jacques Tourneur`. Это довольно просто, хотя возникает один вопрос: что

случится, если вызов `next()` передвинет указатель за последний элемент массива или `prev()` установит указатель перед первым элементом?

```
prev($director);
$current_index_value = key($director);
echo ($current_index_value);
```

Ничего особенного не произойдет. Код не вернет никакого значения и не сгенерирует ошибку (как это было бы в некоторых языках программирования); указатель просто выходит за границу массива, но его можно вернуть обратно:

```
prev($director);
next($director);
next($director);
$current_index_value = key($director);
echo ($current_index_value);
```

В данном случае также не будет никакого вывода.

Итак, очевидно, что порядок, в котором заполняется массив, никак не влияет на порядок перемещения по элементам массива. Любое новое значение, добавленное в массив, помещается в элемент, который следует сразу за существующим элементом, имеющим наибольший индекс. Поэтому если наивысший индекс заполненного элемента равен 34, то следующее значение будет вставлено в массив с индексом 35.

Следует запомнить, что для перемещения по массиву используются функции `next()` и `prev()`, а для отображения текущей позиции (элемента, на который в текущий момент времени ссылается указатель) используются функции `current()` и `key()`.

Функции `list()` и `each()`

Можно упростить перемещение в неупорядоченных массивах. Вместо того чтобы обрабатывать в цикле множество пустых значений, можно использовать функции `list()` и `each()`, и возвращать только те элементы массива, которые содержат данные. Это позволяет с минимальными усилиями отображать все содержимое массива. Для этого используется цикл `while`:

```
while (list(IndexValue,ElementContents) = each(ArrayName))
```

Это означает: для каждого элемента массива `ArrayName` установить значение `IndexValue` равным индексу элемента, а значение `ElementContents` равным содержимому данного элемента. Если требуется возвращать либо индекс, либо значение, то лишний атрибут можно просто опустить:

```
while (list(IndexValue) = each(ArrayName))
```

или

```
while (list(,ElementContents) = each(ArrayName))
```

Для отображения имени каждого режиссера можно написать следующий код:

```
while (list($element_index_value, $element_contents) = each($director)){
    echo "<br>$element_index_value - $element_contents";
}
```

Называть переменные именами `$element_index_value` и `$element_contents` необязательно; это сделано просто для ясности. Их можно было бы назвать по-другому:

```
while (list($MickeyMouse, $DonaldDuck) = each ($Director)){
    echo "<BR>$MickeyMouse - $DonaldDuck";
}
```

В любом случае результат будет следующим:

```
4 - Orson Welles
1 - Carol Reed
93 - Fritz Lang
24 - Jacques Tourneur
94 - Alfred Hitchcock
```

Необходимо помнить только то, что функция `list()` возвращает сначала индекс, а затем содержимое элемента. Это предоставляет программисту некоторые полезные средства, которые применимы даже тогда, когда массив не имеет числовых индексов.

Итерации в ассоциативных массивах

Для перемещения по элементам ассоциативных массивов характерны те же правила, что и для массивов с числовыми индексами, однако имеются небольшие отличия. Во-первых, следующие строки, которые хорошо работали бы, если бы массив был численно индексированным, теперь создают ассоциативный массив:

```
$state_capital["GA"]="Atlanta";
$state_capital["IL"]="Springfield";
$state_capital["CA"]="Sacramento";
$state_capital[] = "Cheyenne";
```

Значение `Cheyenne` будет сохранено в элементе `$state_capital[0]`. Это не удивительно, учитывая, что РНР “не знает”, какие индексные значения намерен использовать программист; это может быть `WY` или `AB`, или `4563`, поэтому РНР пытается найти наибольший числовой индекс. Поскольку при создании массива числовые индексы не добавлялись, РНР начинает с нуля.

Функции `current()` и `key()` будут работать так, как и предполагается для ассоциативных массивов. Следующий код

```
$what_state = current($state_capital);
$what_abbreviation = key($state_capital);
echo "$what_abbreviation - $what_state";
```

вернет первый элемент массива, т.е. элемент с индексом `GA`, содержащий значение `Atlanta`. Можно перемещаться по элементам массива, как и раньше:

```
$state_capital["GA"]="Atlanta";
$state_capital["IL"]="Springfield";
$state_capital["CA"]="Sacramento";
$state_capital[] = "Cheyenne";
next($state_capital);
$what_state = current($state_capital);
$what_abbreviation = key($state_capital);
echo "$what_abbreviation - $what_state";
```

На этот раз отображается результат `IL - Springfield`. Функции `list()` и `each()` тоже работают. Можно создать следующий массив:

```
$state_capital = array ("GA" => "Atlanta", "IL" => "Springfield", "CA" =>
"Sacramento", "WY" => "Cheyenne");
```

А затем передавать данный массив функции `each` и использовать `list()` для отображения содержимого элементов:

```
while (list($state_abbreviation, $state_name) = each ($state_capital)){
    echo "<br>$state_abbreviation - $state_name";
}
```

При этом, как и ожидалось, создается список аббревиатур и названий штатов, а затем они отображаются в том порядке, в котором были созданы:


```
GA - Atlanta
IL - Springfield
CA - Sacramento
WY - Cheyenne
```

Как заставить PHP создать или сохранить другой порядок? Читайте об этом в следующем разделе.

Сортировка массивов

В PHP имеется несколько функций для сортировки массивов. В данном разделе рассматриваются пять наиболее широко используемых функций: `sort()`, `asort()`, `rsort()`, `arsort()` и `ksort()`. Все они работают совместно с уже рассмотренными функциями `list()` и `each()`.

Функция `sort()`

`sort()` — основная функция сортировки. Она принимает элементы массива и сортирует их в алфавитном порядке. Функции требуется передать только имя сортируемого массива:

```
sort(ArrayName)
```

Если массив режиссеров создается с помощью функции `array()`:

```
$director = array ("Orson Welles", "Carol Reed", "Fritz Lang", "Jacques
Tourneur");
```

то можно отсортировать элементы, передав функции `sort()` имя данного массива:

```
sort($director);
```

Чтобы понять, как данная функция влияет на массив, можно снова использовать функции `list()` и `each()`. Как уже отмечалось, в массиве элементы хранятся в том порядке, в котором они были созданы. В данном массиве ожидается следующий порядок элементов:

```
$director[0]= "Orson Welles"
$director[1]= "Carol Reed"
$director[2]= "Fritz Lang"
$director[3]= "Jacques Tourneur"
```

Однако после использования функции `sort()` порядок будет другим, так как функция расположила значения элементов в алфавитном порядке, а затем перестроила индексные номера так, чтобы они соответствовали новому порядку:

```
$director[0]= "Carol Reed"
$director[1]= "Fritz Lang"
$director[2]= "Jacques Tourneur"
$director[3]= "Orson Welles"
```

Это можно проверить, воспользовавшись ранее созданным фрагментом кода, который отображает на экране содержимое списка:

```
while (list($IndexValue, $DirectorName) = each ($director)){
    echo "<BR>$IndexValue - $DirectorName";
}
```

Функция `asort()`

Функция `asort()` принимает массивы со строковыми индексами и сортирует их согласно значениям элементов. Это не совсем то же самое, что делает функция `sort`. `asort()` — оставляет индексные номера или имена элементов нетронутыми, а не перестраивает их, как функция `sort`. Обратимся снова к массиву столиц штатов:

```
$state_capital = array ("GA" => "Atlanta", "IL" => "Springfield", "CA" =>
"Sacramento", "WY" => "Cheyenne");
```

Массив создается так же, как и при использовании следующего кода:

```
$state_capital["GA"] = "Atlanta"
$state_capital["IL"] = "Springfield"
$state_capital["CA"] = "Sacramento"
$state_capital["WY"] = "Cheyenne"
```

Вызовем для данного массива функцию `sort()`:

```
sort($state_capital);
```

В результате этого образуется следующий порядок:

```
$state_capital[0] = "Atlanta"
$state_capital[1] = "Cheyenne"

$state_capital[2] = "Sacramento"
$state_capital[3] = "Springfield"
```

Иными словами, строковые индексы были заменены числовыми индексами, что не очень полезно. Но если вместо этого вызывать функцию `asort()`:

```
asort($state_capital);
```

то порядок будет таким:

```
$state_capital["GA"] = "Atlanta"
$state_capital["WY"] = "Cheyenne"
$state_capital["IL"] = "Sacramento"
$state_capital["CA"] = "Springfield"
```

На этот раз элементы были отсортированы в алфавитном порядке, а строковые индексы остались. Чтобы убедиться, что это действительно так, можно снова использовать функции `list()` и `each()`:

```
while (list($state_abbreviation, $state_name) = each ($state_capital)) {
    echo "<br>$state_abbreviation - $state_name";
}
```

Функции `rsort()` и `arsort()`

Функции `rsort()` и `arsort()` работают аналогично своим близким родственникам `sort()` и `asort()`. Единственным отличием является то, что они возвращают элементы в обратном алфавитном порядке. Для того чтобы отсортировать список режиссеров в обратном алфавитном порядке, можно использовать функцию `rsort()`:

```
$director = array ("Orson Welles", "Carol Reed", "Fritz Lang", "Jacques
Touneur");
rsort($director);
```

Для сортировки списка столиц штатов можно использовать функцию `arsort()`:

```
$state_capital = array ("ga" => "Atlanta", "il" => "Springfield", "ca" =>
"Sacramento", "wy" => "Cheyenne");
arsort($state_capital);
```

Результаты должны быть очевидны. Чтобы проверить их, можно адаптировать код для отображения результатов из предыдущих разделов. Данные функции полезны, если требуется отсортировать элементы по убыванию, а не по возрастанию.

Функция ksort()

Функция `ksort()` сортирует содержимое ассоциативного массива по индексам. Она применяется точно так же, как другие функции сортировки, но упорядочивает ассоциативный массив в алфавитном порядке строковых индексов. Эта функция полезна для сортировки по индексным номерам или именам элементов, когда необходимо манипулировать данными по именам элементов, а не по значениям. Например, в случае списка столиц штатов:

```
$state_capital = array ("ga" => "Atlanta", "il" => "Springfield", "ca" =>
"Sacramento", "wy" => "Cheyenne");
ksort($state_capital);
```

функция `ksort()` создаст следующий порядок:

```
$state_capital["ca"] = "Sacramento"
$state_capital["ga"] = "Atlanta"
$state_capital["il"] = "Springfield"
$state_capital["wy"] = "Cheyenne"
```

Поэтому если для каких-либо целей понадобится использовать аббревиатуры штатов, а не их названия, то их также можно отсортировать.

Многомерные массивы

PHP позволяет создавать массивы массивов. Такие структуры получили название многомерных массивов. Они полезны для представления данных, которые требуют двух наборов индексов, например, координат на карте. В принципе, существует возможность создавать вложенные массивы в многомерном массиве до тех пор, пока PHP будет хватать памяти. Вероятно, этот предел наступает в районе 100 вложенных массивов (тот, кто сможет предложить целесообразное практическое применение для массивов с более чем 10 вложенными массивами, заслуживает особой награды). Многомерные массивы создаются так же, как и обычные массивы, за исключением того, что конструкция `array` вызывается как аргумент для самой себя:

```
ArrayName = array (index => array (элементы массива))
```

В качестве примера можно описать массив, в котором каждый элемент представляет одного члена какой-либо группы людей и одновременно является массивом, содержащим некоторые сведения об этом человеке:

```
$phone_directory = array ("John Doe" => array ("1 Long Firs Drive",
"777-000-000"), "Jane Doe" => array ("4 8th and East", "777-111-111"));
```

В данном массиве содержатся элементы John Doe и Jane Doe; и каждый из этих элементов в свою очередь представляет массив с двумя элементами: адресом и номером телефона. Чтобы распечатать данную информацию, необходимо использовать вложенный цикл:

```
$phone_directory = array ("John Doe" => array("1 Long Firs Drive",
"777-000-000"),
                        "Jane Doe" => array("4 8th and East", "777-111-111"));
while (list($person) = each($phone_directory)) {
    echo("<br>$person");
    while (list(,$personal_details) = each ($phone_directory[$person])) {
        echo (" $personal_details");
    }
}
```

Тег `
` и пробел перед `$personal_details` в операторе `echo` используются просто для того, чтобы сделать вывод информации удобочитаемым. Многомерные массивы встречаются не очень часто, поэтому здесь они не описываются.

Практическое использование массивов

Материал данной главы проиллюстрирован на кратком примере, который включает многие из рассмотренных функций.

Практика Комбинированное использование функций для работы с массивами

В следующем примере группе вымышленных студентов предлагается ответить на вопросы об оценках, которые они получили на экзамене по математике. Кроме всего прочего на Web-странице имена студентов будут отсортированы в порядке полученных студентами оценок (A — самая высокая оценка, F — самая низкая). Чтобы представить результаты в более читабельном виде, используется несколько HTML-таблиц (теги `<table>`).

1. Запустите редактор Web-страниц и введите следующий код:

```
<html>
<head><title></title></head>
<body>
<form method="POST" action="exam.php">
<input type="hidden" name="posted" value="true">
<table border="1">
<?php
$student = array("Альберт Эйнштейн", "Иван Грозный", "Наполеон", "Саймон
Боливар", "Исаак Ньютон");
while (list(,$name) = each($student))
{
    echo "<tr><td>Какую оценку по математике получил <b>$name</b>?</td>";
    echo"<td><select name='math[' . $name . ']'>
        <option>A</option>
        <option>B</option>
        <option>C</option>
        <option>D</option>
        <option>F</option>
    </select>";
    echo"<input type='hidden' name='student[' . $name . ']' value='$name'></td>";
}
?>
</tr>
<tr><td>&nbsp;</td><td>
<input type="submit" value="Показать оценки">
</td></tr>
</form>
</table>

<?php
if (isset($_POST['posted'])) {
?>
<hr>
<table border="1">
<tr><td colspan="2">
Оценки по математике расположены в следующем порядке:
</td></tr>
<?php
    while (list($index,$value)=each($_POST['math']))
    {
        $gradestudent[]=$_POST['math'][$index].$_POST['student'][$index];
    }
```

```

        asort($gradestudent);
        while (list($index,$value)=each($gradestudent))
        {
            $student_index = $_POST['student'][$index];
            $math_index = $_POST['math'][$index];
            echo
            "<tr><td><b>$student_index</b></td><td>$math_index</td></tr>";
        }
    }
    ?>
</table>
</body>
</html>

```

2. Сохраните данный файл как exam.php и закройте его.
3. Откройте созданный файл в браузере и введите несколько оценок (пример на рис. 4.12).

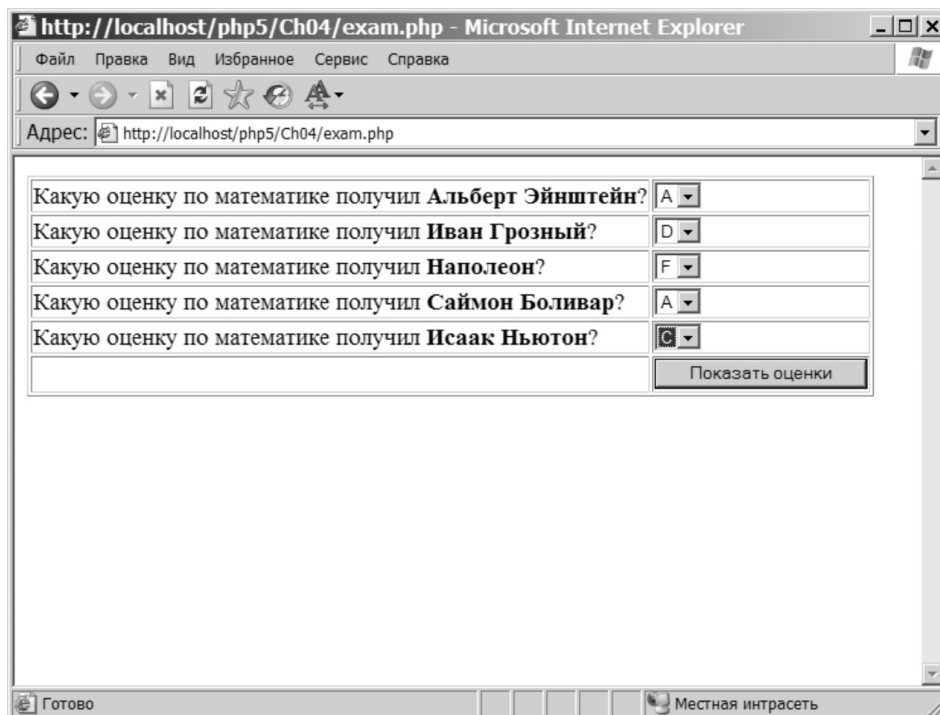


Рис. 4.12.

4. Нажмите кнопку Показать оценки и данные будут отсортированы не только по оценкам, но и в алфавитном порядке имен студентов, если несколько студентов получили одинаковые оценки (рис. 4.13).

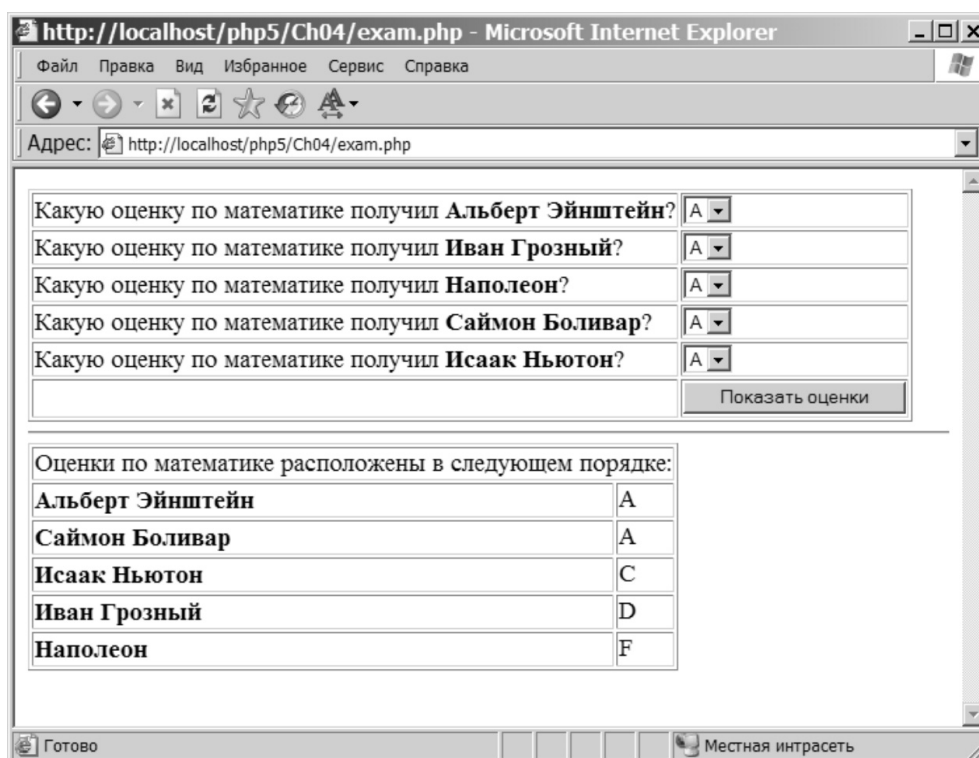


Рис. 4.13.

Как это работает

Представленная программа решает сложную задачу по сортировке полученных данных. Необходимо отсортировать массив оценок, которые согласованы по индексу с массивом студентов. Однако как можно связать отсортированный массив оценок с массивом студентов? Сделать это без особых сложностей позволяет один технический прием.

Программа создает динамическую форму, которая выводит имена студентов из массива `$student`:

```
<form method="POST" action="exam.php">
<input type="hidden" name="posted" value="true">
<table border="1">
<?php
$student = array("Альберт Эйнштейн", "Иван Грозный", "Наполеон", "Саймон
Боливар", "Исаак Ньютон");
```

Цикл `while` обрабатывает элементы данного массива. Понадобятся только имена студентов, а не их индексы, поэтому функции `list()` передается только второй параметр:

```
while (list(,$name) = each($student))
{
```

Для всех студентов в списке отображаются запросы оценок:

```
echo "<tr><td>Какую оценку по математике получил <b>$name</b>?</td>";
```

Затем создается выпадающий список, содержащий пять пунктов, соответствующих пяти оценкам от А до F. Для хранения оценок создается массив `$math`, который PHP сможет передать следующему сценарию:

```
echo"<td><select name='math[]' >
  <option>A</option>
  <option>B</option>
  <option>C</option>
  <option>D</option>
  <option>F</option>
</select>";
```

Значение для второй части сценария передается через скрытое поле формы, которое имеет то же имя, что и массив с именами студентов. Чтобы указать, что данный элемент управления должен быть массивом, к его имени добавляются квадратные скобки. Для каждого студента передается имя элемента управления, а затем цикл завершается:

```
echo"<input type='hidden' name='student[]' value='$name'></td>";
}
```

После цикла вставляется кнопка отправки запроса и форма закрывается:

```
?>
</tr>
<tr><td>&nbsp;</td><td>
<input type="submit" value="Показать оценки">
</td></tr>
</form>
</table>
```

Вторая часть программы получает данные из формы. Эта часть состоит из цикла, сортировки и еще одного цикла.

Первый цикл используется для связи двух массивов, полученных из формы. Это делается путем конкатенации оценки и имени студента и сохранения результата в новом массиве `$gradestudent`:

```
<hr>
<table border="1">
<tr><td colspan="2">
Оценки по математике расположены в следующем порядке:
</td></tr>
<?php

while (list($index,$value)=each($_POST['math']))
{
    $gradestudent[]=$_POST['math'][$index].$_POST['student'][$index];
}
```

Из кода первой части сценария известно, что значение элемента с определенным индексом в массиве `$math` связано со значением элемента с таким же индексом в массиве `$student`. Поэтому конкатенация представляет собой верный способ объединения связанных значений из каждого массива. Результирующий массив `$gradestudent` содержит следующие данные:

```
Сальберт Эйнштейн
ИИван Грозный
ВНаполеон
ДСаймон Боливар
АИсаак Ньютон
```

Теперь необходимо соответствующим образом отсортировать массив `$gradestudent`:

```
asort($gradestudent);
```

Необходимо решить проблему, связанную с тем, как отображать результаты, которые выглядят несколько неприглядно в формате, показанном до сортировки. Используя следующий фрагмент кода, можно распечатать отсортированный список оценок рядом с именами студентов:

```
while (list($index,$value)=each($gradestudent))
{
    echo
    "<tr><td><b>$_POST['student'] [$index]</b></td><td>$_POST[math] [$index]</td>
</tr>";
}
?>
</table>
```

Чтобы понять, как это работает, следует вспомнить, что массив `$gradestudent` отсортирован по оценкам. Во время сортировки порядок элементов изменился, индексы связаны со значениями так же, как и до сортировки. Каждый элемент массива `$gradestudent` представляет собой конкатенацию соответствующих элементов массивов `$math` и `$student`. Что же произойдет, если отображать элементы из массивов `$_POST['math']` и `$_POST['student']` в том порядке, в каком их индексы расположены в массиве `$gradestudent`? Поскольку индексы всех этих массивов согласованы, элементы массивов `$math` и `$student` выводятся отсортированными также по оценкам, как показано в следующей таблице:

<i>\$index</i>	<i>\$student</i>	<i>\$math</i>	<i>Отсортированный массив \$gradestudent</i>
4	Исаак Ньютон	A	"АИсаак Ньютон"
2	Наполеон	B	"ВНаполеон"
0	Альберт Эйнштейн	C	"САльберт Эйнштейн"
3	Саймон Боливар	D	"ДСаймон Боливар"
1	Иван Грозный	F	"ФИван Грозный"

Известно, что использование функций `list()` и `each()` позволяет получить значения из отсортированного массива `$gradestudent` в том порядке, в котором они расположены в массиве, поэтому код выводит элементы из массивов `$_POST['math']` и `$_POST['student']` также отсортированными по оценкам.

Функция `array_multisort()`

Функция `array_multisort()` сортирует несколько массивов или многомерные массивы. В качестве аргументов она принимает имена массивов. При работе с несколькими массивами функция сортирует первый массив и фиксирует индексы всех повторяющихся элементов данного массива. Затем она сортирует повторяющиеся элементы согласно соответствующим индексам во втором массиве. Наконец, функция сортирует второй массив в том же порядке, что и первый. Рассмотрим предыдущий пример. Можно изменить код в сценарии `exam.php` и использовать функцию `array_multisort()`:

```
<?php
array_multisort($math,$student);
while (list($index,$value)=each($student))
{
    echo "<br>$student[$index] - $math[$index] ";
}
?>
```


Теперь представим, что произойдет, если данная функция отсортирует следующие оценки (в массиве `$math`) для данных студентов (имена которых хранятся в массиве `$student`):

```
Альберт Эйнштейн A
Иван Грозный F
Наполеон D
Саймон Боливар D
Исаак Ньютон A
```

Сначала функция `array_multisort()` сортирует массив `$math[]`, в результате чего оценки располагаются так: A, A, D, D, F. Поскольку для каждой из оценок A и D имеется две записи, функция запоминает индексы двух A-записей (0 и 4), обращается ко второму массиву `$student[]` и сортирует элементы 0 (Альберт Эйнштейн) и 4 (Исаак Ньютон) по алфавиту. То же происходит с двумя D-записями. Когда рассматриваемая функция заканчивает сортировку, она фиксирует окончательный порядок индексов массива `$math` (0, 4, 2, 3, 1) и сортирует в этом порядке массив `$student[]`, после чего оба массива оказываются отсортированными сначала по оценкам, а затем по именам.

Циклы foreach

Расширение цикла `for` называется цикл `foreach` и используется для обработки массивов с неизвестным числом элементов. Итерации в таком цикле повторяются вплоть до достижения конца массива. Данный цикл имеет два формата. Первый формат

```
foreach ($ArrayName As $ArrayItem){
    выполнить код в данных фигурных скобках;
}
```

означает, что для каждого элемента в массиве будет выполнена итерация цикла. В качестве примера предположим, что требуется сохранить в базе данных имя каждого болельщика, присутствующего на бейсбольном матче. До начала игры невозможно сказать, сколько болельщиков окажется на стадионе. Этот пример кратко можно описать так:

```
foreach ($crowd As $fan){
    Добавить $fan в базу данных {...}
}
```

Цикл обходит болельщиков и добавляет в базу по одному имени за раз. Не обязательно сообщать количество присутствующих на матче болельщиков, PHP может выяснить это самостоятельно по количеству элементов в массиве `$crowd`.

Рассмотрим второй формат:

```
foreach ($ArrayName As $ArrayIndexValue => $ArrayItem){
    выполнить код в фигурных скобках
}
```

Такая форма записи очень похожа на описанную выше, но она также делает доступными индексы массива. Рассмотрим вкратце работу цикла `foreach` на примере.

Практика Использование цикла foreach

В следующем примере используется перечень штатов, каждый штат в массиве отображается наряду с индексным значением, которое PHP присвоил данному штату.

1. Запустите редактор Web-страниц и введите следующий код:

```
<html>
<head><title></title></head>
<body>
```

```

<?
$states_of_the_USA = array ("Alabama", "Alaska", "Arizona", "Arkansas",
"California", "Colorado", "Connecticut", "Delaware", "Florida",
"Georgia", "Hawaii", "Idaho", "Illinois", "Indiana", "Iowa", "Kansas",
"Kentucky", "Louisiana", "Maine", "Maryland", "Massachusetts",
"Michigan", "Minnesota", "Mississippi", "Missouri", "Montana",
"Nebraska", "Nevada", "New Hampshire", "New Jersey", "New Mexico",
"New York", "North Carolina", "North Dakota", "Ohio", "Oklahoma", "Oregon",
"Pennsylvania", "Rhode Island", "South Carolina", "South Dakota",
"Tennessee", "Texas", "Utah", "Vermont", "Virginia", "Washington",
"West Virginia", "Wisconsin", "Wyoming");
foreach($states_of_the_USA as $state_index => $state)
{
    echo "<br>$state_index - $state";
}
?>
</body>
</html>

```

2. Сохраните данный файл как `foreach.php` и закройте его.
3. Откройте созданный файл в браузере и убедитесь, что на странице выведены все 50 штатов вместе с числами от 0 до 49 (рис. 4.14).

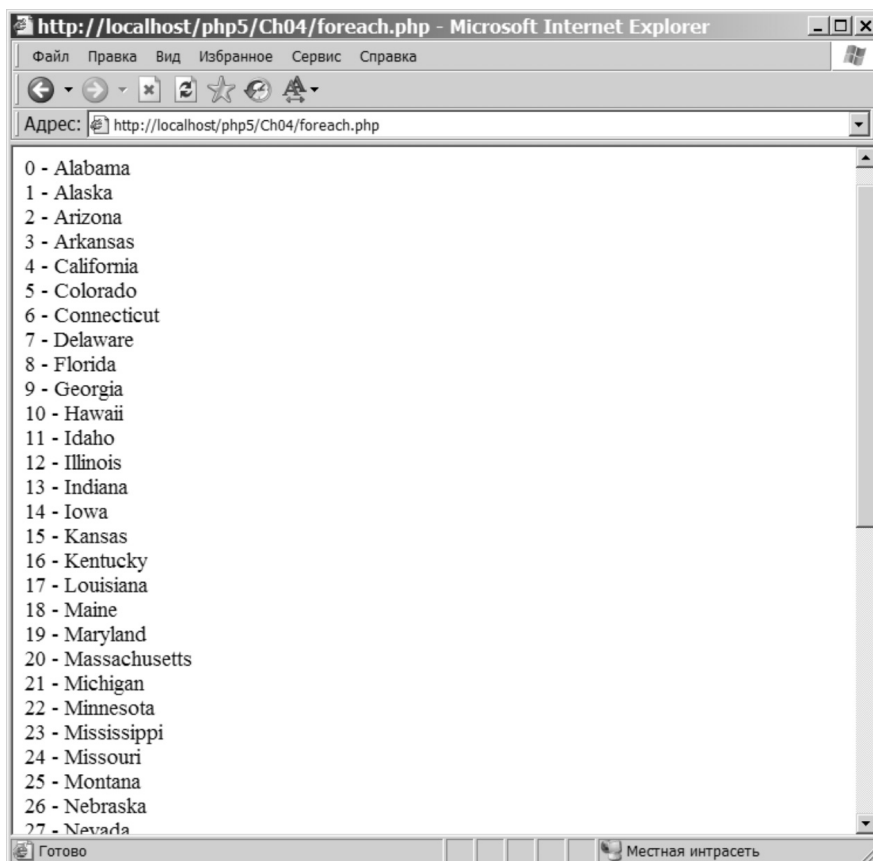


Рис. 4.14.

Как это работает

Данная программа очень проста. Сначала создается массив с названиями всех 50 штатов, который называется `$states_of_the_USA`:

```
$states_of_the_USA = array ("Alabama", "Alaska", "Arizona", "Arkansas",
"California", "Colorado", "Connecticut", "Delaware", "Florida", "Georgia",
"Hawaii", "Idaho", "Illinois", "Indiana", "Iowa", "Kansas", "Kentucky",
"Louisiana", "Maine", "Maryland", "Massachusetts", "Michigan", "Minnesota",
"Mississippi", "Missouri", "Montana", "Nebraska", "Nevada", "New Hampshire",
"New Jersey", "New Mexico", "New York", "North Carolina", "North Dakota",
"Ohio", "Oklahoma", "Oregon", "Pennsylvania", "Rhode Island", "South Carolina",
"South Dakota", "Tennessee", "Texas", "Utah", "Vermont", "Virginia",
"Washington", "West Virginia", "Wisconsin", "Wyoming");
```

Затем имя данного массива передается в качестве первого аргумента цикла `foreach`:

```
foreach($states_of_the_USA as $state_index => $state){
```

Второй и третий аргументы — имена созданных переменных, в которых содержатся индексное значение и связанный с ним элемент массива соответственно. Код в фигурных скобках выполняется для каждого из 50 членов массива:

```
    echo "<br>$state_index - $state";
}
```

В результате этого на странице построчно отображается каждое индексное значение и соответствующее ему название штата.

Здесь следует отметить три момента. Во-первых, текущий элемент массива и индекс доступны как отдельные переменные. Так как в каждой итерации обрабатывается новый элемент массива, этим переменным при каждой итерации присваиваются новые значения.

Во-вторых, указывать количество элементов массива не требуется. PHP определяет их количество автоматически, и в этом заключается главное преимущество цикла `foreach`. Это позволяет перемещаться по массиву, который может быть не упорядоченным ни численно, ни по алфавиту. Кроме того, в массиве могут быть пропущенные записи, но PHP проверяет не каждое индексное значение, а только те элементы, которые содержат какие-либо значения. Можно было бы включить следующий код, который добавляет в массив штатов новый элемент:

```
$states_of_the_USA[100]="Atlantis";
```

Затем при повторном запуске сценария PHP добавил бы новое значение в конец Web-страницы, не проверяя при этом элементы с 50 по 99.

Резюме

В данной главе рассматривалась главная часть языка PHP — логика принятия решений. В ее основе лежит оператор `if`, который имеет несколько различных форматов и представляет собой довольно гибкое средство, поскольку допускает вложения и может быть расширен с помощью конструкции `else if`. Оператор `switch` предлагает более развитый метод обработки нескольких условий, и хотя в главе даны рекомендации о том, когда следует применять операторы `if`, `else if` и `switch`, в конечном счете это личный выбор программиста.

По мере углубления в материал главы увеличивались размеры примеров, а их код часто повторялся. В конце главы было показано, как PHP справляется с повторениями с помощью циклов. Одной из главных причин использования циклов является то,

что они могут сократить количество необходимого кода. Циклы также можно использовать для написания кода, который обрабатывает переменные массивов и делает это быстрее и эффективнее, чем линейный код.

Очевидно, что хотя циклы и массивы являются концептуально различными элементами языка программирования, в РНР они, по сути, тесно связаны. Циклы необходимы для выполнения повторяющихся операций, одна из таких операций — обработка больших связанных наборов индексированных переменных, каковыми являются массивы. В главе рассматривалось три типа циклов — `while`, `do while` и `for`. Каждый из них подходит для решения различных типов задач, поэтому выбор используемого цикла зависит от ситуации. В данной главе подробно рассматривались массивы и методы для работы с ними.

Были рассмотрены РНР-функции для перемещения по элементам массивов, в том числе и по элементам массивов, которые не имеют последовательных или числовых индексов. Кроме того, рассматривались возможности сортировки массивов и некоторые практические примеры использования массивов.

Упражнения

“У нас есть Web-сайт, но информация на нем устарела. Он выглядит не очень привлекательно, но мы уже наняли дизайнера, которой переделает логотип, поэтому как только он закончит свою работу, трафик, вероятно, возрастет. Мы собираемся нанять персонал для поддержки растущих запросов и хотим разместить на сайте программу для сбора резюме.”

“Посетители должны легко находить на сайте перечень вакансий, но эта ссылка не должна быть главным элементом страницы. Мы должны предоставить пользователю возможность искать работу и оставлять свою контактную информацию на сайте. Те, кто не имеет высшего образования, могут претендовать только на работы начального уровня в отделе продаж или отделе доставки. Заработные платы ни по одной из вакансий для начала не превышают 20 000 долларов, на руководящих постах предлагаются поощрительные премии. Соискатели должны отправлять свои пожелания по заработной плате и иметь минимум двухлетний опыт работы. Желающие получить работу на руководящей позиции, в том случае если они не имеют докторской степени, должны иметь, по крайней мере, пятилетний опыт работы. Мы бы хотели, чтобы люди могли искать на сайте работу и подавать заявки на интересующие их позиции, а также чтобы они могли отправлять свои резюме и получать информацию о том, на какую работу они могут претендовать.”

Именно так часто выглядят первоначальные требования к приложению со стороны заказчиков: малознакомый с программированием заказчик просит разработать программу для выполнения определенной функции.

Это упражнение определенно даст читателю некоторые практические навыки о том, как, начиная с формулировки проблемы, двигаться к созданию законченного продукта. Задача в данном случае заключается в том, чтобы изложить требования заказчика в более четкой форме и использовать возможности РНР для выполнения сбора и обработки информации, а также для выдачи ответа пользователям данного Web-сайта. Чтобы выполнить упражнение, в дополнение к законченной программе следует включить следующие пункты:

- ❑ перечень всей необходимой информации в дополнение к уже представленной;
- ❑ описание Web-страниц, которые вы будете разрабатывать, причины их разработки (зачем эти страницы нужны), и то, как пользователи будут взаимодействовать с ними;
- ❑ краткое описание интеграции созданных страниц с существующим Web-сайтом.

Для этого упражнения нет единого программного решения, написанного на PHP, хотя все решения выполняют подобную обработку данных и используют такие же этапы. Базы данных и файловые системы в книге еще не рассматривались. Пока читатель не владеет способами длительного сохранения резюме, поэтому переданная пользователем информация будет потеряна после завершения обработки.

Задание описано — можно приступать к проектированию.

Совет: чтобы разработать программное решение, попытайтесь сначала описать проблему как последовательность Web-страниц и задать себе вопрос, что пользователь хотел бы увидеть. Вспомните уже описанные средства PHP, которые можно было бы применить для получения информации, а также точно опишите данные, которые должны присутствовать для выполнения следующего этапа. Создать работоспособное решение гораздо проще путем разбиения всего цикла разработки на небольшие этапы.

5

Надежный и понятный код

Очень легко (а иногда необходимо) быстро собрать приложение из различных компонентов, не тратя при этом времени на обеспечение надежности и безопасности кода в будущем. Для крупных, сложных или очень важных приложений стоит использовать формальный процесс разработки программного обеспечения, который предполагает изучение документации и обеспечивает определенные метрики для цикла разработки, включая проектирование, собственно разработку, порядок изменений, поддержку, обновление и завершение.

Вся эта глава посвящена написанию надежного и понятного кода. Разработчикам рекомендуется всегда прилагать дополнительные усилия, чтобы гарантировать, что их код обладает указанными качествами. К тому же знание того, как создавать хороший код, дает программисту значительное преимущество, поскольку разработчик, имеющий практические навыки создания хорошего кода, вероятнее всего, будет создавать высококачественный код быстрее других разработчиков.

Что означают эти качества? Надежную программу трудно вывести из строя даже при неверном обращении с ней. Такая программа способна справиться с разнообразными входными данными. А если по каким-либо причинам программа не может продолжать работу или получить желаемый результат, то она *изящно останавливается*, т.е. позволяет пользователю перейти к следующему логичному варианту и выводит какое-либо выразительное сообщение (и определенно не отображает сообщения об ошибках PHP или SQL). В дополнение к этому понятный код пишется с большим количеством внутренней документации, так чтобы впоследствии, когда потребуется доработать код, создатель этого кода (или его коллега) мог легко вспомнить (или понять) первоначальные намерения. Понятный код это не просто работоспособный код, его структуры имеют определенный смысл и абстрагированы друг от друга настолько, что небольшое изменение одного участка кода вряд ли вызовет ошибки в других участках. Вместе данные качества улучшают программы и облегчают их последующее сопровождение.

В этой главе описаны основы тестирования, устранения неисправностей и отладки, а также использование функций обработки ошибок в PHP5. Кроме того, примеры

кода демонстрируют создание функции обработки строк для проверки данных, введенных пользователем в форму, и использование для этого регулярных выражений, а также новую функциональность try/catch.

Тестирование и отладка

Отладка уже обсуждалась вкратце в главе 1 при рассмотрении темы устранения неисправностей в установке PHP. В данной главе эта тема, а также формальный процесс отладки рассмотрены подробно.

После того как определены основные задачи приложения, начинается процесс создания программы. Можно сотрудничать с дизайнерами, создавая страницы, отображаемые пользователям (это можно делать параллельно или до написания большей части кода), а можно чертить диаграммы или логические блок-схемы разрабатываемой программы. Однако главной задачей остается написание кода.

Обычно программисты пишут код и вставляют его, как уже было показано, в Web-страницы либо генерируют HTML-код для Web-страниц с помощью оператора echo или из объектов. Чаще всего программисты пишут достаточное количество кода для выполнения осмысленного набора функций, а затем вызывают созданные страницы в браузере, чтобы проверить работу созданного кода.

Тестирование является первым этапом отладки. Очевидно, что если программист еще не протестировал свой код и не нашел ошибок, то и отлаживать ему еще нечего. Небольшие быстро создаваемые фрагменты кода для простых динамических функций, таких как отображение на одной странице текущей даты, не требуют формального протокола тестирования; корректные данные либо появляются, либо нет. Однако более сложные полнофункциональные приложения требуют более высокого уровня тестирования и формального процесса определения и контроля найденных ошибок. В данной главе основное внимание уделено только тестированию, необходимому для отладки кода, однако в реальной работающей среде используются гораздо более формализованные системы тестирования.

Значения, нарушающие работу кода

Программисты, впервые создающие код для приложения, склонны предполагать, что обрабатываемые значения будут находиться в пределах ожидаемого диапазона. Однако PHP-приложения предназначены для работы с данными, которые вводят Internet-пользователи, поэтому приложение вместо ожидаемых данных может получить значения, которые умышленно подготовлены для того, чтобы нарушить работу приложения (иначе говоря, взломать его).

Следовательно, тестирование разработанного кода с использованием неожиданных значений является полезной частью любого протокола тестирования. Прежде чем изучать неожиданные значения, которые могут быть переданы приложению во время реальной работы, следует рассмотреть данные, присутствующие в Web-приложениях.

Все данные, передаваемые браузером Web-серверу, форматируются как строки. Например, число 1995 может быть годом или ценой, но при передаче Web-серверу посредством HTTP-запроса оно поступает в PHP-процессор в виде строки. Строка может быть пустой или состоять из любого количества символов. Если программа ожидает число, то все в порядке, поскольку PHP весьма снисходительно относится к типам данных. Однако если программа ожидает строку определенной длины и не получает ее, то может возникнуть проблема.

Даже если полученное значение является численным по своей природе (т.е. полностью состоит из цифр) и соответствующим образом преобразовывается РНР-процессором в число, это число все-таки может быть слишком большим или слишком маленьким (выходит за границы диапазона допустимых значений) и поэтому также может создавать проблемы.

Одним из первых мероприятий, которые необходимо предпринять для защиты приложения, является определение диапазонов значений (строковых, численных, диапазонов дат или значений других типов), а затем тестирование приложения с использованием значений, выходящих за рамки этих диапазонов (а, кроме того, с использованием пустых или нулевых значений).

Могут также возникать и другие проблемы, такие как ввод значений, которые попадают в соответствующий диапазон или имеют необходимый тип данных, но все равно по какой-либо причине непригодны для использования в программе. Например, длина e-mail-адреса может быть ограничена в базе данных до 100 символов; что же произойдет, если пользователь введет адрес, состоящий из 101 символа? И что произойдет, если пользователь забудет вставить символ @ в свой адрес? Передаваемые серверу данные в любом случае остаются строковыми и внешне могут выглядеть как e-mail-адрес, но ошибки такого рода все равно могут создавать проблемы.

Поэтому можно начать тестирование приложения с того, чтобы убедиться, что передаваемая строка действительно форматируется как строка, число, дата или значение другого типа данных, а затем продолжать тестирование, проверяя, попадает ли данное значение в необходимый диапазон. После этого следует выполнить другие тесты, которые гарантируют, что переданная строка имеет правильную длину, а возможно даже содержит некоторые специальные символы, подходящие для того типа данных, в который она будет преобразована. Все эти проверки кажутся весьма трудоемкими, однако реальность такова, что пользователи часто вводят неверные данные и данные, которые кажутся правильными, но не могут быть правильно обработаны (например “три” вместо 3), поэтому дополнительные усилия и затраты времени, необходимые для тестирования приложения, окупятся сторицей.

Основные типы ошибок

По мере написания кода программист часто тестирует не все приложение, а небольшие промежуточные точки проекта. Очень часто первый же тест показывает ошибки, которые РНР может перехватывать программно. Например, синтаксические ошибки возникают, когда программист забыл напечатать точку с запятой или закрывающую фигурную скобку. РНР выдает выразительные сообщения об ошибках, помогая тем самым разработчику обнаружить и исправить ошибки такого типа, и если программист знаком с такими сообщениями, то он без труда сможет избавиться от синтаксических ошибок в коде.

Когда приложение не может найти или открыть какой-либо файл или подключиться к базе данных, возникают ошибки другого типа. Эти ошибки называются *ошибками времени выполнения (runtime errors)*, поскольку возникают во время выполнения программы. РНР может программно обнаруживать и такие ошибки, а также генерировать соответствующие сообщения о них.

Ошибки, которые РНР может перехватить, а также сгенерировать сообщения о них, часто легко найти и исправить, но иногда сообщения об ошибках бывают не совсем понятными или даже загадочными (как в любом языке программирования), если про-

граммист с ними не знаком. Подробнее сообщения об ошибках РНР обсуждаются далее в этой главе.

Существует еще один важный тип ошибок — *логические* ошибки, которые возникают, когда программа не имеет синтаксических ошибок и выполняется успешно, но выдает некорректный результат или ответ. Найти и исправить такие ошибки несложно, но иногда они ставят разработчика в тупик, потому что могут возникать очень редко или только при определенных условиях.

В остальной части данной главы описываются наиболее распространенные синтаксические ошибки, логические ошибки и ошибки времени выполнения, а также способы их устранения.

Отладка РНР-сценария

Предположим, что программист приложил все усилия, чтобы сделать свой код непроницаемым для ошибок, и все-таки ошибки продолжают появляться. Избавиться от таких маленьких вредителей поможет процесс, который называется *локализацией неисправностей* (*troubleshooting*). Локализация неисправностей позволяет постепенно отбрасывать потенциальные причины проблем до тех пор, пока не будет найдена единственно верная причина. После того как причина найдена, программист изменяет код и проблема устраняется. Локализация неисправностей работает как для синтаксических, так и для логических ошибок. Естественно, каждый раз изменив код, программисту следует проверить его снова, чтобы убедиться, что исправления работают, что они привели к более глубокому пониманию истоков проблемы или что они вызвали новые проблемы в других участках кода.

Сообщения об ошибках РНР

Всякий раз, когда РНР-процессор сталкивается с ошибочными условиями, которые влияют на его способность завершить обработку данных, он генерирует сообщение об ошибке. Серьезные ошибки приводят к тому, что РНР-процессор останавливает сценарий, прекращает обработку данных и отображает на экране сообщение о “неисправимой ошибке” (“Fatal Error”, если отображение ошибок было включено в конфигурационном файле РНР). Менее серьезные ошибки могут прерывать только небольшую часть общей обработки данных в РНР и в таком случае возможно появление “предупреждения” (“Warning”) на Web-странице. Могут возникать ошибки, которые не влияют на обработку данных, причины таких ошибок также следует выявить самостоятельно.

Конфигурирование РНР для обработки ошибок

Чтобы контролировать режим работы РНР при возникновении ошибок, в файле `php.ini` можно задать множество параметров. Некоторые из наиболее важных конфигурационных параметров перечислены ниже.

- ❑ `display_errors`: включает или выключает отображение на экране сообщений об ошибках. Необходимо отметить, что если отображение ошибок отключено, то при возникновении серьезной ошибки в Web-браузере пользователя появится пустая страница.
- ❑ `error_reporting`: значение, представляющее уровень отображения ошибок. Например, если данный параметр имеет значение 3, то будут генерироваться сообщения об ошибках двух первых из перечисленных в документации типов.

- ❑ `log_errors`: инструктирует РНР регистрировать ошибки в журнале. Как и в случае других систем протоколирования ошибок, можно определить количество аккумулируемых в журнале сообщений об ошибках.
- ❑ `track_errors`: включение этого параметра позволяет в любом сценарии получить самое последнее сообщение об ошибке (если она произошла) из постоянно доступной встроенной переменной `$php_errmsg`. Эта переменная не является суперглобальной; она доступна только в пределах той области видимости, в которой произошла ошибка.

Типы ошибок в РНР

При всем многообразии ошибок всех их можно сгруппировать в две основные категории: ошибки, которые нарушают работу программы из-за неверного синтаксиса или проблем, возникших во время выполнения, и ошибки, которые приводят к неверным результатам. В пределах первой категории РНР выделяет несколько уровней ошибок.

- ❑ Критическая (или неисправимая) ошибка (Fatal Error): полностью прекращает выполнение сценария. Существуют критические ошибки времени выполнения, времени компиляции и критические синтаксические ошибки.
- ❑ Предупреждение (Warning): указывает на то, что, возможно, возникли ошибочные условия, но сценарий после возникновения ошибки будет продолжать работу.
- ❑ Уведомление (Notice): указывает на то, что произошло нечто, заслуживающее внимания. В эту группу ошибок входят уведомления времени выполнения и пользовательские уведомления.

Если в файле `php.ini` включено отображение ошибок (`display_errors = on`), то сообщения об ошибках будут выводиться на экран, часто это самый быстрый способ поиска и отладки ошибок при создании приложения. Для реальных приложений следует отключать вывод ошибок на экран и вместо этого протолировать их в журнале.

Синтаксические ошибки

Синтаксические ошибки — ошибки при написании кода, такие как опечатки, пропущенные символы или знаки препинания, а также некорректное использование операторов. РНР-интерпретатор не может соответствующим образом проанализировать такой код и поэтому генерирует сообщение об ошибке. Существует несколько широко распространенных причин возникновения синтаксических ошибок, которых можно научиться избегать.

- ❑ Простые опечатки: очень легко напечатать вместо двоеточия или квадратной скобки точку с запятой или фигурную скобку. Приучите себя проверять синтаксис по мере ввода кода.
- ❑ Не правильно закрытые операторы: рекомендуется сразу вставлять все необходимые в операторах скобки, даже если они не являются обязательными для работы кода. Для этого придется выполнять небольшую дополнительную работу, но как только это войдет в привычку, программист будет корректно использовать операторы, не теряя скобок. Например, ниже приведен код, в котором неверно закрыты скобки:

```

for ($myloop01 = 0; $myloop01<10; $myloop01++)
{
  for ($myloop02 = 2; $myloop02<20; $myloop02++)
  {
    for ($myloop03 = 3; $myloop03<30; $myloop03++)
    {
      for ($myloop04 = 4; $myloop04<40; $myloop04++)
      {
        ... Здесь еще какой-либо код...
      }
    }
  }
}

```

- ❑ Не соответствующие отступы в коде: не всегда ясно, что цикл закрыт десяти страницами кода дальше. Отступы в коде определенно облегчают понимание кода как в момент его написания, так и в будущем. Можно отформатировать приведенный выше код так:

```

for ($myloop01 = 0; $myloop01<10; $myloop01++)
{
  for ($myloop02 = 2; $myloop02<20;$myloop02++)
  {
    for ($myloop03 = 3; $myloop03<30; $myloop03++)
    {
      for ($myloop04 = 4; $myloop04<40; $myloop04++)
      {
        ... Здесь еще какой-либо код...
      }
    }
  }
}

```

по отступам легко заметить недостающую скобку.

- ❑ Пропуск точки с запятой при закрытии оператора: иногда отсутствие точки с запятой в конце строки допускается, но пропуск этого знака там, где он должен быть, создает серьезные проблемы. Чтобы выработать привычку завершать операторы точкой с запятой, рекомендуется вставлять ее всегда, даже там, где делать это необязательно.
- ❑ Неправильное написание имени функции: даже простая опечатка в имени функции приводит к ошибке. Необходимо избегать перемещения, пропусков или добавления букв в именах функций. Тщательный ввод и частая проверка синтаксиса позволяют избежать подобных ошибок (не следует полагаться на какие-либо программы проверки правописания для обнаружения ошибок в именах функций).
- ❑ Пропуск закрывающих кавычек в строках или неверное смешивание одинарных и двойных кавычек: очень легко забыть закрыть кавычки в строке или ошибиться в количестве использованных одинарных кавычек, особенно при создании длинных строк с большим количеством вставленных переменных, например:


```
echo "Это начало строки с " . $number . " кавычками, как с одинарными,
например, \' , так и с двойными, например, \", а также кавычек в HTML,
например, ;
```
- ❑ Две первые кавычки (одинарная и двойная) экранированы правильно, а две последние в примере HTML-кода — неправильно. В дополнение к этому пропущена завершающая двойная кавычка. При обработке данного оператора PHP сгенерирует синтаксическую ошибку.

Как уже отмечалось, синтаксические ошибки часто появляются при первом тестировании кода. Хотя PHP предоставляет весьма выразительные сообщения об ошибках,

программист при отладке может столкнуться с одной трудностью: сложно найти точку возникновения ошибки. Например, предположим, что имеется следующий код:

```
<?php
//генерирование синтаксической ошибки
$line_of_code = "ошибка: нет точки с запятой"
?>
```

В этом коде есть синтаксическая ошибка или ошибка синтаксического разбора. Из-за ошибки (пропуск точки с запятой) РНР-процессор не может полностью проанализировать данный код, поэтому код не будет выполнен. Вместо этого генерируется следующее сообщение об ошибке:

```
Parse error: parse error in /home/servata/www/error_test_file.php on line 3
(Синтаксическая ошибка: синтаксическая ошибка в файле
/home/servata/www/error_test_file.php в строке 3)
```

Это сообщение содержит описание ошибки и номер строки с ошибкой. Однако иногда номер может неверно указывать ошибочную строку, например, когда пропущена фигурная скобка, как показано в следующем примере:

```
<?php
//генерирование синтаксической ошибки
if ($my_test == 0) {

    //какие-либо операторы
} else {
?>
<html>
<head>
<title></title>
</head>
<body>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
какой-либо HTML-код<br>
</body>
</html>
```

При разборе данного кода РНР-интерпретатор генерирует следующее сообщение:

```
Parse error: parse error in /home/servata/www/error_test_file02.php on line 31
(Синтаксическая ошибка: синтаксическая ошибка в файле
/home/servata/www/error_test_file02.php в строке 31)
```

Каким бы информативным ни было сообщение, в данном конкретном случае номер строки указан неверно, потому что код состоит только из 29 строк. Даже если посмотреть в конец файла, то можно не заметить, что проблема заключается в пропущенной фигурной скобке, так как РНР-блок правильно закрыт до начала HTML-кода.

В большинстве случаев РНР-интерпретатор точно или очень близко указывает строку с ошибкой, однако так бывает не всегда. Поэтому прежде всего следует очень

внимательно писать код — практический опыт позволяет программисту избегать большинства синтаксических ошибок. Чтобы избавиться от ошибки, достаточно переписать код и проверить его снова.

Логические ошибки

Логические ошибки происходят, когда код выполняется, но не генерирует правильный результат. Например, если в программе вместо предполагаемого суммирования двух чисел выполняется вычитание, то код все равно работает хорошо, но результат будет неправильным.

Часто бывает трудно обнаружить логические ошибки, особенно когда неверный ответ генерируется только при определенных обстоятельствах. Если получение правильного ответа критично (как, например, при проведении банковских транзакций), то следует применить более сложную и всеобъемлющую программу тестирования, которая предназначена специально для проверки всех возможных вариантов входных данных. А отчеты об ошибках должны содержать подробные данные об условиях, в которых возникает каждая ошибка.

Логические ошибки возникают нерегулярно. Например, программа запускается и либо начинает возвращать данные, которые могут быть неправильными, либо не возвращает никаких данных вообще. В некоторых случаях программа вообще ничего не выводит на экран. Бывает так, что программа работает хорошо, но при передаче ее конечному пользователю она вдруг начинает работать неправильно. Все эти ситуации обычно являются результатом ошибок в логике программы.

Следует представлять себе различия между типами логических ошибок. Некоторые логические ошибки проявляются во время выполнения программы и только при определенных условиях. Например, РНР-код выглядит хорошо и даже принимается и выполняется РНР-процессором, но при выполнении программы возникают ошибки, в результате которых программа преждевременно завершается. Такие ошибки называются *ошибками времени выполнения*. Ошибки времени выполнения, преждевременно останавливающие программу, называются *критическими (fatal)*. Все без исключения синтаксические ошибки являются критическими, но ошибки времени выполнения, останавливающие сценарий только при определенных обстоятельствах, не всегда можно назвать критическими.

Логические ошибки второго типа фактически не останавливают программу, и она выполняется до завершения, но в конце возвращает некорректную информацию или вообще не возвращает никаких данных. Такие ошибки кратко можно охарактеризовать как неожиданные ошибки вывода. Рассмотрим ошибки времени выполнения более подробно.

Ошибки времени выполнения

От синтаксических ошибок ошибки времени выполнения отличаются тем, что они возникают, когда РНР-процессор уже успешно проанализировал код и выполняет его. Ошибки времени выполнения часто возникают из-за неудачного подключения к внешнему ресурсу, такому как файл или база данных, но иногда такие ошибки тривиальны (хотя их все равно трудно обнаружить). Например, РНР генерирует ошибку,

если сценарий пытается разделить какое-либо число на нуль. Такие типы ошибок обсуждаются в последующих разделах, а файлы и подключения к базам данных описываются в следующих главах.

Деление на нуль

С математической точки зрения не существует приемлемого способа деления на нуль (невозможно вывести корректный ответ), поэтому случайное деление числа на нуль вынуждает РНР генерировать ошибку. Вряд ли потребовалось бы умышленно создавать приложение, которое пыталось бы делить числа на нуль (кроме, вероятно, примеров программ, которые не будут работать), поэтому ошибки данного типа обычно возникают непреднамеренно. Так как для хранения и обработки значений используются переменные, и пользователи иногда вводят неверные значения, то можно заметить, что деление на нуль на практике возникает довольно часто.

Предположим, что имеется форма, в которой пользователю предлагается ввести несколько значений (например, количество приобретаемых товаров), а затем введенные числа используются в каких-либо вычислениях (например, для расчета скидки).

Если пользователь случайно сделает ошибку или забудет ввести одно из значений (или вообще не совсем понимает, что ему следует делать), то все может закончиться тем, что программа попытается выполнить деление на несуществующее число (т.е. на нуль) и РНР-процессор сгенерирует ошибку.

Ниже приведен код очень простой формы, который показывает, что происходит, когда в сценарии выполняется деление на нуль. Запустите HTML-редактор, введите следующий код и сохраните созданный файл как `divide_by_zero.php`.

```
<html>
<head><title></title></head>
<body>
<b>Деление любого числа (кроме нуля)</b>
<br>
<br>
<?php
if (isset($_POST['posted'])) {

    $first_number = $_POST['first_number'];
    $second_number = $_POST['second_number'];
    $answer = $first_number / $second_number;
    echo "Ответ: " . $answer . "<br>";

}
?>

<form method="POST" action="divide_by_zero.php">
<input type="hidden" name="posted" value="true">
Пожалуйста, введите первое число:
<br>
<br>
<input name="first_number" type="text" value="0"><br>
Пожалуйста, введите второе число:
<br>
<br>
<input name="second_number" type="text" value="0"><br>
Нажмите кнопку "Разделить"
<br>
<br>
<input type="submit" value="Разделить">
</form>
</body>
</html>
```

При выполнении данного кода в браузере отображается страница, показанная на рис. 5.1.

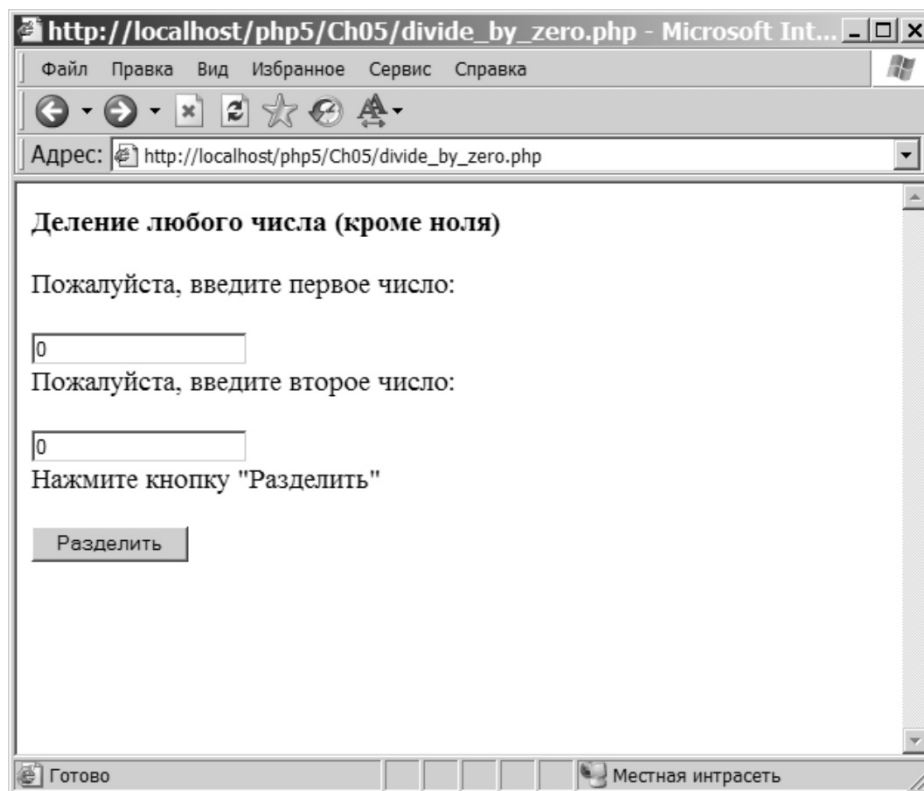


Рис. 5.1.

Протестируйте форму как с числами, так и без. На рис. 5.2 показан результат попытки деления на ноль.

Данный пример весьма прост, синтаксические ошибки не возникают, если пользователь вводит соответствующие значения. Во время разработки программисты в большинстве случаев склонны использовать подходящие значения, они легко могут упустить из виду ошибки такого типа в ходе тестирования и проверки приложения. Об этом следует помнить.

Если пользователь вводит в данной форме (или оставляет) 0 в качестве первого значения, то никакой ошибки не возникает (ответ равен нулю). Однако даже если второе число вообще не введено, PHP все равно интерпретирует пустую строку как 0 и генерирует сообщение об ошибке. Одна из возможностей уменьшить вероятность возникновения ошибки заключается в том, чтобы установить в HTML-коде значения по умолчанию для полей первого и второго чисел равным 1, а не 0. Кроме того, можно проверять входящие значения, чтобы убедиться, что они не равны нулю. (Проверка данных форм рассматривается в этой главе далее.)

Итак, в приложениях необходимо учитывать возможность деления на ноль, а также другие математические функции, которые могут создавать проблемы, например, вычисление квадратного корня отрицательного числа.

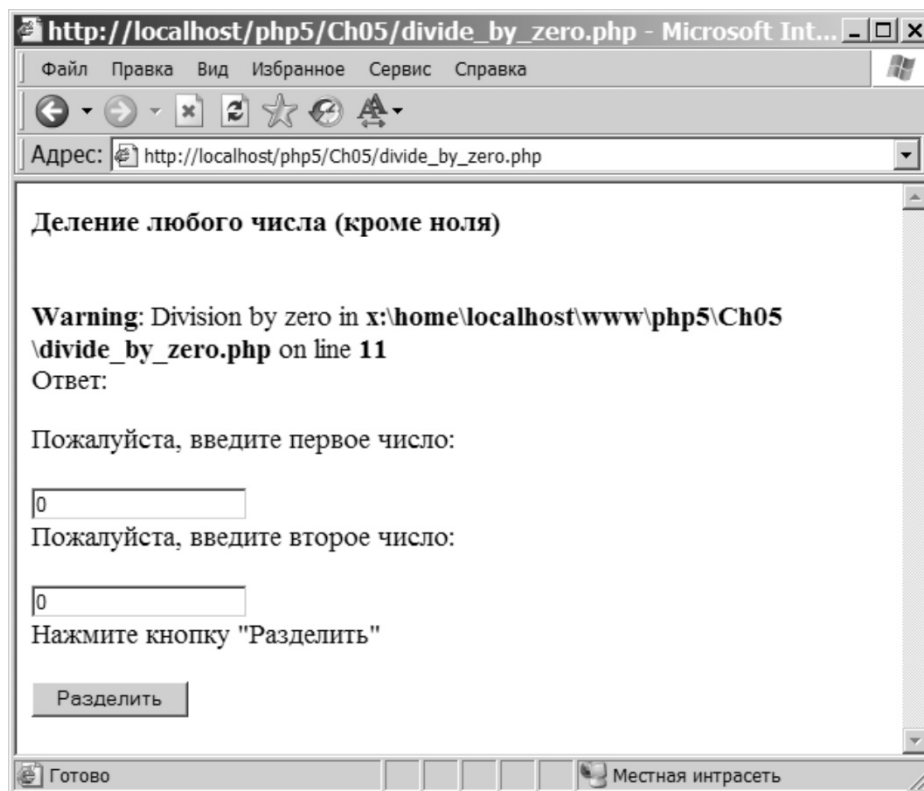


Рис. 5.2.

Бесконечные циклы

Структуры циклов являются одними из наиболее важных конструкций в любом языке программирования, а также основой для одной из наиболее распространенных ошибок в программировании — бесконечных циклов. Их мощь, по сути, делает их восприимчивыми к таким ошибкам, потому что они предназначены для циклического выполнения вплоть до возникновения определенного условия. Если это условие никогда не возникает, то цикл никогда не закончится.

При использовании циклических структур, таких как `while` и `do while`, в PHP возникает проблема: если заданное условие никогда не выполняется, то PHP обрабатывает цикл потенциально бесконечное число раз в пределах, заданных параметром `max_execution_time` (в файле `php.ini`). Если время выполнения сценария превышает значение `max_execution_time`, то PHP генерирует предупреждение и прекращает обработку. Как и любое сообщение об ошибке, это предупреждение может быть отображено и ясно сообщать о случившемся, однако восстановить работу программы при таких обстоятельствах не удастся. Рассмотрим следующий код, не запуская его:

```
$cntr=1;
$test = True;
while ($test)
{
    $cntr++;
}
```


Пока переменная-счетчик (`$cnttr`) увеличивается в каждой итерации цикла, переменная `$test` не может получить значение `False`, поэтому данный цикл никогда не закончится. Запуск такого цикла на Windows-сервере может легко привести к зависанию системы (Linux-машины с Web-сервером Apache справляются с такой ситуацией гораздо лучше, и все же когда время выполнения превысит значение `max_execution_time`, отображается предупреждение).

Чтобы исправить ошибку такого типа (или гарантировать, что она никогда не возникнет), необходимо удостовериться, что, каким бы ни было условие работы цикла, в цикле должен быть способ получить при проверке этого условия значение `False` (или любое другое значение, необходимое для завершения цикла).

Логические ошибки вывода

Приложение, не имеющее синтаксических ошибок и никогда не генерирующее предупреждений об ошибках времени выполнения, может, тем не менее, выдавать логические ошибки вывода. По существу, это означает, что программа выполняется корректно, но программист не обеспечил правильную обработку данных или не сумел предугадать, что во время обработки данных может возникнуть определенное значение. Данные выводятся корректно, просто среди них нет ожидаемого значения.

Очень хорошим примером таких ошибок является использование десятичных дробей. Например, предположим, что пользователь должен ввести значение 0,0775 (налог на продажу в Калифорнии), но вместо этого вводит число 7,75. Можно предположить, как это повлияет на общую сумму налогов. Вместе с тем, ничто в данном сценарии не вызывает автоматический вывод предупреждения или сообщения об ошибке; PHP просто выполнит вычисления и выдаст ответ. Предвидеть такие ситуации и предотвратить возникновение подобных ошибок должен программист, и только тщательное многократное тестирование поможет создать качественное приложение.

Функции, не возвращающие значений

При создании собственных функций (эта тема раскрывается в главе 6) можно включать оператор `return`, который инструктирует функцию о том, какое значение необходимо вернуть при вызове данной функции. Не все функции требуют наличия данного оператора, поэтому о нем довольно легко забыть. А если оператор `return` не используется там, где он требуется, то функция вообще не будет возвращать никакого значения.

Если после этого применить данную функцию и использовать ожидаемый от нее результат в программе, то можно потратить много времени, прежде чем станет ясно, что функция ничего не возвращает. Обнаружение таких ошибок может оказаться весьма утомительным, поэтому всегда следует, как минимум, один раз протестировать функцию и убедиться, что она возвращает ожидаемый результат.

Неправильный порядок аргументов функций

Часто для выполнения обработки необходимо передавать функциям некоторые аргументы. Функции ожидают определенный порядок ввода аргументов и не способны определить нарушение этого порядка. Поэтому если функция для выполнения вычислений должна получить два численных значения, но порядок этих аргументов нарушен, то может возникнуть проблема. Функция возвратит результат, но он вряд ли будет верным.

Присвоение значений вместо сравнения значений

Одной из широко распространенных ошибок в PHP является использование знака равенства для сравнения значений, когда фактически вместо этого происходит присваивание значений. PHP в таких случаях не выводит никаких предупреждений, а просто присваивает переменной значение и продолжает работу сценария с новым значением. Эта ошибка очень распространена в структурах `if... then... else`, например:

```
If ($my_value = $your_value) {  
    //выполнить этот блок  
}  
else {  
    //выполнить этот блок  
}
```

К сожалению, ошибочность подобного кода не всегда очевидна, и поскольку код кажется верным (в конце концов, можно просто не заметить, что используется `=`, а не `==`), особенно если программист знает другие языки, которые допускают использование знака `=` для сравнения значений, то избежать возникновения такой ошибки бывает нелегко. Не существует волшебного средства для разрешения данной проблемы; необходимо просто быть внимательным: использовать знак `=` для присвоения значений, а `==` для сравнения.

Отладка и обработка ошибок в PHP5

В PHP5 появились новые функции `try/catch`, которые предоставляют новые ценные способы обработки ошибок в PHP (эти функции подробнее рассматриваются в конце данной главы), кроме того, PHP также предлагает более традиционные способы отладки и поиска ошибок, а также возможности их изящной обработки во время выполнения приложения. В данном разделе рассматриваются традиционные средства.

Предотвращение отображения конфиденциальной информации

Необходимо принимать во внимание, что в некоторых сообщениях об ошибках не только показывается путь и имя файла, в котором произошла ошибка, но и номер строки с ошибкой. Этого может быть достаточно опытному программисту или хакеру, чтобы взломать приложение. Никто кроме разработчика приложения не должен знать пути и имена файлов, особенно тех, в которых имеются сценарии, подключаемые файлы и т.д. PHP-сообщения об ошибках могут быть весьма многословными, и все это изобилие информации становится вредным, когда сообщает пользователю не только о невозможности открыть файл, но и указывает точный путь, который сценарий пытался открыть. После завершения разработки и до копирования приложения на реальный сервер необходимо отключить отображение ошибок.

Кроме того, не следует забывать о том, что другие приложения, которые используются PHP, могут сообщать еще больше информации об ошибках; PostgreSQL и другие базы данных могут очень точно описывать SQL-выражения, содержащие ошибки.

Создание собственных инструментов для отладки

Для написания HTML- и PHP-кода можно использовать самые простые текстовые редакторы, но в этом случае разработчику будут недоступны развитые средства для предварительной компиляции кода, обнаружения синтаксических ошибок до выполнения программы или отслеживания значений переменных во время ее выполнения. Однако не все потеряно, потому что существует несколько простых и эффективных способов сделать все это самостоятельно.

Использование оператора echo()

С помощью оператора `echo` можно выводить на экран значение, присвоенное какой-либо переменной. Это можно делать внутри функций, внутри циклов, во время обычной обработки — по сути, можно помещать оператор `echo` почти в любой участок кода. А если идентифицировать местоположения того или иного оператора `echo` (например, вставив в него текст вроде “внутри цикла `for`” или “внутри второго оператора `if`”), то в коде можно использовать множество операторов `echo` и знать, что происходит в каждый момент выполнения программы. Например, если есть сценарий, принимающий от пользователя имя и адрес электронной почты, а затем обрабатывающий эти данные, то все происходящее во время обработки можно выяснить с помощью подобного кода:

```
<?php
echo "Имя : " . $name . "<br>";
echo "Email:" . $email . "<br>";
// обработка
echo "Имя после обработки:" . $name . "<br>";
echo "Email после обработки:" . $email . "<br>";
// дополнительная обработка
echo "Имя после дополнительной обработки:" . $name . "<br>";
echo "Email после дополнительной обработки:" . $email . "<br>";
?>
```

Имя и адрес могут изменяться во время каждого цикла обработки, но эти изменения можно будет отличить друг от друга, поскольку они сопровождаются различным поясняющим текстом.

Ошибки внутри HTML-кода

Если PHP генерирует ошибку и отправляет предупреждение или сообщение о критической ошибке с каким-либо HTML-кодом, обработка которого была завершена до возникновения ошибки, то сообщение не всегда отображается на экране. Если ошибка возникает внутри HTML-тега или внутри генерируемого элемента управления формы, браузеры могут быть настолько снисходительными к HTML-ошибкам, что страница может отображаться даже несмотря на то, что на ней отсутствуют ожидаемые данные. Очень важно всякий раз при неверной работе сценария проверять HTML-код, получаемый браузером на предмет скрытых PHP-ошибок, а в ходе такой проверки стоит потратить время и убедиться, что ожидаемые значения (которые могут быть скрыты внутри исходного HTML-кода) также присутствуют.

Предположим, например, что используется PHP-код для подключения к базе данных, получения некоторых записей и последующего использования этих записей в цикле `for` для создания HTML-элемента `<select>`. Известно, что элемент `<select>` необходимо завершать закрывающим тегом `</select>` и что между этими тегами присутствуют элементы `<option>`. Если один из элементов `<option>` записан неверно (например, в нем присутствует PHP-сообщение об ошибке), то браузер просто проигнорирует данный элемент. На экране пользователь увидит выпадающий список и все “хорошие” элементы `<option>`, но в HTML-коде можно будет заметить что-нибудь подобное:

```
<select name="select_customer_id">
  <option value="1">Джон Доу</option>
  PHP-сообщение об ошибке
  <option value="3">Джейн Доу</option>
</select>
```

В данном случае полностью теряется клиент номер 2, но на экране сообщение об ошибке не отображается, хотя оно и отчетливо видно в исходном HTML-коде.

Проверка данных форм

Следует всегда помнить принцип: пользователи могут и будут вводить в HTML-формы любые данные и/или не вводить их вовсе независимо от того, как хорошо сделана форма и что сценарий ожидает получить от пользователя. Более того, злонамеренные пользователи будут умышленно вводить необычные данные, чтобы взломать приложение. Главная защита от таких проблем — проверка информации форм как на клиентской, так и на серверной стороне.

Один из способов сделать это заключается в ограничении допустимых значений для определенного текстового поля. Например, в рассмотренном ранее сценарии `loan.php` форма передает возраст пользователя в переменную `$age`, а PHP-код затем сверяет это значение с более реальным диапазоном возрастов:

```
if ($age < 1 or $age > 130)
{
    echo "Введен некорректный возраст. Пожалуйста, введите возраст в пределах 1 и 130.";
    break;
}
```

Конечно, ничто не запрещает программисту пойти дальше простого информирования пользователя о вводе несоответствующих данных. Можно предпринять и другие действия в случае обнаружения неподходящих значений.

Использование оператора `exit`

Проверка данных — хорошая практика, но что делать, если программа столкнулась с некорректными данными? Иногда необходимо прервать обработку, что и делает оператор `exit`. В отличие от оператора `break`, который прерывает текущую структуру, оператор `exit` останавливает всю работу программы. Можно использовать любой метод для окончания обработки, но оператор `exit` более жесткий, поэтому при изменять его следует осторожно. После того как PHP-интерпретатор встречает оператор `exit`, ни HTML-, ни PHP-код не обрабатывается, и если программист был не особенно внимательным, то пользователь может увидеть неожиданный результат, например, частично сформированную страницу. Можно переработать приложение для обработки заявок на получение ссуд и сделать его более устойчивым по отношению к пользовательским ошибкам путем внедрения некоторой логики проверки HTML-формы.

Практика Проверка форм

1. Откройте файл `loan.php` (из главы 4) и сохраните его как `loan_fv.php`, а затем внесите следующие изменения:

```
<html>
<head><title></title></head>
<body>
<b>Заявка на получение ссуды. Кредитный банк Namllu</b>

<?php
    if (isset($_POST['posted'])) {

        $age = $_POST['age'];
        $first_name = $_POST['first_name'];
        $last_name = $_POST['last_name'];
        $address = $_POST['address'];
        $loan = $_POST['loan'];
        $month = $_POST['month'];
```

```

//проверка данных формы
if ($age < 10 OR $age > 130)
{
    echo "Введен некорректный возраст - нажмите кнопку Назад и заполните
        форму еще раз";
    exit;
}
if ($first_name == "" or $last_name == "")
{
    echo "Необходимо ввести имя - нажмите кнопку Назад и заполните
        форму еще раз";
    exit;
}
if ($address == "")
{
    echo "Необходимо ввести адрес - нажмите кнопку Назад и заполните
        форму еще раз";
    exit;
}
if ($loan != 1000 and $loan != 5000 and $loan != 10000)
{
    echo "Необходимо ввести сумму ссуды - нажмите кнопку Назад и заполните
        форму еще раз";
    exit;
}

$duration = 0;
switch ($loan) {
case "1000";
    $interest = 5;
    break;
case "5000";
    $interest = 6.5;
    break;
case "10000";
    $interest = 8;
    break;
default:
    echo "Вы не выбрали вариант ссуды<hr>";
    exit;
}
while ($loan > 0)
{
    $duration = $duration + 1;
    $monthly = $month - ($loan*$interest/100);
    if ($monthly <= 0)
    {
        echo "Чтобы погасить ссуду, требуются более крупные ежемесячные
            платежи<hr>";
        exit;
    }
    $loan = $loan - $monthly;
}
echo "Для погашения ссуды при процентной ставке $interest процентов
    понадобится $duration месяцев.<hr>";
}
?>
<form method="POST" action="loan_fv.php">
<input type="hidden" name="posted" value="true">
<br>

```

```

Имя:

Фамилия:

Возраст:

<br>
<br>
Адрес:
<textarea name="address" rows="4" cols="40">
</textarea>
<br>
<br>
Каков размер Вашей текущей заработной платы?
<select name="salary">
<option value=0>До $10000</option>
<option value=10000>От $10000 до $25000</option>
<option value=25000>От $25000 до $50000</option>
<option value=50000>Свыше $50000</option>
</select>
<br>
<br>
Выберите необходимую сумму ссуды<br><br>

<br>
<br>

</form>
</body>
</html>

```

2. Сохраните данный файл как `loan_fv.php` и закройте его.
3. Откройте файл в браузере и введите несколько значений, которые выходят за рамки допустимых (иначе говоря, некорректные значения). На рис. 5.3 показано одно из сообщений об ошибке, которое может получить пользователь.

Как это работает

Пользователь может ввести чужой адрес или указать другой возраст, и в PHP нет способа проверить корректность данной информации. Но что действительно можно сделать с помощью нового кода, это гарантировать, что пользователь не забыл ввести необходимые сведения и что он преднамеренно не ввел очевидно неверную информацию о своем возрасте. Для этого используются операторы `if...then...else`. Первый оператор проверяет, попадает ли введенное пользователем значение в диапазон 10–130, ведь в противном случае совершенно ясно, что пользователь лжет:

```

if ($age < 10 OR $age > 130)
{
    echo "Введен некорректный возраст - нажмите кнопку Назад и заполните форму еще раз";
    exit;
}

```

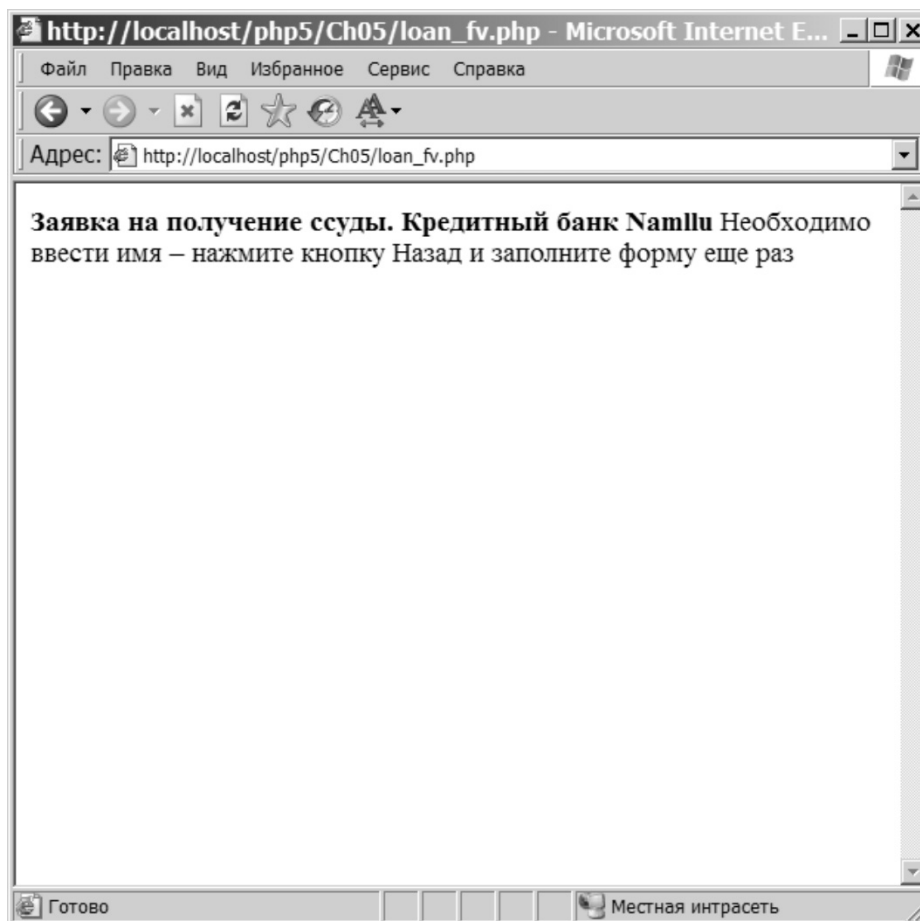


Рис. 5.3.

Если необходимое условие не соблюдается, то сценарий отображает собственное сообщение об ошибке и завершает работу. Если условие выполняется, то больше ничего делать не требуется.

Второй оператор `if` проверяет присутствие имени и фамилии. Две двойные кавычки `" "` означают пустую строку (строку без символов), и проверка выполняется так:

```
if ($first_name == "" or $last_name == "")
{
    echo "Необходимо ввести имя – нажмите кнопку Назад и заполните форму еще раз";
    exit;
}
```

Так же проверяется значение переменной `$address`:

```
if ($address == "")
{
    echo "Необходимо ввести адрес – нажмите кнопку Назад и заполните форму еще раз";
    exit;
}
```

А затем проверяются значения переключателей с тем, чтобы определить, какой из них был выбран:

```
if ($loan != 1000 and $loan != 5000 and $loan != 10000)
{
    echo "Необходимо ввести сумму ссуды – нажмите кнопку Назад и заполните форму еще раз";
    exit;
}
```

Если переменная \$load не равна одному из данных значений, то, очевидно, что пользователь значения не выбрал.

Предотвращение ввода пользователем HTML-кода: функция htmlspecialchars()

Пользователи могут нарушить работу приложения еще одним способом — путем непосредственного включения в поля формы HTML-кода как части вводимых ими данных. Это возможно потому, что HTML-код представляет собой обычный текст, и когда он отправляется назад клиенту, то обрабатывается браузером как любые другие простые текстовые символы, из которых формируется HTML-код. Например, если создать приложение, принимающее от пользователя данные для вывода содержимого каталога, то знающий пользователь может подставить тег перед своим именем и после имени, так чтобы его имя в листинге каталога выводилось жирным шрифтом. Это хороший прием, но он непорядочен по отношению к другим пользователям.

И хотя в данной ситуации такая возможность не является серьезной проблемой, в других обстоятельствах она могла бы быть использована для взлома HTML-кода или иного искажения намерений разработчика. К счастью, в PHP имеется функция htmlspecialchars(), которая заменяет HTML-теги специальными символами (эта тема подробнее описывается далее). В качестве параметра этой функции необходимо передать только строку:

```
$String = htmlspecialchars("<b>После обработки этот текст будет выведен  
обычным, а не жирным шрифтом</b>");
```

Функции также можно передать имя переменной:

```
$String = "<b>После обработки этот текст будет выведен обычным, а не жирным  
шрифтом</b>";  
$String = htmlspecialchars($String);
```

Функция htmlspecialchars() преобразует все HTML-теги в так называемые *специальные символы*. Специальными символами в HTML являются последовательности символов, представляющие HTML-символы, из которых они были получены. Например, тег преобразуется в символы < (знак “меньше”), букву b и символы > (знак “больше”). Когда браузер получает специальные символы, он отображает их на экране как представленные ими HTML-символы, не обрабатывая и не выводя их как обычные HTML-теги.

Эта возможность часто используется, когда необходимо создать Web-страницу, на которой описываются HTML-теги (когда требуется отображать теги простым текстом и не давать браузеру возможность их обработать), кроме этого, данная функция полезна, поскольку предотвращает ввод пользователями их собственного HTML-кода в PHP-приложение.

В конечном итоге, степень проверки ошибок ограничивается только знаниями и фантазией разработчика. Фактически можно было бы предусматривать заранее значения всех переменных, но зачем тогда выводить форму? Можно все делать авто-

матически и заранее. Действительно, мышление с точки зрения пользователя часто реально помогает разработчику. Отношение количества хакеров к количеству пользователей весьма незначительно, т.е. большинство пользователей, нарушающих работу приложения, делают это неумышленно. Они либо не понимают, какие значения нужно вводить, либо просто ошибаются. Поэтому попытайтесь поставить себя на место пользователя, испытывайте приложения на членах семьи или друзьях и старайтесь предугадать все возможные ответы, рассчитывая их как можно полнее. Эта дополнительная работа несомненно окупится.

Проверка строк и регулярные выражения

Все данные, которые получает приложение от пользователя, форматируются в виде строк, поэтому многообразие РНР-функций для работы со строками оказывается весьма полезным для проверки введенных пользователем данных или переформатирования строковых данных в последовательности символов, подходящие для других типов данных. Однако в РНР имеются также и другие функции, которые называются *функциями регулярных выражений* и являются гораздо более мощными, когда речь идет о манипулировании данными. Обе эти темы рассматриваются в нескольких последующих разделах.

Проверка строк

Чтобы осуществить базовую проверку вводимых пользователями данных, можно различными путями использовать РНР-функции для манипуляции строками. В этом разделе рассматриваются несколько таких функций, но следует помнить о том, что, определенным образом комбинируя описанные или не описанные здесь РНР-функции, всегда можно создать собственную функцию. В главе 2, а также в отдельных примерах уже рассматривались некоторые из строковых функций (например, `strlen()` и `substr()`), но ничто не мешает применить их снова в контексте проверки данных.

Использование функции `strlen()`

Некоторые строковые данные имеют определенную длину, например, почтовые коды США, которые всегда состоят из 5 цифр, или в случае ZIP+4 из 5 цифр, дефиса и четырех или более цифр. Поэтому один из способов проверки данных, введенных как почтовые коды, заключается в использовании функции `strlen()`:

```
If (strlen($postal_code) == 5 or strlen($postal_code) == 10) {
    //проверить, находится ли дефис в шестой позиции
    //другие операторы
} else {
    //отправка сообщения об ошибке
}
```

Использование функции `strstr()`

В предыдущем примере необходимо было проверить, действительно ли символ в шестой позиции является дефисом. Для такой проверки удобно использовать функцию `strstr()`, потому что, как можно догадаться по ее имени, она ищет одну строку внутри другой. В данном случае требуется, чтобы функция искала строку, состоящую из одного символа — дефиса, например:

```
if (strlen($postal_code) == 5
    or strlen($postal_code) == 10) {
    if (strlen($postal_code) == 10) {
        if (strstr($postal_code, "-")) {
            //другие операторы
        }
    }
}
```

```
    }  
  }  
} else {  
    //отправка сообщения об ошибке  
}
```

Кроме того, функцию `strstr()` можно использовать для проверки, не является ли шестой символ пробелом, на случай если пользователь вместо дефиса поставил пробел.

Использование функции `substr()`

Продолжая тот же пример, предположим, что необходимо отделить часть +4 от строки ZIP+4. Известно, что эта часть начинается с седьмой позиции и должна быть длиной четыре символа. С этой целью можно задействовать функцию `substr()`. Данная функция принимает в качестве аргументов строку, целое число, представляющее позицию, с которой следует начать поиск, и необязательное целое число возвращаемых символов, например:

```
If (strlen($postal_code) == 5 or strlen($postal_code) == 10) {  
    If (strlen($postal_code) == 10) {  
        $plus4_portion = substr($postal_code, 7, 4)  
    }  
} else {  
    //отправка сообщения об ошибке  
}
```

Использование функций `addslashes()` и `stripslashes()`

Для приложений, позволяющих пользователю вводить информацию, которая может затем передаваться в базу данных, полезно использовать функцию `addslashes()`. Эта функция добавляет обратную косую черту всякий раз, когда встречается строковый символ, который может создать проблему в базе данных (символы `'`, `"`, `\` и `NULL`). Позднее, когда данные будут выводиться пользователю, необходимо применять функцию `stripslashes()`, которая, как можно догадаться, удаляет обратные косые, вставленные функцией `addslashes()`.

Зачем нужны функции, предназначенные для защиты записей баз данных? Тот, кто когда-либо составлял SQL-строку для вставки записи (или для любой другой функции базы данных), знает, что SQL не терпит неверного использования апострофов (SQL рассматривается в главах 9–11).

Предположим, что SQL-строка выглядит так:

```
$query = "INSERT INTO clients (username) values('$username')";  
mysql_query($query);
```

Если пользователь вводит строку `joe'blow` в качестве имени пользователя, то данный запрос работает корректно, однако если пользователь вводит строку `joe'blow`, то запрос не будет выполнен, потому что база данных “увидит” следующий запрос:

```
$query = "INSERT INTO clients (username) values('joe'blow')";
```

Это не совсем очевидно, но если рассмотреть данную строку внимательно, то станет ясно, что апостроф в имени пользователя выглядит для базы данных как нарушение набора разделителей. Как можно гарантировать, что этого не случится? Достаточно применить к введенному пользователем значению функцию `addslashes()` и сохраняемое в базе данных имя пользователя будет корректным. Для этого подойдет следующий код:

```
$username = addslashes($username);  
$query = "INSERT INTO clients (username) values('$username')";  
mysql_query($query);
```

Во время выполнения данного кода функция `addslashes` превратит строку `joe'blow` в строку `joe\'blow`. Обратная косая черта экранирует апостроф и, таким образом, заставляет базу данных принять оба символа (комбинацию обратной косой черты и апострофа) без сбоя.

Однако при выводе имени пользователя следует позаботиться об использовании функции `stripslashes()`. В противном случае пользователь увидит в качестве своего имени строку `joe\'blow`, и если он не удалит обратную косую черту, то следующий запрос запишет в базу данных строку `joe\\\`blow`, так как функция `addslashes` попытается экранировать оба символа обратной косой черты, а это может привести к проблемам.

Кроме того, необходимо убедиться, что функция `addslashes` применяется везде, где используется имя пользователя. Хотя на экране имя пользователя отображается корректно как `joe'blow`, в базе данных оно хранится как `joe\'blow`, таким образом, только применяя функцию `addslashes()` снова, можно быть уверенным, что данное значение будет корректно сравниваться со значением в базе данных.

Регулярные выражения

Поиск заданной строки внутри другой строки в некоторых ситуациях весьма полезен и, конечно, если искомая строка известна, использовать функцию `substr()` очень удобно. Однако предположим, что точная искомая строка неизвестна. Зная об этой строке совсем немного, ее можно найти, используя функции регулярных выражений.

Предположим, например, что необходимо найти строку, полностью состоящую из алфавитных символов и не имеющую цифр. В таком случае, как минимум, известен *образец (pattern)* поиска, и этого для начала достаточно. Простейший образец представляет собой слово или один символ, как в примере с функцией `strstr()`, когда требуется найти дефис:

```
if (strstr($postal_code, "-")) {
```

А для разделения значений в строке (например, последовательности значений, отделенных друг от друга запятыми) можно использовать функцию `explode()`, которая разделяет строку по заданным символам и помещает результаты в массив. Эта функция принимает два аргумента: символ (или строку), по которой следует разделять строку, заданную вторым аргументом, и возвращает массив с результатами разделения. Для того чтобы проверить, имеется ли в строке определенное слово, можно использовать функцию `explode()` с простым оператором `if`, как показано в следующем примере.

```
<?php
$words = "you, should, vote, happily";
$wordarray = explode(",", $words);
foreach ($wordarray as $word) {
    if ($word == "vote") {
        echo "Найдена строка 'vote'";
    }
}
?>
```

Хотя для того чтобы найти определенный символ внутри более крупной строки, можно использовать какую-либо простую РНР-функцию, например, `strstr()`, или функцию `explode()` для более сложных операций сопоставления, когда требуется искать более сложные образцы. В этом случае очень удобно использовать регулярные выражения.

Регулярные выражения (regular expressions), или *regexps*, очень напоминают мини-язык программирования для создания чрезвычайно эффективных образцов поиска. В ре-

гулярных выражениях для формирования образцов поиска значений (или частей значений) используется специальная нотация. В контексте регулярных выражений некоторые символы имеют специальное значение, позволяющее расширять или сужать круг поиска подстрок. Некоторые регулярные выражения позволяют искать символы, входящие в определенную группу; другие позволяют искать символы, повторяющиеся определенное количество раз. Регулярные выражения непременно следуют определенным правилам синтаксиса, которые в общих чертах будут описаны ниже.

Применение регулярных выражений не ограничивается PHP. Такие языки программирования, как Perl и Python, наряду с Unix-утилитами sed и egrep используют ту же нотацию для поиска образцов в тексте. PHP-функции регулярных выражений, в которых используется Perl-нотация, называются PCRE-функциями и их имена начинаются с preg (Perl Regular Expression — регулярные выражения языка Perl), а обычные PHP-функции регулярных выражений называются POSIX-совместимыми функциями регулярных выражений. Обычные (POSIX-совместимые) PHP-функции регулярных выражений не следует использовать для обработки двоичных данных (они искажают двоичные данные); вместо них следует применять PCRE-функции.

Для начала рассмотрим сравнение с образцом с помощью некоторых обычных PHP-функций регулярных выражений, например, с помощью функции `ereg()`.

Использование функции `ereg()`

Приведенный выше пример работает, но он громоздкий, сложный и жестко закодирован (слово “vote” фактически является частью кода, а не поступает как входное значение; по сути, весь массив жестко закодирован), и что еще хуже, функция `explode()` фактически содержит всю пунктуацию — строку “you” найти невозможно, тогда как “you,” (с запятой) находится. На первый взгляд это может показаться сложной проблемой, но на самом деле все решается просто с помощью регулярного выражения:

```
<?php
$words = "you, should, vote, happily";
if (ereg("vote", $words)) {
    echo "Найдена строка 'vote'";
}
?>
```

В данном случае используется функция `ereg()` и указывается образец (слово, с которым необходимо сравнивать выражение) и строка для сравнения. Функция возвращает True, если сравнение с образцом оказалось успешным (в данном случае когда в значении переменной `$words` функция находит последовательность символов “vote”), и False, если совпадение не найдено.

Функции `ereg()` можно также передать третий аргумент: имя массива, который будет использоваться для хранения совпавших выражений. Ниже показан соответствующим образом измененный код:

```
<?php
$words = "you, should, vote, happily";
if (ereg("vote", $words, $reg)) echo "Найдена строка '$reg[0]'";
?>
```

Текст, написанный буквально, представляет собой простейший случай регулярного выражения. Так можно искать одно слово или определенную фразу, однако при этом все искомые символы должны в точности совпадать — слова (с учетом регистра символов), числа, знаки препинания и даже пробелы:

```
<?php
$words = "Vote twice or more if you can.";
if (ereg("twice if", $words, $reg)) echo "Найдена строка '$reg[0]'";
?>
```

Эта строка не будет совпадать с регулярным выражением, потому что в ней нет точного соответствия образцу. Аналогично пробелы внутри образца также имеют значение:

```
<?php
$words1 = "The bigdog is in the pound...";
$words2 = "{...}but the dog is in the cornfield";
$regexp = " dog";
if (ereg($regexp, $words1, $reg)) echo "Найдена строка '$reg[0]'";
if (ereg($regexp, $words2, $reg)) echo "Найдена строка '$reg[0]'";
?>
```

Такое регулярное выражение может найти только второе вхождение слова dog, потому что оба вызова `ereg()` специально ищут пробел, за которым следует три буквы “d”, “o” и “g”.

Специальные символы

Синтаксис регулярных выражений включает в себя использование специальных символов (которые не следует путать со специальными символами HTML). В регулярных выражениях специальные символы позволяют создавать более сложные выражения, в которых целые части образца могут состоять из одного из множества символов или совпадение с образцом должно начинаться с определенной позиции строки.

Как уже отмечалось, чтобы экранировать специальное значение какого-либо символа, можно использовать обратную косую черту. Например, для вывода на экран символа двойной кавычки необходимо использовать `escape`-последовательность `\` (то же самое делает функция `addslashes()` при добавлении строк в базу данных).

Ниже показаны символы, имеющие специальное значение в регулярных выражениях. Для того чтобы использовать их буквально, необходимо предварять их символами обратной косой черты.

```
. * ? + [ ] ( ) { } ^ $ | \
```

Для всех остальных символов автоматически предполагается их буквальное значение. Например, чтобы в предыдущем примере задать образец “...”, необходимо использовать следующий код:

```
<?php
$words1 = "The bigdog is in the pound...";
$regexp = "pound\\.\\.\\.\\.";
if (ereg($regexp, $words1, $reg)) echo "Найдена строка '$reg[0]'";
?>
```

Регулярное выражение “pound...” совпадает с частью тестовой строки, кроме этого, оно применимо в следующем примере, потому что точка в данном случае интерпретируется не как точка в регулярном выражении, а как специальный символ, который соответствует любому символу:

```
<?php
$words1 = "The bigdog is in the pound but the dog is in the cornfield.";
$regexp = "pound\\.\\.\\.\\.";
if (ereg($regexp, $words1, $reg)) echo "Найдена строка '$reg[0]'";
?>
```

Этот код выдаст следующий результат:

Найдена строка 'round bu'

Это происходит потому, что точка (.) является специальным символом. Она соответствует одному любому символу кроме символа новой строки. Поэтому в данном случае три точки совпадают с любыми тремя символами после слова “round”, а не только с тремя точками.

Некоторые распространенные образцы и их варианты

Существует несколько доступных вариантов для создания образцов в регулярных выражениях. Рассмотрим их по порядку.

Классы символов: [xyz]

Образец, заключенный в квадратные скобки, называется *классом символов* и означает, что допускается совпадение с любым из этих символов. Например, регулярное выражение “w[ao]nder” совпадает как со словом “wander”, так и со словом “wonder”.

Чтобы создать исключаящий класс символов, необходимо начать его со знака ^. Например, регулярное выражение “[^1234567890]” совпадает с любым символом, кроме цифры.

Чтобы задать диапазон символов, можно использовать дефис. Например, предыдущее регулярное выражение можно переписать так: [^0-9], а выражение [a-z] будет соответствовать строчным латинским буквам.

Можно использовать один или несколько диапазонов вместе, поэтому для поиска одной шестнадцатеричной цифры можно использовать выражение [0-9A-F]. Квадратные скобки содержат целое выражение, но представляют только один-единственный символ, который соответствует любому символу, попадающему в один из диапазонов данного класса. Если бы использовалось выражение [0-9][A-F], то оно соответствовало бы одной цифре, за которой следует одна буква от A до F включительно.

Некоторые классы символов, такие как цифры, буквы и различные типы пустого пространства, встречаются в проверяемых строках снова и снова. Для них используется несколько лаконичных шаблонов. Ниже представлены наиболее распространенные из таких шаблонов, а также их значения.

Шаблон	Класс символов	Описание
\d	[0-9]	Цифры от 0 до 9
\w	[0-9A-Za-z_]	Символ “слово” (алфавитно-цифровые символы)
\s	[\t\n\r]	Пробельные символы (пробел, символ табуляции, новой строки или возврата каретки)

В следующей таблице приведены исключаящие формы указанных шаблонов:

Шаблон	Класс символов	Описание
\D	[^0-9]	Любой нецифровой символ
\W	[^0-9A-Za-z_]	Любой символ кроме символа “слово”
\S	[^\t\n\r]	Любой символ кроме пробельного

Якоря

Классы символов совпадают с символами в любом месте строки, однако определенные символы можно использовать для того, чтобы указать, где в строке должно встретиться совпадение. Такие символы называются *якорями* (*anchors*).

Имеется два якоря: якорь `^` используется в начале образца и привязывает выражение к началу строки, а якорь `$` используется в конце образца и привязывает его к концу строки. Чтобы проверить, заканчивается ли строка точкой (не забудьте, что точка может быть специальным символом), можно использовать регулярное выражение `\.$`. Аналогично, чтобы проверить, присутствует ли в начале строки прописная буква “В”, можно использовать выражение `^В`.

Границы слова

Поиск слов, перед которыми или после которых есть знаки пунктуации, можно облегчить с помощью специальных символов — *границ слова* (*word boundaries*). Символы границ слова позволяют задать образец как совпадение с началом или концом слова. Такие символы необходимы, поскольку слова не всегда отделяются друг от друга пробелами — иногда слова разделяются запятыми, точками и другими знаками пунктуации.

Например, чтобы искать однобуквенные слова, можно использовать границу слова `\b`, такое регулярное выражение будет иметь вид `\b\w\b`. Как и якоря, шаблон `\b`, по сути, совпадает не с каким-либо отдельным символом, а с позицией между чем-то, не являющимся символом слова (`\W` или конец строки), и символом слова (отсюда и название — граница слова).

Альтернативы

В некоторых случаях может оказаться полезным символ, создающий исключительное условие (или/или). В регулярных выражениях оператор “исключающего или” такой же, как битовый оператор “или”: `|`. Например, чтобы найти строку “yes” или “maybe”, следует использовать выражение `yes|maybe`.

Квалификаторы

Квалификаторы (*qualifiers*) — символы `?`, `+` и `*` — позволяют создавать регулярные выражения, совпадающие с набором символов, который может встречаться только один раз, несколько раз или не встречаться вообще. Простейшим из них является знак вопроса `?`, соответствующий предыдущему символу (символам) или метасимволу (метасимволам), который встречается только однажды или ни разу. Например, чтобы найти слово “he” или “she”, можно использовать выражение `s?he` (буквы “s” и “h” отделены друг от друга знаком вопроса). Такое выражение ищет любую из двух букв. Если в строке нет буквы “s”, но есть буква “h” (как в слове “he”), то такая строка все равно совпадает с данным регулярным выражением.

Чтобы сделать последовательность символов (или метасимволов) необязательной, их можно сгруппировать в скобках: с помощью регулярного выражения `(wo)?man` можно искать слова “man” или “woman”.

Искать одно или несколько вхождений символа (или символов) можно с помощью знака “плюс”. Например, для поиска целого слова без указания его длины используется выражение `\w+`.

Можно также искать символы, которые могут встречаться любое количество раз или не встречаться вообще (т.е. ни одного, одно или несколько вхождений). Для таких случаев используется квалификатор `*` (“звездочка”). Например, чтобы найти прописную букву после любого количества (или ни одного) пробелов в начале строки, используется выражение `^\s*[A-Z]`.

Различия данных квалификаторов наглядно демонстрируются следующими примерами:

- ❑ `hea?t` — находит слова “heat” или “het”;
- ❑ `hea+t` — находит “heat”, “heaat”, “heaaat” и т.д.;
- ❑ `hea*t` — находит “het”, “heat”, “heaat” и т.д.

Начинающие программисты склонны к излишнему использованию комбинаций точки (представляющей любой символ) и звездочки (представляющей любое количество символов или ни одного символа), что часто приводит к неожиданным и нежелательным результатам. Можно сформулировать два практических правила:

- ❑ регулярное выражение почти никогда не следует начинать с комбинации какого-либо символа и звездочки;
- ❑ шаблоны `.*` или `.*` в середине регулярного выражения совпадают с максимально возможным количеством подстрок в строке.

Квантификаторы

Квантификаторы (*quantifiers*) используются для установки ограничений и диапазонов количества совпадающих символов. Например, выражение `\s{2,3}` соответствует двум или трем пробелам (или другим символам, упомянутым в приведенной выше таблице шаблонов). Символ `\s` заставляет регулярное выражение искать пробелы, а фигурные скобки, окружающие цифры 2 и 3, заставляют использовать эти цифры в качестве диапазона: от 2 до 3 символов. Первый символ в фигурных скобках представляет минимальную границу диапазона, а второй — максимальную. Диапазоны квантификаторов функционируют подобно диапазонам в классах символов, но их синтаксис несколько отличается: сначала пишется искомый символ, а затем в фигурных (а не в квадратных) скобках указываются числа, разделенные запятой.

Опуская числа максимума или минимума (но оставляя запятую), можно задать диапазоны “не менее” или “не более” соответственно. Например, запись `{2,}` означает “не мене двух”, а `{,3}` — “не более трех”. В этих случаях применяются те же правила, что и для оператора `*`.

Можно создавать весьма мощные выражения, используя несколько из описанных специальных символов. Например, можно указать количество определенных подстрок в строке, задав это число в фигурных скобках: выражение `\b\w{5}\b` соответствует слову из пяти букв.

Сводная характеристика метасимволов

В следующей таблице представлено краткое описание рассмотренных метасимволов:

Метасимвол	Значение
<code>[abc]</code>	Любой из символов a, b или c
<code>^abc</code>	Один любой символ a, b или c
<code>[a-z]</code>	Один любой ASCII-символ между a и z включительно
<code>\d</code> и <code>\D</code>	Цифра и не цифра
<code>\w</code> и <code>\W</code>	Символы “слово” и “не слово”
<code>\s</code> и <code>\S</code>	Символ пустого пространства и его альтернатива
<code>\b</code>	Граница между символами <code>\w</code> и <code>\W</code>

Окончание таблицы

Метасимвол	Значение
.	Любой символ (кроме символа новой строки)
(abc)	Фраза “abc” как группа символов
?	Предыдущий символ или группа символов может присутствовать 0 или 1 раз
+	Предыдущий символ или группа символов может присутствовать 1 или несколько раз
*	Предыдущий символ или группа символов может присутствовать 0 или несколько раз
{x,y}	Предыдущий символ или группа символов присутствует от x до y раз
{,y}	Предыдущий символ или группа символов присутствует не более y раз
{x,}	Предыдущий символ или группа символов присутствует не менее x раз
{x}	Предыдущий символ или группа символов присутствует x раз
^	Начало строки
\$	Конец строки

Проверка ввода данных

Одним из наиболее распространенных элементов, вводимых пользователями в Web-приложениях, является e-mail-адрес, который благодаря своему уникальному формату представляет отличный пример для проверки данных формы. Во-первых, нелегко определить длину адреса, потому что обычно неизвестно, какие символы используются в именной части адреса. Во-вторых, адрес содержит знак @, который обычно не встречается в других распространенных элементах данных. И в-третьих, обязательно присутствие .com, .net, .org или любой другой подстроки допустимой в конце доменного имени.

Адрес электронной почты столь часто запрашивается в Web-формах, потому что он может оказаться единственной контактной информацией, предоставляемой пользователем, и если по какой-либо причине адрес введен некорректно, возможность связаться с данным пользователем теряется.

Существует множество побудительных мотивов для проверки e-mail-адресов, а также множество различных программ проверки. Некоторые из них очень сложны и способны проверять адреса с высокой степенью точности, другие весьма просты и их точность намного ниже. Одна из простейших программ лишь проверяет наличие единственного знака @ в адресе — очевидно, что если этого знака нет, то адрес не может быть корректным, с другой стороны, если знак @ встречается несколько раз, то такой адрес также некорректен. Хотя само по себе наличие одного знака @ не гарантирует корректности адреса.

Регулярные выражения удобны для создания подпрограмм проверки адресов, потому что с их помощью можно реализовать множество вариантов условной проверки. Например, в любом e-mail-адресе перед и после знака @ должны присутствовать какие-либо символы. Справа от символа @ должна быть точка (.), перед и после которой также присутствуют какие-либо символы. Кроме того, существуют символы, которые не допускаются использовать в e-mail-адресах (например, пробел). Образцы регулярных выражений можно конструировать так, чтобы они весьма точно определяли аномалии в структуре адреса:

```
^[^@ ]+@[^@ ]+\.[^@ \.]+$
```

Этот образец на первый взгляд может показаться бессмысленным; его следует развернуть. В следующей таблице описываются по порядку все символы и их значения.

Символ	Значение
^	В начале строки...
[^@]	...один любой символ, кроме @ или пробела,
+	...который встречается один или несколько раз
@	Символ @
[^@]	Затем один любой символ, кроме @ или пробела,
+	...который встречается один или несколько раз
\.	Точка (которую необходимо экранировать с помощью обратной косой черты)
[^@ \.]	Любой символ, кроме @, пробела или точки,
+	...который встречается один или несколько раз. Сразу за его последним вхождением должен быть...
\$...конец строки

Данный образец прост, но ни в коем случае не идеален. Существуют гораздо более сложные процедуры анализа e-mail-адресов и URL, которые могут быть очень запутанными.

Рассмотрим на примере использование данного регулярного выражения в функции `ereg()` для проверки вводимых e-mail-адресов.

Практика Проверка e-mail-адресов

1. Введите в HTML-редакторе следующий код:

```
<html>
<head><title></title></head>
<body>
<?php
//email_validation.php
if (isset($_POST['posted'])) {

    $email = $_POST['email'];
    $theresults = ereg("^[^@ ]+@[^@ ]+\.[^@ \.]+$", $email, $strashed);
    if ($theresults) {
        $isamatch = "корректен";
    } else {
        $isamatch = "некорректен";
    }

    echo "Введенный для проверки адрес $email " . $isamatch;
}

?>
<form action="email_validation.php" method="POST">
<input type="hidden" name="posted" value="true">
Введите e-mail-адрес для проверки:
<input type="text" name="email" value="name@example.com">
<input type="submit" value="Проверить">
</form>
</body>
</html>
```

2. Сохраните данный файл как `email_validation.php` и закройте его.
3. Запустите сценарий и проверьте его работу как с “хорошими”, так и с “плохими” e-mail-адресами. На рис. 5.4 показан примерный результат после ввода некорректного адреса; неправильно введенный адрес отображается вверху страницы.

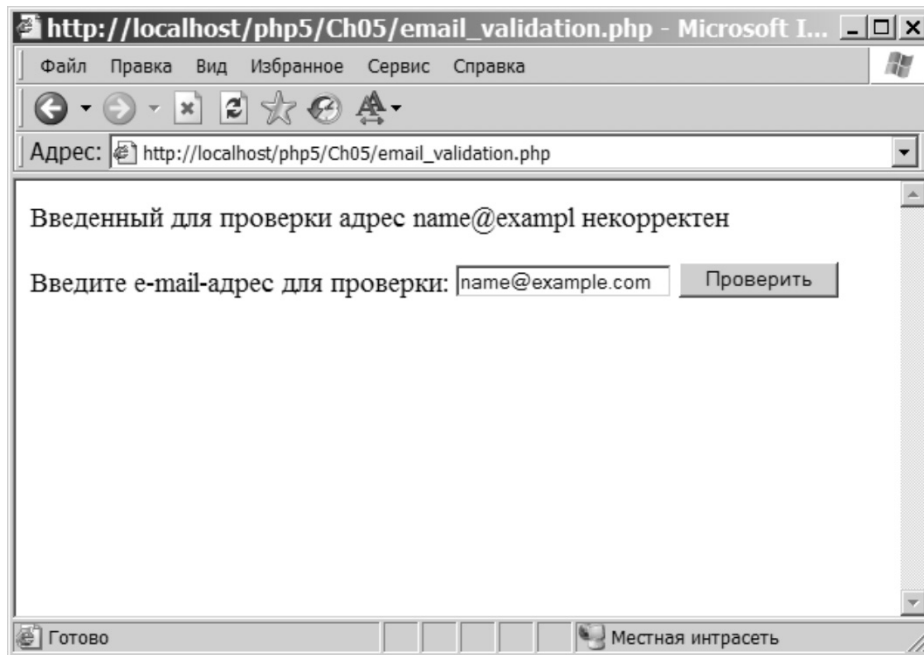


Рис. 5.4.

Как это работает

Сравнение введенной строки с образцом выполняется с помощью функции `ereg()`. Когда данные формы отправляются, введенная пользователем строка передается в переменную `$email` (из переменной `$_POST['email']`). Затем для проверки корректности данной строки используется функция `ereg()`. Если адрес правильный, то в переменную `$ismatch` записывается значение “корректен”, в противном случае — “некорректен”. Затем результат отображается в браузере пользователя.

```
<?php
//email_validation.php
if (isset($_POST['posted'])) {
    $email = $_POST['email'];
    $theresults = ereg("^[^@ ]+@[^@ ]+\.[^@ \.]+$", $email, $strashed);
    if ($theresults) {
        $ismatch = "корректен";
    } else {
        $ismatch = "некорректен";
    }
}

echo "Введенный для проверки адрес $email " . $ismatch;
}
?>
```

Чтобы показать пользователю необходимый формат вводимых данных, в поле формы предустановлен e-mail-адрес по умолчанию (предполагается, что пользователь понимает, что этот адрес необходимо заменить реальным значением).

Приведенного выше сценария вполне достаточно для проверки e-mail-адресов, хотя он не идеален. Он позволяет вводить после знака @ недопустимые в доменном имени символы (например, *) и, по сути, не проверяет существование введенного доменного имени. Наилучшим способом проверки e-mail-адреса является передача этого адреса серверу; даже правильно сформированный e-mail-адрес является некорректным, если сервер, с которым он связан, не принимает его.

Использование регулярных выражений для проверки URL-указателей

Доменные имена и полные URL (Uniform Resource Locator — унифицированный указатель ресурсов) представляют собой интересный пример для применения поисковых возможностей регулярных выражений. Структура доменного имени проста — имя, за которым следует точка и расширение доменного имени (например, .com, .net, .org и т.д.), но существуют правила, запрещающие использование в доменном имени некоторых символов; кроме того, включать в доменное имя префикс www (или другой) необязательно. URL-указатели включают в себя доменные имена и префикс http://, а в состав полного URL также может быть включен путь (имена каталогов, разделенных косой чертой), имя файла и строка запроса, которая добавляется в конец URL. Существует немало вариантов доменных имен и URL, поэтому, как и в случае e-mail-адресов, важно правильно их понимать.

Ниже описан формат унифицированного указателя ресурсов (URL):

- протокол (например, ftp или http);
- доменное имя или имя сервера (например, wtoх.com; www использовать необязательно);
- необязательный в некоторых случаях каталог и путь к файлу (каталог и имя файла, разделенные символом косой черты, например, images/myimage.gif);
- необязательная строка запроса (начинается со знака вопроса, за которым следует одна или несколько пар имя/значение).

Рассмотрим код регулярного выражения:

```
^[a-zA-Z0-9]+://[^\ ]+$
```

Аналогично ключевой строке в предыдущем примере, данное выражение можно использовать в следующем коде:

```
$theresults = ereg("^[a-zA-Z0-9]+://[^\ ]+$", $intext, $trashed);
```

В следующем примере проверяется корректность формата URL-указателей.

Практика Проверка корректности формата URL

1. Запустите редактор Web-страниц и введите следующий код:

```
<html>
<head></head>
<body>
<?php
//url_validate.php
if (isset($_POST['posted'])) {
    $url = $_POST['url'];
```

```

    $theresults = ereg("^[a-zA-Z0-9]+://[^\"]+$", $url, $trashed);
    if ($theresults) {
        $isamatch = "корректен";
    } else {
        $isamatch = "некорректен";
    }
    echo "Введенный URL $url " . $isamatch;
}
?>
<form action="url_validate.php" method="POST">
<input type="hidden" name="posted" value="true">
Введите URL для проверки:
<input type="text" name="url" value="http://www.example.com" size="30">
<input type="submit" value="Проверить">
</form>
</body>
</html>

```

2. Сохраните созданный файл как `url_validate.php` и закройте его.
3. Запустите сценарий и проверьте его работу путем ввода как корректных, так и некорректных URL-указателей.

На рис. 5.5 показан примерный результат работы данного сценария после ввода некорректно сформированного URL-указателя.

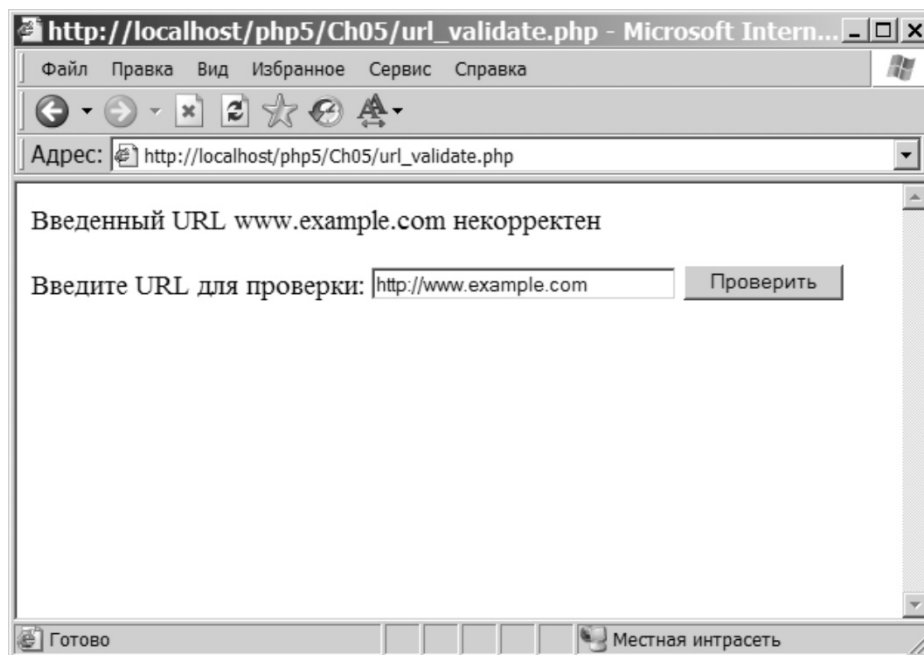


Рис. 5.5.

Как это работает

Аналогично `email_validation.php` данный сценарий главным образом основывается на единственном регулярном выражении, которое обрабатывается функцией `ereg()`:

```
$theresults = ereg("^[a-zA-Z0-9]+://[^\"]+$", $url, $trashed);
```

Это регулярное выражение совпадает с корректными URL-указателями, но, к сожалению, оно также совпадает со многими другими строками, которые не являются корректными URL. Как и в случае многих образцов регулярных выражений, выясняется, что добиться идеального совпадения с образцом, не затрачивая при этом большого количества времени, очень трудно (некоторые образцы проверки URL насчитывают сотни символов в длину и, несмотря на это, не являются идеальными). Главная цель заключается в том, чтобы устранить наиболее распространенные ошибки. В конце концов, любые незначительные изменения в составе корректного URL-указателя (или e-mail-адреса) могут сделать регулярное выражение бессмысленным — что идеально сегодня, завтра может оказаться быть неприемлемым.

Использование регулярных выражений для проверки параметров файловых путей

Файловые системы и PHP-функции для работы с файлами и каталогами обсуждаются в главе 7. Здесь мы вкратце рассмотрим те функции регулярных выражений, которые могут оказаться полезными для защиты данных, сохраняемых в файлах (именно в файлах — базы данных не единственный способ постоянного хранения данных; для длительного хранения информации приложений часто используются обычные текстовые файлы).

Термин постоянные применительно к сохраняемым данным используется для обозначения данных, которые не теряются даже после закрытия приложения или выключения системы.

Эти функции (а также примеры кода с их использованием) упрощают проверку данных, которые представляют собой имена файлов или каталогов, и поэтому могут оказаться очень полезными для ограничения доступа к определенным каталогам и файлам. Можно разрешить доступ пользователям к определенным файлам и каталогам, однако к некоторым из них доступ должен иметь только администратор.

В следующем примере путем удаления конфиденциальной информации из файловых путей запрещается просмотр обычными пользователями дерева каталогов. В данном случае можно эффективно использовать вариант функции `ereg()` — функцию `ereg_replace()`.

Чтобы решить эту задачу, необходимо составить образец, который с помощью функции `ereg_replace()` позволяет удалить из строки символы "\", "/" или ". ./". В Unix-системах в качестве ограничивающего символа используется косая черта, тогда как в Windows применяются символы обратной косой черты, а в MacOS используется двоеточие. Кроме того, данный код будет удалять любые абсолютные пути, начинающиеся с символа "/" или [A-Z].

Практика Предотвращение доступа к секретным файлам

1. Откройте редактор Web-страниц и введите следующий код:

```
<html>
<head><title></title></head>
<body>
<?php
//clean_path.php
```

```

if (isset($_POST['posted'])) {
    $path = $_POST['path'];
    $outpath = ereg_replace("\.[\.]+" , "", $path);
    $outpath = ereg_replace("^[/]+" , "", $outpath);
    $outpath = ereg_replace("[A-Za-z][:\[\]][/]?" , "", $outpath);
    echo "Старый путь: " . $path . " . Новый путь: " . $outpath;
}
?>
<form action="clean_path.php" method="POST">
<input type="hidden" name="posted" value="true">
Введите файловый путь для очистки:
<input type="text" name="path" size="30">
<input type="submit" value="Очистить">
</form>
</body>
</html>

```

2. Сохраните созданный файл `clean_path.php` и закройте его.
3. Запустите программу. Ее вывод должен выглядеть примерно так, как показано на рис. 5.6.

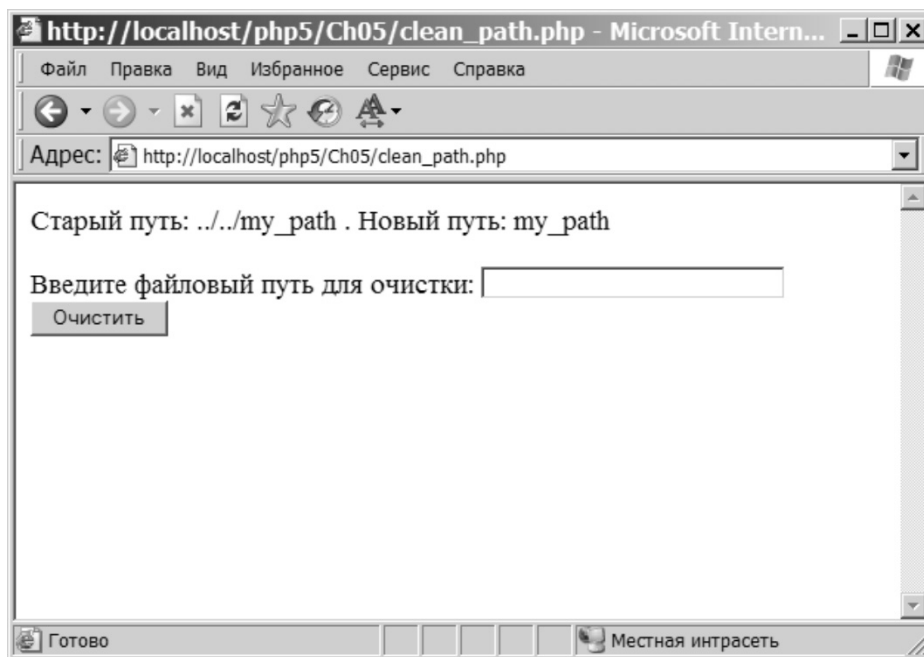


Рис. 5.6.

Очевидно, что PHP можно использовать для фильтрации вводимых пользователем данных, которые являются либо потенциально опасными, либо просто некорректными.

Как это работает

Первая строка программы удаляет комбинации `".."` (которые используются для перехода на один уровень выше в дереве каталогов):

```
$outpath = ereg_replace("\.[\.]+" , "", $inpath);
```

Вторая строка удаляет ограничивающие символы косой черты или обратной косой черты:

```
$outpath = ereg_replace("^[/\]+", "", $outpath);
```

Третья строка удаляет префиксы DOS/Windows-стиля (например, "C:\"):

```
$outpath = ereg_replace("^([A-Za-z] [:\|] [\/]?)", "", $outpath);
```

Изящная обработка ошибок

Главное отличие между изящной и неряшливой обработкой ошибок заключается в том, что видит пользователь. Конечно, правильная обработка ошибок в программе может быть реализована по-разному, но для пользователя, который только что пытался открыть какую-либо страницу, гораздо более понятным будет четко написанное сообщение, а не просто встроенное PHP-сообщение об ошибке.

Как и многие другие языки программирования, PHP постепенно становится более зрелым. В PHP5 появились новые встроенные возможности обработки ошибок (они рассматриваются в следующем разделе). Для начала следует описать, в чем заключается обработка ошибок, а затем рассмотреть обработку ошибок с момента загрузки PHP в систему и до выполнения программ в реальном времени.

Конфигурирование обработки ошибок в PHP

Главными конфигурационными параметрами, которые следует использовать в разрабатываемых (а не в уже действующих) системах, являются уровень ошибок (`error_reporting`), настройка отображения сообщений об ошибках (`display_errors`) и параметр `log_errors`. В приложении “Конфигурация PHP5” рассматриваются и многие другие настройки, однако данные три параметра вызывают наибольший интерес. Первый параметр устанавливает уровень отображаемых ошибок, второй включает или выключает отображение сообщений об ошибках в браузере (в реально действующих сайтах отображать стандартные сообщения об ошибках нежелательно), а третий включает или выключает протоколирование ошибок в файле журнала.

Подавление сообщений об ошибках

Подавление ошибок, строго говоря, нельзя назвать элементом отладки, но иногда известно, что ошибки в PHP-сценарии есть и избавляться от соответствующих сообщений нежелательно, необходимо просто проигнорировать их или обработать ошибки особым образом. Для таких ситуаций в PHP используется специальная форма записи: @-нотация. Когда символ @ используется с какой-либо функцией, PHP подавляет все сообщения об ошибках, возникающие в данной функции. Например, следующая функция вызывается с предшествующим символом @:

```
function ProcessFormDetails ($Name, $Email)
{
    //какая-либо обработка
}
@ProcessFormDetails($Name, $Email);
```

Благодаря такой записи любые ошибки внутри данной функции не отображаются. Конечно, если в функции есть ошибка, то все равно возвращается нулевое значение. Если эта ошибка критическая и в обычных условиях привела бы к остановке сценария,

то она и в этом случае останавливает сценарий, но не выводит на экран каких-либо сообщений. И хотя подавление сообщений об ошибках может привести к некоторой путанице, оно, тем не менее, остается полезным средством. Оно позволяет обрабатывать ошибку отдельно внутри предназначенной для этого функции, которая возвращает пользователю специальное сообщение.

Проверка журнала ошибок

По умолчанию PHP не записывает сообщения об ошибках в журнал Web-сервера, однако такое поведение можно исправить. PHP может протоколировать ошибки в файл журнала, если в файле `php.ini` присутствуют следующие директивы:

```
log_errors = On
error_log = /var/log/php.log
```

В PHP также имеется функция `error_log`, которая позволяет протоколировать неожиданные ситуации, не предусмотренные программистом и не имеющие специального способа программной обработки. Это позволяет изучать возникающие ошибки, даже если вывод соответствующих сообщений был подавлен описанным выше способом.

Функция `error_log` может принимать до четырех аргументов, но обязательными являются только первые два из них. Функция имеет следующий формат:

```
error_log("Error Message", MessageType, "Destination", "Extra Headers");
```

Первый аргумент `Error Message` представляет собой фактическое сообщение об ошибке. Второй аргумент, `MessageType`, — код, означающий то место, куда необходимо отправлять сообщение об ошибке; существует четыре возможных значения этого аргумента:

- ☐ 0: сообщения отправляются в журнал ошибок PHP;
- ☐ 1: сообщения отправляются на e-mail-адрес, заданный третьим аргументом;
- ☐ 2: сообщения отправляются в отладочный канал PHP (только если включен режим отладки);
- ☐ 3: сообщения отправляются в файл журнала, путь к которому задается третьим аргументом.

Третий аргумент, `Destination`, принимает либо e-mail-адрес, либо путь к файлу, а четвертый аргумент, `Extra Headers`, можно использовать для отправки дополнительной информации в форме e-mail-заголовков.

Чтобы получить подробные сведения о возникшей ошибке, можно включить протоколирование ошибок в журнал или использовать функцию `error_log`, а затем просмотреть содержащуюся в журнале информацию.

Try/Catch — нововведения в PHP5

Несколько новых средств в PHP5 позволяют программам изящно реагировать на возникающие ошибки. Речь идет о новых возможностях по обработке исключительных ситуаций — блоках `try/catch`. По существу, создается блок `try` и в него помещается обрабатываемый код, который может генерировать ошибки. Затем в зависимости от сгенерированной ошибки используется блок `catch`, в котором принимается решение о том, что следует делать с этой ошибкой. Одно из важных преимуществ такой схемы состоит в том, что она облегчает перехват ошибок в блоках функций.

Функциональность try/catch генерирует исключения, а исключения являются объектами. Из блока try с помощью оператора new Exception “выбрасываются” исключения, которыми затем занимается блок catch. Созданное исключение представляет собой объект, его можно использовать так же, как и любой другой объект (подробнее объекты рассматриваются в этой книге далее). Метод getMessage объекта Exception позволяет получить сообщение, связанное с возникшей ошибкой. Использование функциональности try/catch наглядно демонстрируется в следующем примере.

Практика Использование блоков try/catch для проверки данных формы

1. Запустите HTML-редактор и создайте файл со следующим кодом:

```
<html>
<head><title>PHP5 для начинающих</title></head>
<body bgcolor="#FFFFFF">
<?php
if (isset($_POST['posted'])) {

    //записать переданные значения в обычные переменные

    $first_name = $_POST['first_name'];
    $last_name = $_POST['last_name'];
    $birth_date = $_POST['birth_date'];
    $phone = $_POST['phone'];
    $age = $_POST['age'];
    $address = $_POST['address'];
    $city = $_POST['city'];
    $state = $_POST['state'];
    $postal_code = $_POST['postal_code'];

    //создать массив из имен полей и типов данных

    $field_names = array("first_name" => "string",
        "last_name" => "string",
        "birth_date" => "date",
        "phone" => "string",
        "age" => "integer",
        "address" => "string",
        "city" => "string",
        "state" => "string",
        "postal_code" => "string");

    //по имени поля проверить тип данных каждого переданного значения

    function form_validate($fns) {

        foreach ($fns as $key => $value) {
            $field_value = $key;
            global $$field_value;
            //echo "фактическое значение поля " . $$field_value . "<br>";
            switch ($value) {
                Case "string";
                    if ((strlen($$field_value) < 1) or (strlen($$field_value) > 99)) {
                        throw new Exception("Пожалуйста, введите в поле
                            <b>$key</b> строку от 1 до 100 символов");
                    }
                    break;
                Case "date";
                    if (!ereg("[0-9]{4}\-([1-9]|(0[1-9])|(1[0-2]))\-([1-9]|
                        (0[1-9])|([1-2][0-9])|3[0-1])$", $$field_value)) {
```

```

        throw new Exception("Пожалуйста, введите в поле <b>$key</b>
                               корректную дату в формате ГГГГ-ММ-ДД");
    }
    break;
    Case "integer";
    if (!is_numeric($$field_value)) {
        throw new Exception("Пожалуйста, введите в поле <b>$key</b>
                               число без десятичных разделителей или алфавитных символов.");
    }
    break;
    default;
    break;
}
}
}
//перехватить исключение и сгенерировать сообщение об ошибке
try
{
    form_validate($field_names);
}
catch (Exception $e)
{
    echo $e -> getMessage();
    echo "<br>";
}
}
//если ошибок не было, то поблагодарить пользователя
if (!is_object($e) and isset($posted)) {
    echo "Спасибо за информацию, мы свяжемся с Вами.";
} else {
    //вернуть пользователю заполненную форму и предложить ввести
    исправленные данные
?>
<form action="try_catch.php" method=post>
<input type="hidden" name="posted" value="true">
<table width="50%" border="1">
    <tr>
        <td colspan="2"><font face="Arial, Helvetica, sans-serif" size="-1">
            Пожалуйста, введите контактную информацию:</font></td>
    </tr>
    <tr>
        <td width="26%"><font face="Arial, Helvetica, sans-serif" size="-1">Имя
        </font></td>
        <td width="74%">
            <input type="text" name="first_name" value="<?php echo $first_name; ?>">
        </td>
    </tr>
    <tr>
        <td width="26%"><font face="Arial, Helvetica, sans-serif" size="-1">
            Фамилия
        </font></td>
        <td width="74%">
            <input type="text" name="last_name" value="<?php echo $last_name; ?>">
        </td>
    </tr>
    <tr>
        <td width="26%"><font face="Arial, Helvetica, sans-serif" size="-1">
            Дата рождения</font></td>
        <td width="74%">
            <input type="text" name="birth_date" value="<?php echo $birth_date; ?>">
        </td>
    </tr>
    <tr>

```

```

        <td width="26%"><font face="Arial, Helvetica, sans-serif" size="-1">
            Номер телефона</font></td>
        <td width="74%">
            <input type="text" name="phone" value="<?php echo $phone; ?>">
        </td>
    </tr>
    <tr>
        <td width="26%"><font face="Arial, Helvetica, sans-serif" size="-1">
            Возраст</font></td>
        <td width="74%">
            <input type="text" name="age" value="<?php echo $age; ?>">
        </td>
    </tr>
    <tr>
        <td width="26%"><font face="Arial, Helvetica, sans-serif" size="-1">
            Адрес</font></td>
        <td width="74%">
            <input type="text" name="address" value="<?php echo $address; ?>">
        </td>
    </tr>
    <tr>
        <td width="26%"><font face="Arial, Helvetica, sans-serif" size="-1">
            Город</font></td>
        <td width="74%">
            <input type="text" name="city" value="<?php echo $city; ?>">
        </td>
    </tr>
    <tr>
        <td width="26%"><font face="Arial, Helvetica, sans-serif" size="-1">
            Имя</font></td>
        <td width="74%">
            <input type="text" name="state" value="<?php echo $state; ?>">
        </td>
    </tr>
    <tr>
        <td width="26%"><font face="Arial, Helvetica, sans-serif" size="-1">
            Почтовый индекс</font></td>
        <td width="74%">
            <input type="text" name="postal_code" value="<?php echo $postal_code; ?>">
            <input type="submit" value="Отправить информацию" name="submit">
        </td>
    </tr>
</table>
</form>
<?php
}
?>

</body>
</html>

```

2. Сохраните данный файл как `try_catch.php` и закройте его.
3. Откройте файл в браузере (рис. 5.7), введите значения для каждого поля и просмотрите полученные после отправки формы сообщения об ошибках. На рис. 5.8 и 5.9 показаны некоторые примеры сообщений об ошибках.

Если все данные ввести правильно, то появится сообщение с благодарностью.

Рис. 5.7. Скриншот веб-браузера Microsoft Internet Explorer, отображающий форму для ввода контактной информации. Заголовок окна: "PHP5 для начинающих - Microsoft Internet Explorer". Адресная строка: "http://localhost/php5/Ch05/try_catch.php".

Пожалуйста, введите контактную информацию:

Имя	<input type="text"/>
Фамилия	<input type="text"/>
Дата рождения	<input type="text"/>
Номер телефона	<input type="text"/>
Возраст	<input type="text"/>
Адрес	<input type="text"/>
Город	<input type="text"/>
Штат	<input type="text"/>
Почтовый индекс	<input type="text"/>

Отправить информацию

Рис. 5.7.

Как это работает

Программа начинается с формы, в которую пользователь вводит данные. Эти данные должны быть определенных типов. Форму можно было бы легко расширить для многих других типов данных. Затем, как и во всех остальных примерах, чтобы определить, была ли отправлена данная форма, используется функция `isset()` (в данном случае проверка выполняется в нескольких местах). Переданные данные записываются в обычные переменные, и создается массив имен полей и типов данных, который будет использоваться позднее для настройки подпрограмм проверки данных. Массив выглядит следующим образом:

```
//создать массив из имен полей и типов данных
$field_names = array("first_name" => "string",
    "last_name" => "string",
    "birth_date" => "date",
    "phone" => "string",
    "age" => "integer",
    "address" => "string",
    "city" => "string",
    "state" => "string",
    "postal_code" => "string");
```

PHP5 для начинающих - Microsoft Internet Explorer

Файл Правка Вид Избранное Сервис Справка

Адрес: http://localhost/php5/Ch05/try_catch.php

Пожалуйста, введите в поле **birth_date** корректную дату в формате ГГГГ-ММ-ДД

Пожалуйста, введите контактную информацию:

Имя	Дэйв
Фамилия	Мерсер
Дата рождения	1 января 1999
Номер телефона	
Возраст	
Адрес	
Город	
Штат	
Почтовый индекс	

Отправить информацию

Готово Местная интрасеть

Рис. 5.8.

PHP5 для начинающих - Microsoft Internet Explorer

Файл Правка Вид Избранное Сервис Справка

Адрес: http://localhost/php5/Ch05/try_catch.php

Пожалуйста, введите в поле **age** число без десятичных разделителей или алфавитных символов.

Пожалуйста, введите контактную информацию:

Имя	Дэйв
Фамилия	Мерсер
Дата рождения	1999-01-01
Номер телефона	2223333
Возраст	xx
Адрес	
Город	
Штат	
Почтовый индекс	

Отправить информацию

Готово Местная интрасеть

Рис. 5.9.

Разработка баз данных в книге пока еще не рассматривалась (эта тема описана в нескольких последующих главах), тем не менее, следует отметить, что имена полей и типы данных (а также другую информацию для проверки формы) можно легко получить из базы данных. На самом деле можно было бы хранить в базе данных большое количество сведений о форме и процессе проверки данных этой формы. И хотя такой подход требует несколько больших трудозатрат, заказчик приложения впоследствии сможет легко изменить количество и тип отображаемых в форме полей, зная, что форма все равно будет обрабатываться соответствующим образом.

Затем создается функция, осуществляющая проверку данных, переданных формой. Для того чтобы процесс проверки охватил все поля формы, внутри функции используется цикл `foreach`, проверяющий имена полей и их значения. Создавать данную функцию было необязательно, однако это облегчило бы работу, если бы надо было проверять несколько массивов значений полей. Функция начинается так:

```
function form_validate($fns) {
    foreach ($fns as $key => $value) {
```

Затем имя поля передается в переменную, после чего используется способность PHP обращаться к значению определенного поля переданной HTML-формы (необходимо использовать ключевое слово `global`, потому что внутри области видимости функции внешние переменные не видны; функции и этот аспект их работы подробнее рассматриваются в главе 6):

```
$field_value = $key;
global $$field_value;
```

Затем используются Case-блоки оператора `switch` для выделения типов проверки. Внутри блока выполняются различные проверки переданных значений:

```
switch ($value) {
    Case "string";
        if ((strlen($$field_value) < 1) or (strlen($$field_value) > 99)) {
            throw new Exception("Пожалуйста, введите в поле <b>$key</b> строку от 1 до 100 символов");
        }
        break;
    Case "date";
        if (!ereg("[0-9]{4}\-([1-9]|(0[1-9])|(1[0-2]))\-([1-9]|(0[1-9])|([1-2]
            [0-9])|3[0-1])$", $$field_value)) {
            throw new Exception("Пожалуйста, введите в поле <b>$key</b> корректную дату
                в формате ГГГГ-ММ-ДД");
        }
        break;
    Case "integer";
        if (!is_numeric($$field_value)) {
            throw new Exception("Пожалуйста, введите в поле <b>$key</b> число без десятичных
                разделителей или алфавитных символов.");
        }
        break;
    default;
        break;
}
```

Обратите внимание, что оператор `throw new Exception` выполняется, когда значение не проходит какую-либо проверку. Например, если длина строкового значения не попадает в пределы 1–99 символов, то генерируется исключение. Во время создания исключения ему назначается сообщение — строковое значение в круглых скобках после ключевого слова `Exception`.

Чтобы функция обрабатывала переданные значения, используются блоки `try/catch`. Функция выполняется внутри блока `try`, а в блоке `catch` перехватывается каждое исключение и выводится соответствующее сообщение. (В блоке `catch` при желании можно реализовать любой вид обработки ошибок.) Кроме того, если возникает исключение, то работа функции прекращается и создается объект `Exception` (исключение) со своим сообщением:

```
try
{
    form_validate($field_names);
}
catch (Exception $e)
{
    echo $e -> getMessage();
    echo "<br>";
}
```

Последняя особенность заключается в том, что если возникает ошибка, то отображаются введенное пользователем значение и сообщение об ошибке, чтобы пользователь мог исправить проблему, не вводя все данные заново. Если ошибок не было, то выводится сообщение с благодарностью. Поля формы заполняются существующими данными, потому что в атрибутах `value` всех полей вызывается оператор `echo`, отображающий переданные данные (когда форма отображается впервые, атрибут `value`, естественно, не содержит никаких данных). В следующем блоке `if` выполняется проверка того, возвращается ли пользователю сообщение с благодарностью. Для этого необходимо проверить, является ли исключение (`$e`) объектом, а также была ли форма отправлена:

```
//если ошибок не было, то поблагодарить пользователя
if (!is_object($e) and isset($posted)) {
    echo "Спасибо за информацию, мы свяжемся с Вами.";
} else {

//вернуть пользователю заполненную форму и предложить ввести исправленные данные
?>
```

Резюме

В данной главе рассматривались базовые функции тестирования, локализации неисправностей, отладки и обработки ошибок, а также проверка строк (для перехвата некорректных данных до того, как они нарушат работу программы), регулярные выражения (для улучшенной проверки) и новые функции PHP5 `try/catch`.

Дефекты возникают в каждой программе. Некоторые дефекты являются результатом неправильного синтаксиса, другие — результатом логических ошибок (когда обработка выполняется, но при этом правильный ответ не генерируется, или обработка выполняется верно, но генерирует неверный ответ из-за ввода неправильных данных). Найти и исправить ошибки любого типа можно с помощью поэтапного процесса тестирования, локализации неисправностей и отладки, который повторяется снова и снова до тех пор, пока программа не будет работать верно.

Организуя обработку ошибок в приложении, разработчик ограничен лишь рамками имеющегося времени и навыками программирования. Как было показано в этой главе, проверка ошибок не только помогает избежать неприглядных и потенциально опасных сообщений об ошибках, но и дает возможность убедиться, что там, где это требуется, пользователи вводят соответствующие данные. Способ обработки входных

данных полностью определяется разработчиком. Чем “умнее” будет сценарий при обнаружении и обработке неправильного ввода, тем лучше будут генерируемые им данные. Чтобы добиться этого, можно использовать множество функций проверки строк и регулярных выражений, описанных в этой главе.

Планирование (и желательно разумное использование регулярных выражений) при обработке ошибок может значительно улучшить Web-сайт. Не забывайте закон Мерфи: если какая-либо ошибка может возникнуть, то она обязательно возникнет. Помня об этом, можно избежать неприятностей — проверить все места, где возникновение ошибки наиболее вероятно, и соответствующим образом исправить сценарии. Предусмотрительность поможет сократить большое количество проблем в будущем.

Упражнение

Создайте не менее трех регулярных выражений и вставьте их в соответствующие места последнего примера в данной главе. Примерные регулярные выражения:

- ☐ семи- и десятизначные номера телефонов в США;
- ☐ номера социального обеспечения;
- ☐ буква для обозначения пола пользователя (М — для мужчин и Ж — для женщин).

6

Создание высококачественного кода

Высококачественный код — это код, который правильно и эффективно работает, хорошо документирован, прост в отладке и сопровождении, а также прост в работе для всех, кто его использует. И поскольку существует множество различных способов создания приложений и реализации одной и той же функциональности, в конечном итоге можно получить код, который выполняет свою задачу, но написан некачественно и, следовательно, труден в сопровождении.

Создать и придерживаться формального процесса разработки может быть очень трудно (особенно если разработчики привыкли программировать по наитию), однако такой процесс облегчает разработку высококачественного кода и является существенно важным для крупных проектов. И хотя некоторые различия в стиле кодирования возможны и иногда даже поощряются, использование согласованного подхода (определенного стандартом написания кода) приносит огромные преимущества, особенно при разработке крупных проектов внутри организации. Использование стандарта кодирования, пример которого приводится в данной главе, является жизненно важной частью создания высококачественного кода.

Другой мерой качества кода является то, насколько хорошо он структурирован. Функции и подключаемые файлы следует использовать всякий раз при выполнении одинаковой или очень похожей обработки в различных частях программы — это означает необходимость модификации кода только в одной точке, а не в нескольких. Эти и другие, обсуждаемые в данной главе, показатели могут помочь в написании первоклассного кода.

В главе 5 было показано, как сделать обработку ошибок более изящной, более безопасной и эффективной. В данной главе продолжается обсуждение темы создания качественного кода, описываются стандарты кодирования, а также использование функций, объектов и операторов `include/require`. Кроме того, в данной главе дан краткий обзор процессов разработки и оптимизации кода. Очевидно, что создание высококачественного кода связано с большими трудозатратами, однако качественный код того стоит.

Планирование разработки

Важность планирования подчеркивается на протяжении всей этой книги. Хотя не каждая работа позволяет заблаговременно предвидеть ход выполнения, разработку приложения всегда лучше тщательно планировать. Усилия, приложенные к планированию приложения, значительно облегчают последующее написание кода.

Формальный процесс разработки программного обеспечения

Существует множество формальных процессов разработки программного обеспечения, которые можно использовать; все они имеют определенные общие черты, такие как написание спецификации, разработка протоколов тестирования и т.д. Компания, разрабатывающая программное обеспечение для правительственных учреждений, должна придерживаться процесса разработки программного обеспечения, определенного в таком стандарте, как ISO 12207 (ISO, International Standards Organization — Международная организация по стандартизации). Если же в компании используется менее формальный подход, например, так называемая “быстрая разработка приложений” (Rapid Application Development, RAD), несколько разработчиков несут общую ответственность за быстрое создание и модификацию работы приложения.

Какой бы процесс разработки не использовался, стандартное соблюдение формальных процедур повышает шансы успешного завершения проекта в срок и в рамках определенного бюджета. Нетрудно убедиться, что спонтанный подход крайне неэффективен, особенно если над проектом работает несколько разработчиков.

Однако даже проекты с очень хорошим менеджментом не застрахованы от задержек, изменения спецификаций, ошибок обычных и трудно обнаруживаемых и т.д. Это означает, что определение и соблюдение формальной процедуры особенно важно.

В следующих разделах обсуждаются основные черты, характерные для формальных процессов разработки программного обеспечения.

Написание спецификации

Написание спецификации — первый этап, однако его нередко игнорируют или не придают ему значения, поскольку часто бывает трудно точно и полностью определить, каким должен быть окончательный продукт. В общих чертах практически всем известно, как создаются кинофильмы. Часто бывает так, что окончательный кинопродукт может очень слабо походить на оригинальную книгу или киносценарий. Фильм может не совсем точно соответствовать сценарию, но сценарий есть всегда — было бы очень трудно без сценария убедить кого-либо вложить деньги в производство фильма.

Какими должны быть спецификации для различных приложений? Рассмотрим несколько аналогий с работой фотографов и кинематографистов.

- ❑ **Моментальная фотография.** Некоторые приложения похожи на любительскую фотосъемку — незапланированный фотоснимок и не более. Конечно, получаемый при этом результат очень похож на быстрый фотоснимок, но иногда большего и не требуется. Минимальное приложение, такое как отображение текущей даты на Web-сайте, на самом деле не требует какой-либо письменной спецификации. Не долго думая, программист решит, где расположить такое приложение, и определенно ему не придется долго размышлять над созданием кода.

- **Портрет.** Даже небольшие, но завершенные приложения очень похожи на профессиональную фотографию — хорошо подобранное освещение, верно выбранная композиция и превосходное качество изображения. Такие приложения стоят дороже и требуют больше времени на разработку, а появляющийся на экране результат, цвет, гарнитура, начертание и размер текста продуманы заранее. В коде имеются ясные комментарии и соответствующим образом используются соглашения по кодированию (даже для очень небольших приложений). Чтобы определить все требования, возможно, будет достаточно одного или двух пунктов спецификации.
- **Реклама.** Рекламные ролики стоят гораздо дороже, они делаются профессионально и для них пишутся сценарии. Они во многом похожи на небольшие, но профессионально разработанные приложения для решения специфических задач. В них не много импровизации, и достаточно просто предопределить (и обеспечить в ходе производства) все аспекты процесса, так чтобы финальный продукт очень близко соответствовал исходному сценарию и целям производства. Такие приложения как системы подготовки и e-mail-рассылки новостей похожи на рекламные ролики — они весьма просты в проектировании и структуре. Можно еще до производства создать хорошую спецификацию и придерживаться ее без каких-либо значительных отклонений вплоть до завершения проекта.
- **Кинофильм.** Кинофильм подобен очень сложному приложению, поскольку несмотря на продуманный от начала и до конца сценарий окончательное действие может сильно отличаться от того, которое было запланировано первоначально. Это происходит потому, что окончательные решения принимаются после завершения съемки и зависят от реакции режиссера на то, что он видит на пленке. Это очень похоже на разработку сложного приложения, потому что часто разработчик не знает, что в созданном приложении хорошо, а что плохо (или не нужно), до тех пор, пока бета-тестеры или пользователи основательно не испытают приложение в работе. И как в случае любого дорогого проекта, чтобы сделать данное приложение правильно, может потребоваться множество изменений (с точки зрения пользователя “сделать приложение правильно” может в итоге означать “заставить его работать”).

Представленные аналогии дают понять, что для некоторых проектов требуется лишь незначительная спецификация (если она вообще нужна), другие же требуют гораздо более серьезного подхода. Но даже если хорошая спецификация создана еще до программирования, существует большая вероятность того, что приложение придется сильно изменять и переделывать по мере обретения им формы. Итак, следующий этап работы — процесс написания кода.

Процесс написания кода

Существует мнение о том, что если программировать все небольшие функции независимо, а потом объединить их, то получится готовое приложение. Идея в данном случае заключается в том, что если программист может определить функциональность каждой части приложения и запрограммировать эту функциональность, то впоследствии он сможет связать все эти части вместе и получить работающее приложение. В какой-то степени это верно, но (формально) работающее приложение отличается от превосходного приложения тем, как интегрированы составляющие части (насколько они эффективно и изящно работают вместе).

Большое значение имеет также простота, с которой другие люди обучаются использованию данного приложения. Большинство приложений предназначены для людей, а люди по-разному воспринимают и понимают слова, изображения, формы и т.д. Поэтому дизайн пользовательского интерфейса на самом деле больше является искусством, а не наукой.

Учитывая то, что существует множество инструментов, облегчающих быстрое создание рабочей модели приложения, не удивительно, что такие методики, как RAD и экстремальное программирование (Extreme Programming), столь популярны. Существует несколько определений RAD, однако, по существу, эта методика предполагает быструю разработку пользовательского интерфейса, решающего необходимые задачи, и демонстрацию этого интерфейса пользователям на ранней стадии цикла разработки. Это относительно недорогой процесс, и поскольку он многократно и быстро повторяется, разработчик за очень короткое время может отшлифовать дружелюбный по отношению к пользователю дизайн интерфейса. А как только появляется ясное понимание всех аспектов, необходимых для удовлетворения основных целей приложения, можно приложить усилия к правильной организации, документированию и оптимизации приложения.

Экстремальное программирование подобно RAD, но, как правило, предполагает совместную работу двух программистов, которые попеременно пишут код и могут ассистировать друг другу, замещая друг друга в перерывах на обед (или после увольнения одного из них), и, таким образом, ускоряя создание решения. “Одна голова хорошо, а две лучше” — экстремальное программирование на практике часто доказывает это реальными результатами.

Тестирование, отладка и сопровождение

Едва ли можно найти программу, которая была бы создана настолько хорошо, что не потребовала бы выпуска новых версий, изменений, обновлений или исправлений ошибок. Ошибки есть в большинстве приложений даже после выхода исправленных версий. На самом деле нет предела ошибкам, есть только надежда на то, что создатели программы со временем сделают ошибки настолько незначительными, что у большинства пользователей в самых распространенных обстоятельствах приложения будут работать правильно.

В дополнение к этому, постоянно изменяется среда, в которой работают приложения (включая протоколы операционной системы и другие приложения), а сообразительные хакеры постоянно находят новые нетривиальные способы обхода систем безопасности и взлома приложений. Наконец, сами пользователи предлагают внести в приложение новые функции.

В итоге цикл тестирование—отладка—усовершенствование на самом деле не заканчивается, но количество исправлений и изменений сокращается до контролируемого уровня, и вся эта работа превращается в сопровождение приложения. Избежать этой части процесса разработки невозможно, однако всегда существует возможность правильно ею управлять.

Оптимизация кода

Предположим, что существует 1000 Web-страниц, на каждой из которых необходимо отображать текущую дату и соответствующее ей приветственное сообщение. С помощью PHP можно было бы значительно сократить работу по обновлению данных

страниц, если посредством одной или двух строк РНР-кода заставить их генерировать даты и сообщения динамически, а не обновлять каждый день все 1000 страниц вручную.

К тому же необходимо найти способ написать код и в дальнейшем вставлять его во все страницы динамически, с тем чтобы в случае редактирования сообщения изменения распространялись сразу на все страницы. Эта широко распространенная проблема легко решается различными способами с помощью таких РНР-функций, как `include` и `require`.

Использование РНР и его функций, сокращающих работу человека, — один из способов оптимизации кода. Оптимизация не ограничивается лишь повторным использованием кода. Данное понятие также описывает множество методик, которые можно использовать для того, чтобы сделать код быстрее и эффективнее (чтобы он использовал меньше вычислительных ресурсов, таких как CPU-циклы, оперативная память или пространство жесткого диска), избежать переписывания кода или чтобы сделать программы проще в отладке и сопровождении.

В оставшейся части главы основное внимание уделяется подробному описанию практических приемов, которые помогают сэкономить время, а также сделать код более устойчивым. Эти приемы включают в себя формальные стандарты написания кода, написание программной логики в виде функций, а также использование РНР-функций `require()` и `require_once()`.

Использование стандартов написания кода

Программисты утверждают, что правильные отступы в коде для блоков ветвления (управляющих структур) значительно облегчают читаемость кода и, следовательно, упрощают его отладку и сопровождение. Однако фактически существует несколько вариантов отступов в коде, например, использование табуляции или нескольких пробелов. Обе методики в результате дают отступы в коде, который легко читать, но чтобы добиться согласованности, очевидно, следует использовать один и тот же подход на протяжении всего проекта, особенно когда проект разрабатывается совместно с другими программистами. Большинство программистов выработали при написании кода некоторые привычки. В этом нет ничего плохого, но иногда эти привычки приходится изменять, например, при переходе в другой коллектив, в котором используется отличающийся стиль написания кода.

Соглашения об использовании различных методик при написании программ (которые фактически не оказывают никакого влияния на правильную работу кода) можно классифицировать как *стандарты написания кода* (*coding standards*). Стандарты кодирования охватывают документацию и комментарии, отступы, соглашения об именовании, формат вызовов функций, подключение кода, расположенного в других файлах, и т.д.

В книге обсуждаются (и используются в примерах) соглашения PEAR (PHP Extension and Add-on Repository — репозиторий РНР-расширений и приложений). Написание кода с соблюдением данных стандартов делает код готовым для включения в репозиторий. (Несколько соглашений относятся к файлам классов; объекты и классы более подробно описываются в последнем разделе данной главы, а также в главах 12 и 13, изучение репозитория PEAR — тема главы 14). Рассмотрим основные PEAR-стандарты.

- **Отступы.** Использовать четыре пробела, а не табуляцию. Некоторые текстовые редакторы позволяют настроить клавишу Tab так, чтобы она вместо символа табуляции вставляла четыре пробела. Подобные настройки могут значительно

облегчить работу тем, кто привык использовать клавишу Tab для создания отступов. Стандарт написания кода PEAR также рекомендует ограничивать длину строк 75–85 символами, однако жесткого правила на этот счет нет.

- ❑ **Соглашения по именованию.** В стандарте кодирования PEAR имеются соглашения по именованию классов, функций и переменных.
 - ❑ Классы: рекомендуется использовать описательные имена, начинать имена с прописных букв и подразделять классы на категории, а имена категории и подкатегории разделять символом подчеркивания (например, `Cart_orderitem` для элемента заказа в контексте покупательской тележки).
 - ❑ Функции и методы: начинайте имя функции со строчной буквы, а каждое последующее слово в многословном имени функции — с прописной буквы. Например, имя `getData()` хорошо подходит для метода, который получает данные, а имя `calcTax()` — для функции, вычисляющей сумму налога. (Кроме того, для разделения слов в именах функций и методов можно также использовать символы подчеркивания, хотя этот способ именования не является стандартным для PEAR: например, `array_combine()`).

В данной книге методы еще не встречались; методы — это функции, но исключительно для объектов. Более подробно методы рассматриваются в главах, где обсуждаются объекты.

- ❑ Константы: рекомендуется использовать в именах констант символы верхнего регистра, а составляющие имя слова разделять символами подчеркивания.
- ❑ Глобальные переменные: начинать имя глобальной переменной рекомендуется с символа подчеркивания.
- ❑ Предопределенные значения: для таких значений, как `true`, `false` и `null`, рекомендуется использовать буквы нижнего регистра.
- ❑ **Формат вызовов функций.** Согласно данному стандарту между именем функции и открывающей скобкой, а также между открывающей скобкой и первым аргументом не должно быть пробела. После каждой запятой, отделяющей аргументы друг от друга, ставится один пробел:

```
$somevariable = myFunction($arg1, $arg2, $arg3);
```

В стандарте PEAR также указывается, что перед и после знака равенства там, где переменной присваивается значение или результат функции, следует ставить пробелы. Вообще разделение операторов пробелами облегчает чтение и понимание кода.

- ❑ **Формат структур управляющей логики.** Между началом оператора и условием следует использовать один пробел:

```
if (условие) {
    //другие операторы;
}
```

Следует всегда использовать фигурные скобки, даже там, где формально это необязательно.

- ❑ **Комментарии.** Для комментирования одиночных строк кода следует использовать двойную косую черту (`//`):

```
//это комментарий
```

Чтобы закомментировать блок кода, рекомендуется использовать символы `/*` и `*/`, например:

```
/*
несколько строк кода
несколько строк кода
несколько строк кода
несколько строк кода
*/
```

Для записи внутренних комментариев для объектов классов следует использовать стандарты PHPDoc.

- ❑ **Подключение кода.** Подробности подключения файлов, содержащих код (или текстовое наполнение), с помощью функций `include()` или `require()` рассматриваются далее в настоящей главе. Стандарт кодирования PEAR рекомендует использовать функцию `require_once()` для безусловного подключения файлов классов, а функцию `include_once` — для условного подключения таких файлов.
- ❑ **Разделители.** Используйте символы `<?php` в начале блока PHP-кода, а `?>` — в конце.
- ❑ **Определения функций.** При написании функций рекомендуется размещать имя функции и определение ее аргументов в одной строке, в следующей строке ставится открывающая фигурная скобка, а после всего кода функции ставится закрывающая фигурная скобка. Это соглашение называется *единственно правильной скобкой*. Кроме того, в определении функции любые аргументы, имеющие стандартные значения, следует размещать в конце перечня аргументов. Функции всегда следует программировать так, чтобы они возвращали выразительные значения (если они вообще должны возвращать какие-либо значения), например:

```
function myFunction($arg1, $arg2 = '')
{
    if (условие верно) {
        какие-то операторы;
    }
    return true;
}
```

- ❑ **Примеры URL.** В случае, когда требуется указать фиктивный URL (например, `http://www.fake_url.com`), используйте URL `example.com`, `example.net` и т.д., например: `http://www.example.com`.

Написание пользовательских функций в PHP

В PHP имеется множество встроенных функций, таких как `isset()`, `strlen()` и т.д. И хотя встроенные функции всегда решают очень полезные задачи, можно с уверенностью утверждать, что некоторых функций в PHP нет и не будет в последующих версиях. Предположим, что при разработке проекта понадобилась функция, которая отсутствует в PHP. Здесь на помощь приходят пользовательские функции, т.е. программист может создавать собственные функции и заставлять их решать любые задачи.

Любая создаваемая в PHP функция называется *определенной пользователем (user-defined)* или *пользовательской функцией*. Для того чтобы определить функцию, используется ключевое слово `function`. Сразу после определения функция становится доступной в любой точке кода и ее можно вызывать по имени. А возможность подключать файлы с кодом делает функции весьма мощным средством в арсенале программиста.

Функции позволяют инкапсулировать блоки кода, как будто они являются независимыми автономными программами. Можно передавать значения функциям и получать от них результаты. Можно вызывать функции в любой точке РНР-сценария и, таким образом, перенаправлять поток его выполнения.

Функции также вводят понятие области видимости (которое подробно обсуждается далее в настоящей главе) в программах. Обычно в РНР-программах имя и значение созданной переменной остаются постоянными до тех пор, пока это значение не изменится, либо пока сценарий не завершит свою работу. Однако при использовании функций необходимо понимать, что переменные могут существовать только внутри определенной функции, и за ее пределами они не содержат значений.

В этой главе рассматривается создание функций, работающих по тому же принципу, что и встроенные РНР-функции, которые уже обсуждались в книге: когда им передается какая-либо информация, они выполняют определенную операцию и возвращают ответ.

Структура функций

Функции представляют собой блоки кода, которые определяются пользователем для выполнения конкретных задач и имеют имя для обращения к ним. Известно, что функция может принимать входные значения, которые называются аргументами, выполнять определенные операции, а затем может возвращать другое значение (результат). Значения всех аргументов функция переносит в новые переменные, которые называются *параметрами* и обычно используются внутри самой функции. В примерах кода на страницах книги до сих пор очень часто использовались встроенные РНР-функции, предназначенные для разнообразных операций, например, для определения типа данных переменной (`gettype()`) или сортировки массивов (`sort()`). Написание пользовательской функции предполагает назначение этой функции придуманного имени, однако вызываются пользовательские функции так же, как и встроенные РНР-функции.

Рассмотрим поэтапно процесс работы с пользовательскими функциями.

1. Для создания функции необходимо написать ее имя с последующими круглыми скобками, в которых указываются имена принимаемых данной функцией значений (эти значения называются аргументами и являются необязательными), записанные в виде имен переменных (первым символом является знак доллара, как у обычных переменных) и отделенные друг от друга запятыми.
2. Чтобы вызвать функцию, необходимо написать ее имя и поставить круглые скобки. Если функция требует аргументов, то они должны быть записаны в круглых скобках и разделены запятыми. Передаваемыми аргументами могут быть переменные или константы, и они не обязательно должны быть такими же, как имена переменных в списке аргументов в определении функции.
3. Когда функция вызывается, переданные ей аргументы становятся *параметрами* и используются внутри функции для выполнения обработки данных. Параметры можно представить себе как ячейки для значений, передаваемых в функцию.
4. В функции может использоваться ключевое слово `return`, для того чтобы после завершения обработки данных функция передавала значение обратно вызывающему коду. Функции не обязательно должны возвращать значения; некоторые функции просто выполняют свою работу, а затем завершаются,

никак не уведомляя при этом вызывающий код. Однако хорошей практикой является возвращение, по крайней мере, значения true или false, которое позволяет сообщить остальному коду об успешном или неудачном завершении работы функции.

5. Значение, переданное из функции, можно вывести на экран с помощью оператора echo или присвоить какой-либо внешней по отношению к данной функции переменной.

Определение и вызов функций

Создание функции начинается с написания ключевого слова `function` с последующим пробелом и именем функции, по которому она будет вызываться. Если нужно, чтобы функция принимала какие-либо аргументы (это необязательно), то их необходимо перечислить в круглых скобках после имени функции. Затем необходимо поместить в фигурные скобки код, который выполняется при вызове функции. Код функции обрамляется фигурными скобками так же, как блоки кода в условных структурах. Ниже представлен синтаксис определения функции:

```
function functionname (parameters) {  
    //код функции  
}
```

Например, чтобы написать функцию для вычисления размера премии, можно использовать такой код:

```
function bonus($sales)  
{  
    $bonus = $sales * 0.15;    return $bonus;  
}
```

Обратите внимание на оператор `return` в данном примере. Если функция просто записывает информацию в базу данных, то, возможно, возвращаемого значения не будет. Однако если нужно, чтобы функция отправляла данные обратно, необходимо использовать оператор `return`, как показано выше. Например, так выглядит код для вызова только что определенной функции:

```
$my_total_sales = 120499;  
//вычислить размер премии на основе общего объема продаж  
$my_bonus = bonus($my_total_sales);
```

В этом примере возвращается результат обработки входного значения (которое в данном случае записано в переменную `$my_total_sales` и равно 120 499), поэтому переменной `$my_bonus` в итоге будет присвоено возвращаемое функцией значение.

Функция может содержать любое количество строк кода, а каждая строка может выполнять любую операцию в пределах возможностей PHP: подключаться к базе данных, открывать файл, проверять регулярное выражение и т.д. Кроме того, можно заставить функцию выводить на экран какой-либо текст с помощью операторов `echo` или `print`.

Сам по себе оператор `return` не отображает значение; он лишь передает результат (если он есть) из функции. Процесс выполнения содержимого функции внутри PHP-сценария называется *вызовом* функции.

Функции можно непосредственно передавать число:

```
echo (bonus(120499));
```

или имя уже созданной переменной:

```
$total_sales=120499;  
echo (bonus($total_sales));
```

Функции можно передать несколько параметров, но они должны быть отделены друг от друга с помощью запятой, например:

```
function bonus($total_sales, $bonus_factor)
{
    $bonus = $total_sales * $bonus_factor;
    return $bonus;
}
```

Теперь, чтобы вызывать модифицированную функцию расчета премии, необходимо передать ей два значения: `total_sales` и `bonus_factor`.

```
$total_sales = 120000;
$bonus_factor = 0.15;
echo (bonus($total_sales, $bonus_factor));
```

Функции можно определять в любом месте PHP-программы. Обычно функции определяют в отдельном файле и просто подключают его (с помощью функций `include` или `require`) в начале главного файла сценария. Определять или инициализировать переменные перед тем, как определяется функция, необязательно; однако необходимо убедиться, что они определены или проинициализированы перед *вызовом* функции.

Например, следующий код работает:

```
<?php
$total_sales = 190999;           // здесь начинается выполнение
$bonus_factor = 0.15;
echo (bonus($total_sales, $bonus_factor)); // здесь функция вызывается
function bonus($total_sales, $bonus_factor) // здесь функция определяется {
    $bonus = $total_sales * $bonus_factor;
    return $bonus;
}
?>
```

Но и представленный ниже код работает также хорошо:

```
<?php
function bonus($total_sales, $bonus_factor) // здесь функция определяется {
    $bonus = $total_sales * $bonus_factor;
    return $bonus;
}
$total_sales = 190999;           // здесь начинается выполнение
$bonus_factor = 0.15;
echo (bonus($total_sales, $bonus_factor)); // здесь функция вызывается
?>
```

Необходимо сделать важное замечание о работе функций. Чтобы передавать значения в функцию, можно использовать любые имена переменных. Например, предположим, что вместо использования одних и тех же имен (`$total_sales` и `$bonus_factor`) как вне, так и внутри функции, используются имена `$out_total_sales` и `$out_bonus_factor` для переменных, существующих за пределами функции, а внутри функции используются имена `$in_total_sales` и `$in_bonus_factor`. Такой код работал бы, и выглядел бы он следующим образом:

```
<?php
function bonus($in_total_sales, $in_bonus_factor) // здесь функция определяется {
    $bonus = $in_total_sales * $in_bonus_factor;
    return $bonus;
}
$out_total_sales = 190999;           // здесь начинается выполнение
$out_bonus_factor = 0.15;
echo (bonus($out_total_sales, $out_bonus_factor)); // здесь функция вызывается
?>
```

Рассмотрим параметры и их работу с другой стороны. Одна или несколько переменных, записанных в круглых скобках в определении функции, называются аргументами. Когда функция вызывается и ей передаются данные (как аргументы), значения в аргументах (независимо от того, как они поступают в функцию: как переменные или как постоянные значения) превращаются в параметры. Параметры подобны временным переменным внутри функции, и поскольку они обрабатываются внутри нее (и, возможно, во время обработки получают новые значения), они не выходят за пределы функции.

После того как результирующие значения обработанных параметров передаются за пределы функции, параметры аннулируются. То, что делают параметры (появляются при вызове функции, а затем исчезают после завершения обработки), никак не связано с какими-либо переменными, которые могут быть созданы внутри функции.

Прежде чем написать следующий пример работающей пользовательской функции, следует обобщить некоторые моменты:

- ❑ имена пользовательским функциям задает разработчик;
- ❑ функции могут принимать параметры, которые являются значениями или переменными, определенными в скобках после имени функции;
- ❑ параметры отделяются друг от друга запятыми;
- ❑ код функции записывается в фигурных скобках после имени функции и параметров;
- ❑ внутри функции необходимо использовать ключевое слово `return` для возвращения значения, которое можно использовать за пределами функции;
- ❑ если значение возвращать не требуется, то ключевое слово `return` обозначает конец кода функции;
- ❑ функции не выполняются до тех пор, пока они не вызваны в какой-либо строке PHP-сценария;
- ❑ вызывать функцию можно как до, так и после ее объявления в коде — месторасположение определения функции не имеет значения;
- ❑ вызывать функцию можно столько раз, сколько необходимо, и передавать в функцию любые значения, отправляя их как постоянные значения или как значения внутри каких-либо переменных.

Последний пункт в этом перечне дает ключ к повторному использованию кода. Тот факт, что нет необходимости переписывать выполняемый функцией код каждый раз, когда требуется его функциональность, значительно сокращает усилия по написанию и отладке кода.

Рассмотрим пример кода `holiday.php` из главы 4 и напомним функцию для вычисления стоимости недельного отдыха.

Практика Использование простой функции

1. Введите следующий код и сохраните его как `holiday3.php` (чтобы сэкономить время, можно использовать ранее написанные блоки кода):

```
<html>
<head><title></title></head>
<body>
<b>Бронирование гостиниц Namllu</b>
<?php
```

```

function calculator($price, $city_modifier, $star_modifier)
{
    return $price = $price * $city_modifier * $star_modifier;
}
if (isset($_POST['posted'])) {
    $price = 500;
    $star_modifier = 1;
    $city_modifier = 1;
    $destgrade = $_POST['destination'].$_POST['grade'];
    switch($destgrade) {
        case "Barcelonathree";
            $city_modifier = 2;
            break;
        case "Barcelonafour";
            $city_modifier = 2;
            $star_modifier = 2;
            break;
        case "Viennathree";
            $city_modifier = 3.5;
            break;
        case "Viennafour";
            $city_modifier = 3.5;
            $star_modifier = 2;
            break;
        case "Praguethree";
            break;
        case "Praguefour";
            $star_modifier = 2;
            break;
        default;
            $city_modifier = 0;
            echo ("Пожалуйста, вернитесь и начните сначала");
            break;
    }
    if ($city_modifier <> 0)
    {
        echo "Недельная стоимость проживания в $_POST[destination] - " . "$" .
            calculator($price, $city_modifier, $star_modifier);
    }
}
?>
<form method="POST" action="holiday3.php">
<input type="hidden" name="posted" value="true">
Где Вы хотели бы провести отпуск?
<br>
<br>
<input name="destination" type="radio" value="Prague">
Прага
<br>
<input name="destination" type="radio" value="Barcelona">
Барселона
<br>
<input name="destination" type="radio" value="Vienna">
Вена
<br>
<br>
В гостинице какого класса Вы хотели бы остановиться?
<br>
<br>
<input name="grade" type="radio" value="three">
Трехзвездочная
<br>
<input name="grade" type="radio" value="four">
Четырехзвездочная

```

```

<br>
<br>
<input type="submit" value="Отправить запрос">
</form>
</body>
</html>

```

2. Сохраните и закройте файл.
3. Откройте страницу holiday3.php (рис. 6.1) в браузере, сделайте выбор и нажмите кнопку Отправить запрос.

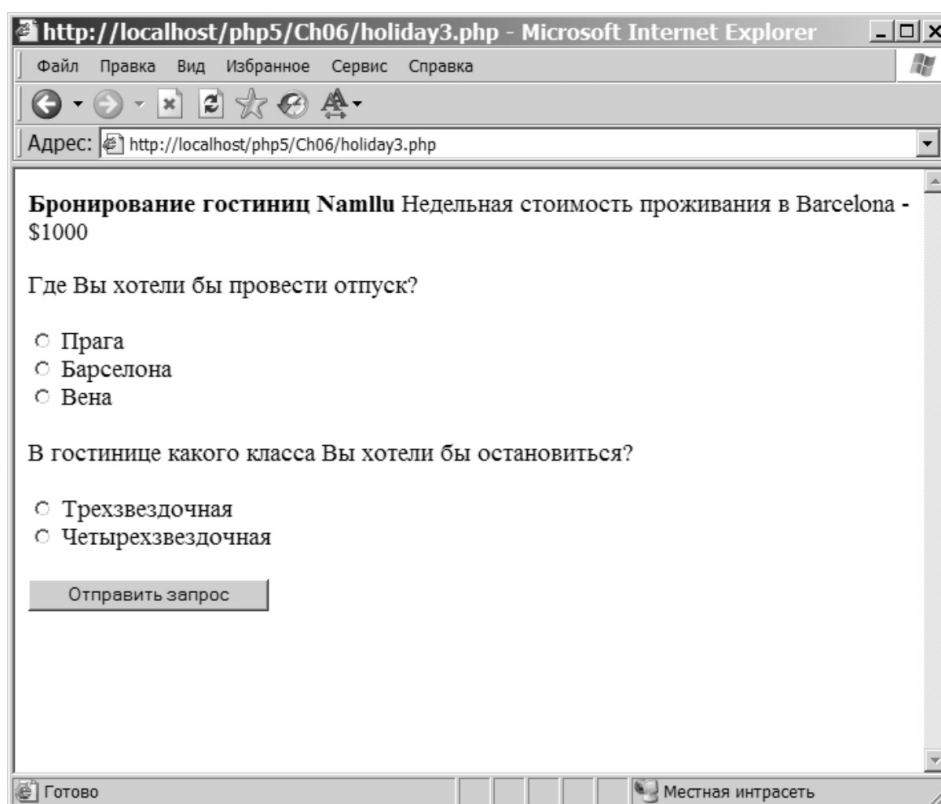


Рис. 6.1.

Как это работает

Данная форма аналогична форме в главе 4 — она принимает от пользователя два значения и передает их посредством \$_POST-переменных (\$_POST[destination] и \$_POST[grade]) PHP-сценарию. PHP-код начинается с определения функции (calculator) для вычисления стоимости отпуска:

```

function calculator($price, $city_modifier, $star_modifier)
{
    return $price = $price * $city_modifier * $star_modifier;
}

```

Пользовательская функция `calculator()` принимает три параметра: базовую стоимость отдыха, коэффициент наценки для выбранного города и коэффициент для класса гостиницы. Затем функция перемножает эти переменные и возвращает значение (`$price`) — окончательную стоимость отдыха в гостинице.

Остальная часть программы работает так же, как и раньше, объединяя пункт назначения и класс гостиницы в одно значение, которое используется для выбора различных вариантов. Есть, однако, одно небольшое изменение в `case`-блоке по умолчанию: переменной `$city_modifier` присваивается значение 0, если ни один из вариантов не выбран. Перед вызовом функции `calculator()` выполняется проверка данной переменной. Если ее значение не равно нулю, то функцию можно вызывать внутри оператора `echo()`, добавляя к результату функции текст с помощью оператора конкатенации (`.`):

```
if ($city_modifier <> 0)
{
    echo "Недельная стоимость проживания в $ _POST[destination] - " . "$" .
    calculator($price, $city_modifier, $star_modifier);
}
```

Функции переключения

Можно написать функцию, которая бы принимала входящие значения и на основании этих значений выбирала бы другие функции для обработки данных, так же, как это делается в PHP-программе с помощью структуры `switch`. Это весьма эффективная особенность пользовательских функций в PHP.

Рассмотрим небольшой пример. Предположим, что необходимо создать приложение, которое выбирает записи из различных таблиц базы данных. С помощью переменной `$table_name` в функцию переключения можно передать какое-либо входное значение, а функция на основании этого значения примет решение, какую функцию с SQL-запросом необходимо выполнить:

```
function query_switch($table_name) {
    switch ($table_name){
        case "clients":
            $query = query_clients_gen();
            break;
        case "orders":
            $query = query_orders_gen();
            break;
        case "employees":
            $query = query_employees_gen();
            break;
        default:
            echo "Выберите имя таблицы";
    }
}

function query_client_gen()
{
    //формирование запроса к базе данных для выбора записей из
    //таблицы clients
}

function query_orders_gen()
{
    //формирование запроса к базе данных для выбора записей из
    //таблицы orders
}

function query_employees_gen()
```

```
{
    //формирование запроса к базе данных для выбора записей из
    //таблицы employees
}
```

Особенность использования такой структуры заключается в том, что созданные в данном случае функции можно поместить в отдельный файл и подключать его (с помощью функции `include`) к любой странице, что значительно упрощает формирование запросов на различных страницах. В дополнение к этому, в случае усовершенствования данных функций код придется модифицировать только в одном файле, кроме того, такой подход облегчает чтение кода. На самом деле, как только станет ясно, что функции делают именно то, что они должны делать по замыслу разработчика, читать их во время кодирования уже необязательно, достаточно просто вызывать их при необходимости.

Поскольку результат функции (которая возвращает значение) можно непосредственно присвоить переменной, этот результат можно использовать в операторе `switch...case`. Например, предположим, что требуется создать небольшую управляющую функцию, которая показывает количество посещений определенной страницы. Количество посещений, возвращаемое функцией, можно использовать в операторе `echo` и вывести пользователю понятное сообщение, как показано в следующем примере:

```
switch ($hit_counter = get_hits($current_page)){
    case $hit_counter <100:
        echo "Сегодня мало посещений";
        break;
    case $hit_counter <1000:
        echo "Сегодня много посещений";
        break;
    case $hit_counter <10000:
        echo "Сегодня очень много посещений";
        break;
}
```

В переменной `$hit_counter` можно хранить любое значение, возвращаемое функцией `get_hits($current_page)` (в которой переменной `$current_page` присвоено имя файла текущей страницы и которая получает от какого-либо сценария общее число посещений за текущий день). Можно присваивать значение переменной `$hit_counter` непосредственно в скобках оператора `switch()`.

Чтобы применить эту функцию, ее можно было бы установить в каждой странице и запускать каждый раз, когда отображается страница. Пользователю немедленно выводилось бы число посещений этой страницы (хотя некоторые пользователи могли бы многократно обновлять страницу, чтобы повысить количество посещений).

Как значения попадают в функции

Как уже отмечалось, параметры являются ячейками, представляющими передаваемые в функцию значения. Существует два способа передачи данных в параметр, так чтобы функция могла его обработать. Данные можно передавать в функцию либо *по значению*, либо *по ссылке*. Различие между этими способами едва уловимое, но очень важное.

Данные, передаваемые в функцию по значению, непосредственно присваиваются параметру. При передаче данных по ссылке параметр непосредственно связывается с внешней по отношению к функции переменной, заставляя эту переменную принять любое значение, которое получит параметр.

Передача по значению

Во все функции, которые до сих пор рассматривались в этой книге, аргументы передаются по значению. Значение передается в функцию путем ввода этого значения (буквально или путем ввода имени содержащей его переменной) в зарезервированное для этого значения место (в скобках, следующих за именем функции).

Например, в первом примере функции для вычисления премии общий объем продаж можно передать как число, введя его непосредственно в качестве аргумента, или присвоить переменной значение, равное этому числу, а затем передать в качестве аргумента имя этой переменной. В следующем примере демонстрируются оба метода передачи аргументов по значению:

```
function bonus($total_sales) {
    $bonus = $total_sales * 0.15;
    return $bonus;
}
$total_sales = 120499;
echo "Сумма премии " . bonus($total_sales);
echo "Общий объем продаж " . $total_sales;
echo "Общий объем продаж равен \$120 499, а сумма премии равна " .
bonus(120499);
```

Следует отметить, что даже после того, как значение `$total_sales` было установлено за пределами функции, а затем было передано ей для обработки, внешнее значение переменной `$total_sales` не изменилось.

Передача по ссылке

Передача значений в функцию по ссылке имеет совершенно другой эффект. Внешняя переменная, содержащая передаваемое в функцию значение, фактически изменяется при выполнении обработки. Она изменяется потому, что на самом деле в функцию передается не значение, а ссылка, которая указывает непосредственно на внешнюю переменную. Поэтому все операции, имеющие место внутри функции, влияют на внешнюю переменную.

Как это реализовать в коде? В определении функции перед именем аргумента, передаваемого по ссылке, необходимо поставить амперсанд (&). В приведенном ниже примере показана передача аргумента `$salary`.

Предположим, что премия прибавляется к значению переменной `$salary`. Очевидно, это можно было бы сделать за пределами функции, но проще и понятнее сделать это внутри функции. Рассмотрим следующий код:

```
function bonus($total_sales,&$salary)
{
    $bonus = $total_sales * 0.15;
    $salary = $salary + $bonus;
    return $salary;
}
$total_sales = 120499;
$salary = 40000;
bonus($total_sales,$salary);
echo "Ваша заработная плата, включая премию равна " . $salary;
echo "Объем продаж равен " . $total_sales;
```

В данном случае нет необходимости присваивать какой-либо переменной результирующее значение, возвращаемое функцией `bonus()`; переменной `$salary` присваивается значение 40000, а затем переменная передается по ссылке в функцию `bonus()`, и, когда функция завершает свою работу, она возвращает значение непосредственно в переменную `$salary`. Очень лаконично.

Установка значений по умолчанию

Как и во многих других языках программирования и связанных с компьютерами технологиях, в PHP можно писать функции так, чтобы они имели стандартные значения своих параметров. Например, можно определить аргумент как имеющий значение по умолчанию. Это значение впоследствии во время вызова функции будет передано параметру при условии, что в момент вызова функции пользователь не присваивал данному аргументу никакого другого значения.

Однако есть одна особенность: необходимо определять все аргументы со значениями по умолчанию справа от аргументов, не имеющих стандартных значений. Эта особенность не очень заметна, но если ее проигнорировать, то написанная функция будет работать не так, как должна. В следующем примере определяется функция для расчета премии. Значение по умолчанию для зарплаты (\$40 000) устанавливается справа от аргумента `total_sales`:

```
function bonus($total_sales, &$salary = 40000)
{
    $bonus = $total_sales * 0.15;
    $salary = $salary + $bonus;
    return $salary;
}
$total_sales = 120499;
bonus($total_sales);
echo "Ваша зарплата, включая премию, равна " . $salary;
echo "Ваш общий объем продаж равен " . $total_sales;
```

Это означает, что если при вызове функции ей не было передано какое-либо другое значение для аргумента `$salary`, то функция автоматически будет выполнять вычисления для значения 40000. Однако если какое-либо значение было передано функции (неважно какое: больше, меньше или равное 40000), то функция для вычислений будет использовать именно переданное значение.

Важность порядка аргументов

При вызове PHP-функций (пользовательских или встроенных) важен порядок передачи значений аргументов. Если опустить необязательный аргумент(-ы) в конце списка, то можно отбросить последнюю запятую, разделяющую аргументы. Кроме того, если опустить аргумент, то функция автоматически подставит для этого аргумента значение по умолчанию (если оно указано в определении функции) и продолжит обработку. Предупреждение или сообщение об ошибке появится только тогда, когда будет пропущен обязательный аргумент (аргумент без стандартного значения), однако уровень предупреждения или сообщения об ошибке будет зависеть от используемой версии PHP, а также, возможно, от настроек вывода сообщений об ошибках.

Область видимости переменных

Продолжим начатое в главе 2 обсуждение темы области видимости переменных. В PHP-сценарии можно создавать переменные как внешние (вне функции), так и внутренние (внутри функции) по отношению к функции. Если переменная `$total_sales` создается за пределами кода функции, а затем ее значение передается в функцию, которая обрабатывает свою внутреннюю переменную с таким же именем, то в сценарии будет две разные переменные с именем `$total_sales`. Очевидно, такой сценарий будет работать правильно, только если переменные не будут влиять друг на друга. Поэтому

PHP (как и многие другие языки программирования) использует понятие области видимости и различает подобные переменные.

Можно представить себе множество ситуаций, когда удобно использовать в одном сценарии несколько переменных с одинаковыми именами. Область видимости позволяет PHP обрабатывать две переменные с одним и тем же именем, пока они остаются в соответствующих блоках кода. (Во многом таким же образом пространства имен позволяют работать XML-элементам с одинаковыми именами внутри одного XML-документа и оставаться при этом разделенными. Пространства имен XML рассматриваются в главе 8.)

Кроме области видимости, созданные в PHP-сценарии переменные также имеют время жизни. Переменная, созданная внутри функции, существует только до тех пор, пока функция не закончит свою работу. Если эта переменная не определена как *статическая* (подробнее статические переменные рассматриваются в этой главе далее), то по окончании работы функции переменная теряется. Таким образом, область видимости переменной в данном случае является функция, а время жизни переменной равно времени работы функции (каждый раз при вызове функции создается новая внутренняя переменная и она имеет новое время жизни). Все переменные в PHP-сценарии уничтожаются, когда сценарий завершает свою работу.

Глобальные и локальные переменные

Переменные, созданные вне функций, существуют до конца работы сценария и, следовательно, имеют глобальную область видимости, т.е. доступны из любой точки сценария. Однако чтобы получить к ним доступ из какой-либо функции, необходимо использовать ключевое слово `global`. Наличие ключевого слова `global` перед именем переменной означает, что переменную можно использовать внутри функции (но создать другую переменную с тем же именем нельзя, потому что это имя принято также внутри функции). Переменные, созданные внутри функции, имеют локальную область видимости.

Рассмотрим пример работы локальных и глобальных переменных:

```
<?php
$global_message = "Глобальное сообщение";
function addto_message($global_message)
{
    global $global_message;
    $global_message .= " и немного больше";
    return $global_message; }
addto_message($global_message);
echo "Глобальное сообщение - это " . $global_message;
?>
```

Код выводит на экран предложение: “Глобальное сообщение — это Глобальное сообщение и немного больше”, так как вызываемая функция изменяет значение переменной `$global_message`. Это происходит потому, что используется ключевое слово `global`; изменение переменной таким способом очень похоже на передачу аргумента в функцию по значению.

Фактически использование ключевого слова `global` перед именем переменной внутри функции создает ссылку на суперглобальный массив с именем `$_GLOBALS`. В данном случае создание переменной `$global_message` (или любой другой переменной в данном контексте) за пределами функции (и, следовательно, с глобальной областью видимости) автоматически делает ее частью массива `$_GLOBALS`. Итак, можно выделить два способа доступа к глобальным переменным внутри функций: использование ключевого слова `global` (например, так: `global $global_message`) и использование суперглобального массива (`$_GLOBALS[$_GLOBALS[global_message]]`).

Создание статических переменных в функциях

В главе 2 уже отмечалось, что создание внутри функции переменной с предшествующим ключевым словом `static` позволяет переменной существовать после того, как функция завершила свою работу. Поскольку статическая переменная существует только в локальной области видимости функции, то и переменная, и ее последнее значение будут доступны при следующем вызове данной функции. Это может оказаться полезным для подсчета или измерения каких-либо значений внутри функции.

Предположим, например, что необходимо каждый раз подсчитывать извлекаемые из базы данных записи. Для этого можно использовать такой код:

```
<?php
function get_records($table)
{
    //подключение к базе данных
    //извлечение записей
    //get_record_count
    static $record_count;
    //присвоить $record_count количество возвращенных записей
    //плюс текущее значение данной переменной
    echo "текущее количество выбранных записей равно " . $record_count;
    //вернуть какое-либо значение
}
```

Хотя представленный пример почти полностью состоит из псевдокода, он показывает, как создавать статическую переменную `$record_count`. Должно быть очевидно, что при каждом вызове данной функции (по крайней мере, в пределах выполнения одного РНР-сценария) значение переменной `$record_count` будет увеличиваться на количество выбранных записей. Например, если в пределах одного РНР-сценария в первый раз было выбрано 20 записей, во второй 30, а в последний раз 40, то переменной `$record_count` будет присвоено значение 90. Естественно, можно каждый раз выводить текущее количество записей, что и делается в данном примере.

Обобщим знания о глобальных, локальных и статических переменных:

- ❑ глобальные переменные существуют на протяжении работы всей программы, но чтобы их использовать внутри функции, необходимо предварять их имена ключевым словом `global` или обращаться к ним посредством суперглобального массива `$_GLOBALS`;
- ❑ локальные переменные содержат значения, которые существуют только внутри функции и только во время ее работы;
- ❑ статические переменные являются локальными по отношению к функции переменными, значения которых сохраняются между вызовами данной функции.

Рассмотрим пример небольшого приложения, иллюстрирующий использование глобальных, локальных и статических переменных.

Практика Демонстрация области видимости

1. Откройте редактор Web-страниц и введите следующий код:

```
<html>
<head><title></title></head>
<body>
```

```

<font size=-1>
<?php
$global_message = "Глобальное сообщение";
function my_function()
{
    $local_message = "Локальное сообщение";
    static $static_number = 0;
    echo "<br>содержимое глобального сообщения: " . $GLOBALS["global_message"];
    echo "<br>содержимое локального сообщения: " . $local_message;
    echo "<br>статическое число: " . $static_number;
    return $static_number = $static_number+1;
}
echo "<b>Первый вызов функции:</b>";

my_function();

echo "<br><br><b>За пределами функции:</b>";
echo "<br>содержимое глобального сообщения: " . $global_message;
echo "<br>содержимое локального сообщения: " . $local_message;
echo "<br>статическое число " . $static_number;
echo "<br><br><b>Второй вызов функции:</b>";

my_function();
echo "<br><br><b>За пределами функции:</b>";
echo "<br>содержимое глобального сообщения: " . $global_message;
echo "<br>содержимое локального сообщения: " . $local_message;
echo "<br>статическое число: " . $static_number;
echo "<br><br><b>Третий вызов функции:</b>";

my_function();
echo "<br><br><b>За пределами функции:</b>";
echo "<br>содержимое глобального сообщения: " . $global_message;
echo "<br>содержимое локального сообщения: " . $local_message;
echo "<br>статическое число " . $static_number;
?>
</font>
</body>
</html>

```

2. Сохраните данный файл как `score.php` и закройте его.
3. Откройте браузер и вызовите страницу `score.php` (рис. 6.2).

Как это работает

Представленная выше программа не имеет практического применения, но учитывая то, что тема области видимости переменных весьма сложна для понимания, этот пример акцентирует внимание на работе различных типов переменных в PHP. В первой строке создается и инициализируется переменная `$global_message`:

```
$global_message = "Глобальное сообщение";
```

Затем определяется функция `my_function`:

```
function my_function()
{
```

Внутри функции создаются локальная переменная `$local_message` и статическая переменная `$static_number`:

```
$local_message = "Локальное сообщение";
static $static_number = 0;
```

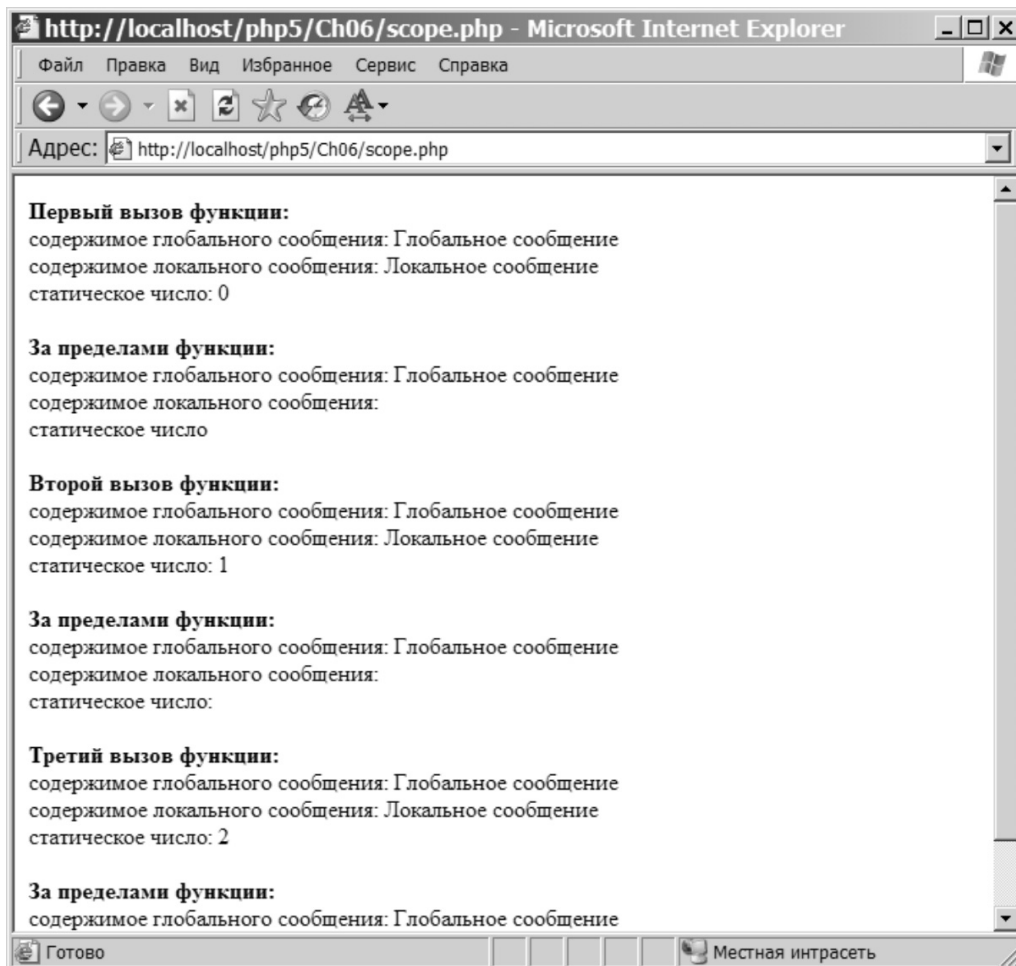


Рис. 6.2.

Следующие три строки отображают значения глобальной, локальной и статической переменных внутри функции:

```

echo "<br>содержимое глобального сообщения: " . $GLOBALS["global_message"];
echo "<br>содержимое локального сообщения: " . $local_message;
echo "<br>статическое число: " . $static_number;

```

Оператор `return` изменяет значение `$static_number`, увеличивая его на 1:

```

return $static_number = $static_number+1;
}

```

Программа начинает свою работу с вызова функции `my_function()`:

```

echo "<b>Первый вызов функции:</b>";
my_function();

```

Отображаются значения глобальной, локальной и статической переменных, которые внутри функции равны "Глобальное сообщение", "Локальное сообщение" и 0.

Как только функция заканчивает работу, выводится содержимое данных переменных вне функции. Здесь в зависимости от уровня отображения сообщений об ошибках возможно появление предупреждений о неопределенных переменных — это нормально. Переменные `$local_message` и `$static_number` определены только внутри кода функции, поэтому вне его они не существуют:

```
echo "<br><br><b>За пределами функции:</b>";
echo "<br>содержимое глобального сообщения: " . $global_message;
echo "<br>содержимое локального сообщения: " . $local_message;
echo "<br>статическое число " . $static_number;
```

Данная функция вызывается еще несколько раз, и каждый раз показывает значения своих внутренних переменных, а также значения переменных за ее пределами. Внутри функции изменяется только переменная `$static_number`.

Вложенность функций

Возможность создавать и вызывать функции внутри других функций называется *вложенностью функций* (*nesting*). Однако в большинстве обстоятельств вложенность функций не особенно полезна. Возможность вызова функций из других функций удобна, в книге уже приводились соответствующие примеры, и все же определение и вызов функции внутри другой функции может привести к возникновению проблем.

Например, можно написать родительскую функцию `parent`, внутри которой определяется другая функция (назовем ее `child`). Проблема заключается в том, что если вызвать функцию `parent` дважды, то интерпретатор дважды попытается определить функцию `child`, а так как переопределять функции нельзя, сценарий аварийно завершится. В большинстве случаев лучше определять функции отдельно.

Однако возможна другая ситуация (кроме использования функции переключения, которая рассматривалась выше), когда необходимо вызвать функцию из функции: рекурсия (функция вызывает сама себя).

Рекурсия

Вызов функции из самой себя называется *рекурсией* (*recursion*). Рекурсивный вызов функции создает циклические структуры, в которых функция продолжает вызывать себя до тех пор, пока удовлетворяется определенное условие. Однако условие необходимо явно создать внутри кода функции. Если нет уверенности, что существует какое-либо стоп-значение, которое так или иначе будет достигнуто, то рекурсия может длиться бесконечно. Рассмотрим небольшой пример.

Практика Рекурсивный вызов функции

1. Запустите редактор Web-страниц и введите следующий код:

```
<html>
<head></head>
<body>
<?php

if ($_POST['posted']) {

    $num = $_POST['num'];
    function recursion($num) {
```

```
        if ($num <= 1) {
            return 1;
        } else {
            return $num * recursion($num-1);
        }
    }
    echo "Факториал " . $num . " равен " . (recursion($num));
}
?>
<form method="POST" action="recursion.php">
<input type="hidden" name="posted" value="true">
Найти факториал числа
<input name="num" type="text">
<br>
<br>
<input type="submit" value="Рассчитать факториал">
</form>
</body>
</html>
```

2. Сохраните файл как `recursion.php` и закройте его.
3. Откройте страницу `recursion.php` в браузере (рис. 6.3).

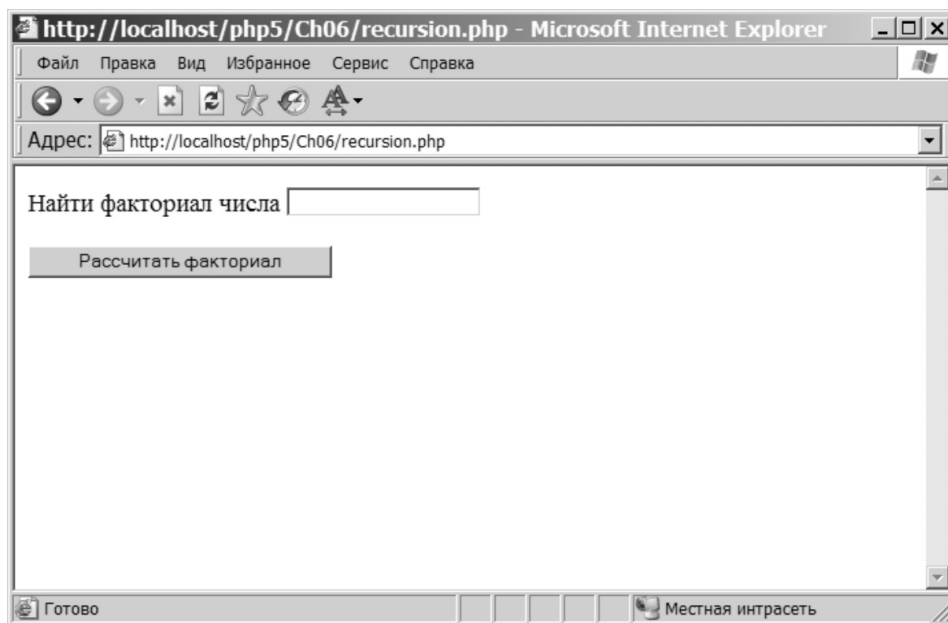


Рис. 6.3.

Теперь, если выбрать значение и нажать кнопку **Рассчитать факториал**, то добросовестный код с рекурсией выдаст ответ. Пример показан на рис. 6.4.

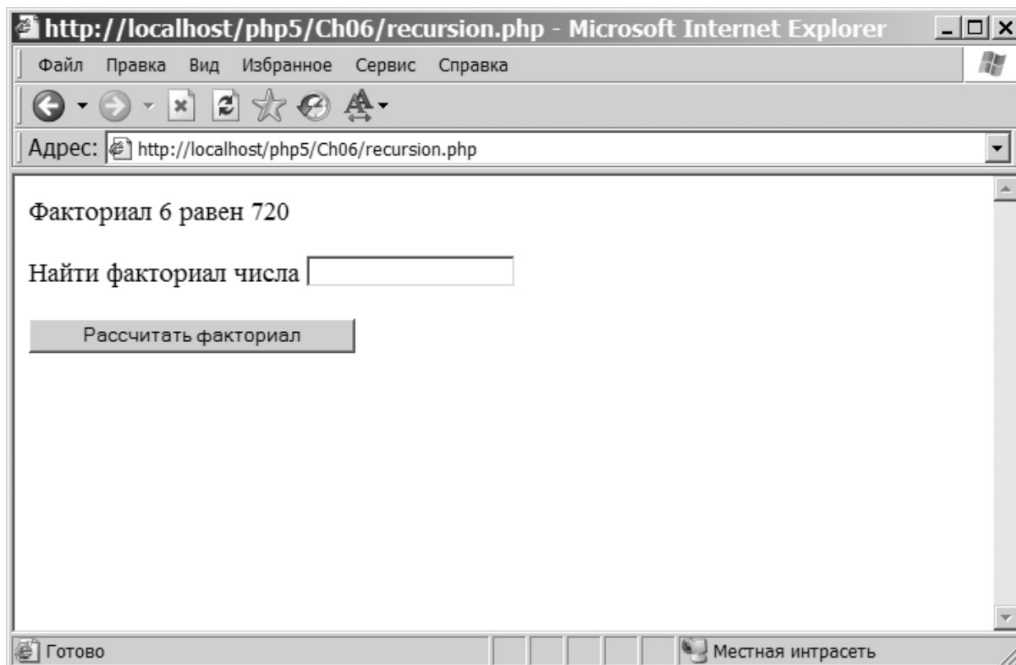


Рис. 6.4.

Как это работает

Сначала объявляется рекурсивная функция, которой затем в качестве аргумента передается переменная `$num_value`:

```
<?php
if ($_POST['posted']) {
    $num = $_POST['num'];
    function recursion($num) {
        if ($num <= 1) {
            return 1;
        } else {
            return $num * recursion($num-1);
        }
    }
    echo "Факториал " . $num . " равен " . (recursion($num));
}
?>
```

Затем необходимо убедиться, что значение `$num` больше единицы; если это не так, то возвращается 1 (факториал от 1, как известно, равен 1). Если введенное число больше единицы, то выполняется рекурсивная часть функции. В конце концов, выводится результат.

В каждой итерации цикла значение не только умножается на последнее полученное число (рекурсивная часть), но и постоянно уменьшается на единицу, чтобы гарантировать, что цикл, в конце концов, завершится. Этот сценарий не подходит для вычисления факториалов больших чисел, кроме того, он не работает с отрицательными числами, но в качестве примера его работы вполне достаточно.

Операторы `include` и `require`

Еще один способ облегчить использование и сопровождение кода заключается в подключении файлов (подключаемые файлы уже несколько раз использовались в примерах кода). Подключать внешние файлы к текущему сценарию и запускать их очень легко. Все, что требуется сделать, это использовать конструкцию `include` или `require` (это именно языковые конструкции, а не функции, несмотря на то, что иногда они называются функциями) с именем файла, который необходимо подключить. Единственное различие между двумя этими конструкциями заключается в том, как они обрабатывают неудачное подключение внешнего файла. Сбой `include` приводит к появлению предупреждения, а сбой `require` — к критической ошибке.

Можно отметить некоторые особенности работы данных конструкций.

- ❑ Файл подключается путем вставки одной из этих конструкций в РНР-код, но синтаксический анализ основного файла в этой точке переходит из РНР-режима в HTML-режим, поэтому чтобы запускать код подключаемого файла, этот код все равно необходимо помещать между соответствующими РНР-разделителями.
- ❑ С помощью одной из этих конструкций можно подключать файл изнутри какой-либо функции, но в таком случае любые переменные, созданные в коде подключаемого файла, будут иметь локальную по отношению к данной функции область действия.
- ❑ РНР-сценарий внутри подключаемого файла выполняется при подключении. Любые РНР-переменные и их значения доступны в главном файле, как будто весь код подключаемого файла непосредственно вставлен в главный файл.

Рассмотрим несколько примеров использования данных конструкций в коде (можно использовать полные абсолютные или относительные пути):

```
include("filename")
require("filename");
```

Например, чтобы подключить с помощью `include` файл `test.txt`, необходимо использовать такой код:

```
include("test.txt");
```

Если файл `test.txt` содержит какой-либо текст, то этот текст будет включен в Web-страницу так, как если бы он был непосредственно записан в ее исходном коде.

Примечательно то, что в круглых скобках функций `include` или `require` можно также использовать имена переменных или присоединять переменные к имени файла с тем, чтобы сформировать имя существующего файла:

```
$Name = "1";
include("test" . $Name . ".txt");
```

В результате этой конкатенации создается имя `test1.txt`, и если файл с таким именем существует, значит он будет подключен, в противном случае генерируется ошибка.

Ниже перечислены ситуации, в которых подключаемые файлы удобны:

- ❑ подключение текстового файла к странице;
- ❑ определение переменных и/или констант, а также текста некоторых сообщений об ошибках;

- ☐ вставка в страницу значений HTTP-переменных;
- ☐ выполнение отдельного PHP-сценария (даже если он ничего не возвращает);
- ☐ подключение часто используемых функций, которые нежелательно определять в каждой странице.

Подключаемые файлы удобны потому, что их можно непосредственно добавлять к файлу и в его коде определять необходимость подключения того или иного файла. Эта возможность рассматривается в следующем примере.

Практика Условное подключение

1. Запустите редактор Web-страниц и введите следующий текст:

Подключен первый файл

2. Сохраните созданный файл как `file1.txt`.
3. Закройте его, создайте новый текстовый файл и введите в него следующий текст:

Подключен второй файл

4. Сохраните данный файл как `file2.txt`.
5. Закройте данный файл, создайте новый и введите в него следующий код:

```
<html>
<head><title></title></head>
<body>
<?php
if (isset($_POST['posted'])) {
    $choice = $_POST['choice'];
    if ($choice <> "") {
        include("file" . $choice . ".txt");
        echo "<hr>";
    }
}
?>
<form method="POST" action="include.php">
<input type="hidden" name="posted" value="true">
Выберите подключаемый файл?
<select name="choice">
<option value="1">первый файл</option>
<option value="2">второй файл</option>
</select>
<br>
<br>
<input type="submit" value="Вывести текст">
</form>
</body>
</html>
```

6. Сохраните этот файл как `include.php` в том же каталоге, что и первые два текстовых файла.
7. Откройте в браузере страницу `include.php` и выберите файл из списка. На рис. 6.5 показан примерный результат.

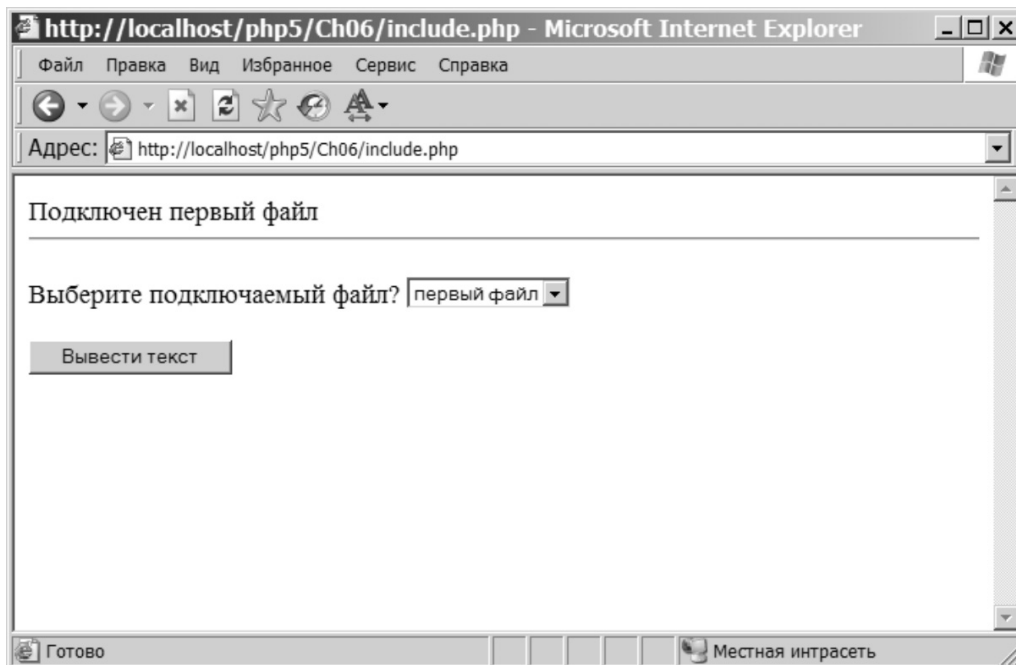


Рис. 6.5.

Как это работает

Сначала выводится небольшая форма, предоставляющая пользователю выпадающий список (с именем `choice`) с двумя вариантами. Следующий PHP-код при выполнении проверяет, какое значение было выбрано (если выбран, например, первый пункт, то значение равно 1) и использует это значение для формирования имени подключаемого файла.

```
<?php
if (isset($_POST['posted'])) {
    $choice = $_POST['choice'];
    if ($choice <> "") {
        include("file" . $choice . ".txt");
        echo "<hr>";
    }
}
?>
```

Естественно, должны существовать текстовые файлы `file1.txt` и `file2.txt`, содержащие какой-либо текст (в противном случае появится сообщение об ошибке).

О чем следует позаботиться при использовании подключаемых файлов

Многие Web-серверы не настроены на синтаксический анализ файлов с расширением `.inc`, поэтому лучше задавать для подключаемых файлов расширение `.php` — в таком случае никто не сможет просмотреть через Web-браузер содержимое этих

файлов (возможно, секретное). Например, зная (или лишь догадываясь), что на сайте имеются `.inc`-файлы (возможно, содержащие секретную информацию, такую как пароли к базе данных), злоумышленник может просто ввести в своем браузере полный URL-указатель на такой файл и просмотреть его необработанный исходный код. Рекомендуется всегда задавать для подключаемых файлов расширение `.php`, тогда они безусловно будут обрабатываться PHP-интерпретатором.

Если требуется подключить один файл через другой подключаемый файл, например, через `common.inc.php`, который подключается вверху каждой страницы (это весьма распространенная Практика), но есть вероятность, что некоторые функции могут случайно определяться дважды, то следует использовать операторы `include_once` или `require_once`. Они работают так же, как `include` и `require`, но всегда подключают файл только однажды независимо от того, сколько раз файлы подключаются в коде.

Резюме

Функции представляют собой блоки кода, которые можно вызывать внутри программы. Они либо определены в PHP автоматически, как, например, функции обработки массивов `sort()` и `ksort()`, которые описывались в предыдущих главах, либо создаются пользователями (пользовательские функции), как было показано в данной главе. Функции можно представить как небольшие “черные ящики”, которые возвращают значение в ответ на переданные им соответствующие переменные. Переменные передаются в функции через аргументы, определяемые в начале функции. Если переменные передаются не по ссылке, то значение, возвращаемое функцией, не влияет на переменные вне данной функции. Возвращаемое функцией значение можно использовать для переключения между различными блоками кода и выполнения различных действий.

В примерах данной главы было показано, что переменные ведут себя по-разному внутри функций. Эти различия вызваны областью видимости переменных. Переменные с локальной областью видимости существуют только во время работы функции и в конце ее работы уничтожаются, если они не были определены с использованием ключевого слова `static`. Переменные, определенные за пределами функций, внутри функций должны использоваться либо с ключевым словом `global`, либо через массив `$_GLOBALS`. В этой главе рассматривались подключаемые файлы, функции `include()`, `require()`, `include_once()` и `require_once()`, а также их использование для разбиения кода на модули.

Упражнение

Напишите функцию, с помощью которой пользователь сможет создать собственную Web-форму. В форме должны быть предусмотрены возможности выбора имен полей и типов полей, отправки информации в PHP-страницу и просмотра переданных значений. Для выбора различных функций внутри главной функции необходимо использовать оператор `switch..case`.

7

Файлы и каталоги

Вряд ли пользователям хотелось бы посещать Web-сайты, код которых мог бы получить контроль над жестким диском пользовательского компьютера и делать все, что понадобилось разработчику сайта, например, записывать и стирать файлы и каталоги. По этой причине создатели языка JavaScript оставили эти возможности за рамками данного языка. РНР не имеет возможности записывать и стирать файлы с жесткого диска пользователя, но может создавать и удалять файлы и каталоги на Web-сервере. Это очень полезная возможность, потому что часто самый простой и наиболее эффективный способ реализации в приложении определенной функциональности заключается в использовании файловой системы сервера.

Файлы хранятся в каталогах на жестком диске, и поскольку в них информация сохраняется после выключения или перезагрузки машины, они в отличие от временного хранилища, такого как оперативная память, представляют собой *постоянный* механизм хранения данных. Каталоги также являются файлами, но файлами особого типа, предназначенными для хранения других файлов. Каталоги создаются иерархически внутри других каталогов, начиная с корневого каталога.

Файлы могут содержать любые данные, а также информацию о самих себе, например, имя владельца файла и дату создания. РНР облегчает работу с файловой системой, предоставляя ряд функций, которые позволяют получать информацию о файлах, открывать их, а также осуществлять считывание информации из файлов и запись в них. Эта глава посвящена РНР-функциям для работы с файловой системой. Здесь описываются РНР-функции чтения и записи файлов, открытие и закрытие файлов с помощью функций `fopen()` и `fclose()`, переименование и удаление файлов, навигация в файлах и чтение каталогов. Кроме того, в данной главе рассматривается получение информации о правах доступа к файлам и пользователям файлов, а также реализация загрузки файлов на сервер через Web-интерфейс.

Существует множество различных типов файловых систем, хотя, возможно, пользователям, привыкшим к Windows, не очень знакомы другие системы. Большинство Windows-систем предоставляют пользователю свободный доступ к файлам и каталогам,

тогда как Linux-системы обычно ограничивают доступ к файлам и каталогам путем обеспечения различных прав доступа для пользователей в системе. Работа файловой системы NTFS, которую можно использовать в Windows NT/2000 и более старших версиях, больше похожа на работу файловых систем в Linux-машинах — она также позволяет предоставлять разным пользователям различные права доступа.

Для PHP-разработчика весьма важно знать, используются ли в системе права доступа к файлам и как эти права устанавливаются, потому что некоторые функции требуют установки определенных прав, прежде чем они смогут выполнить свою работу. К счастью, PHP предоставляет функции для проверки и установки прав доступа, а также для проверки привилегий текущего пользователя. PHP обычно запускается от имени того же пользователя, что и Apache (на Linux/Apache-серверах), или с привилегиями, предоставленными гостевой учетной записью в PS-системах.

Обработка файлов и каталогов

Вся информация на жестком диске хранится в файлах того или иного типа, хотя большинство пользователей оперирует понятиями файлов и каталогов. Существуют обычные программные файлы, файлы данных, файлы, которые являются каталогами, и специальные файлы, которые позволяют жесткому диску отслеживать содержимое каталогов и файлов. В PHP имеются встроенные функции, которые способны обрабатывать любые существующие файлы, но чаще всего приходится работать с текстовыми файлами, содержащими какие-либо данные.

Файл — не что иное, как упорядоченная последовательность байтов, хранящаяся на жестком диске, дискете, CD-ROM или другом носителе. Каталог представляет собой специальный тип файла, в котором содержатся имена других файлов и каталогов (иногда называемых подкаталогами) и указатели на область их хранения на жестком диске. Для того чтобы манипулировать файлами, требуется знать, как сценарии подключаются к файлам.

Между операционными системами Linux и Windows существует множество различий, например способ указания путей к каталогам. В Unix-подобных системах, таких как Linux, для разделения элементов пути используются символы косой черты, например:

```
/home/dan/data/data.txt
```

В Windows используются символы обратной косой черты:

```
C:\MyDocs\data\data.txt
```

К счастью, PHP в Windows в большинстве ситуаций автоматически конвертирует косую черту в обратную косую черту, например, код

```
$fp = fopen("/data/data.txt", "r");
```

не должен создавать каких-либо проблем на Windows-платформе. В некоторых случаях необходимо использовать путь непосредственно (например, когда загруженный на сервер файл копируется из временного каталога в какой-либо другой каталог). В таких ситуациях использование обратной косой черты обязательно. Поскольку PHP интерпретирует обратную косую черту как escape-символ (экранирующий следующий за ним символ), путь необходимо задавать следующим образом:

```
"C:\\MyDocs\\data\\data.txt"
```

Позднее будет показан простой способ автоматического конвертирования косой черты в обратную косую черту.

Работа с файлами

РНР предоставляет две группы связанных с файлами функций, которые отличаются способом обработки файлов: первая группа функций использует *дескриптор файла* (*file handle*) или, как его часто называют, *указатель файла* (*file pointer*); вторая группа использует непосредственно строковое *имя файла*. То же касается РНР-функций, связанных с обработкой каталогов.

Указатель файла представляет собой целое число, используемое для идентификации файла, с которым необходимо работать до тех пор, пока файл не будет закрыт. Если открывается несколько файлов, то каждый из них идентифицируется по его собственному уникально назначенному дескриптору.

Дескриптор файла содержится в переменной (имя которой создается так же, как имя любой другой РНР-переменной; в приведенных здесь примерах эта переменная обычно имеет имя `$fp`). Целое значение, содержащееся в данной переменной, идентифицирует соединение с тем файлом, с которым в данный момент работает сценарий. Например, как только с помощью функции `fopen()` открывается какой-либо файл, появляется возможность записывать в этот файл данные, используя функцию `fwrite()`. Функции `fwrite()` требуется дескриптор файла (в данном случае `$fp`), указывающий на файл, с которым функция должна работать:

```
fopen($fp, "myfile.txt", "w+");  
fwrite($fp, 'Hello world!');
```

В данном случае `$fp` — переменная, представляющая дескриптор файла (в нее записывается целое значение, когда используется функция `fopen()` для открытия указанного в первой строке файла). Функция `fwrite()` записывает в представленный данным дескриптором файл текст, содержащийся в ее втором аргументе (вторая строка примера).

С другой стороны, функция `file()`, которая используется для считывания данных из файла, принимает строковый аргумент, содержащий путь к данному файлу:

```
$lines = file('./data.txt');
```

В следующих разделах эти новые функции описываются подробнее.

Все упомянутые в данной главе функции возвращают значение `True`, если заданная операция выполнена успешно, и `False` — в противном случае.

Рассмотрим функции для работы с файлами.

Открытие и закрытие файлов

Работа с файлами обычно состоит из трех этапов:

1. Открытие файла и связывание с ним дескриптора.
2. Чтение данных из файла или запись данных в файл с помощью дескриптора.
3. Закрытие файла с помощью дескриптора.

Функция `fopen()`

Функция `fopen()` используется для открытия файла и возвращает дескриптор, связанный с открытым файлом. Она принимает два или три аргумента: имя файла, режим и необязательный параметр `use_include_path`.

Дескриптор файла помогает определить, был ли данный файл успешно открыт (если это так, то дескриптор файла представляет собой положительное целое число, в противном случае он равен нулю). Операции с файлами часто вызывают ошибки, поэтому всегда следует принимать во внимание, что необходимая операция может быть выполнена неверно. Хорошей практикой является использование процедуры для обработки ошибок, которая в случае возникновения ошибки (например, сценарий не имеет необходимых прав для доступа к данному файлу или файл вообще не существует) завершит работу сценария аккуратно и желательно с подходящим сообщением. Например, учитывая то, как PHP интерпретирует целые числа в контексте булевых операций, можно использовать тот факт, что значение дескриптора файла преобразуется в `True` в случае успешного завершения операции и в `False` — в случае ошибки. Это позволяет проверить открытие файла с помощью следующего кода:

```
$fp = fopen("./data.txt", "r");
if(!$fp) die ("Невозможно открыть файл");
```

Этот код при желании можно переписать так:

```
if(!$fp = fopen("./data.txt", "r")) die ("Невозможно открыть файл");
```

В данном случае проверяется не равенство `$fp` результату вызова функции `fopen()` (в таком случае использовался бы оператор равенства `==`), а лишь успешность открытия файла. Это сокращенный способ проверки, который позволяет в случае успешного выполнения операции сохранить дескриптор файла в переменной `$fp` для последующего использования. Используйте тот код, который покажется удобнее.

Первый аргумент функции `fopen()` определяет имя файла, который требуется открыть. Значением этого аргумента может быть имя файла, относительный (например, `./data.inc`) или абсолютный (`/myfiles/data.inc`) путь к файлу. (Что произойдет, если указать только имя файла, например, `data.txt`, будет показано далее при рассмотрении аргумента `use_include_path`.) Данная функция также позволяет указать файл на удаленном узле и открывает его через HTTP или FTP URL, как показано в примере ниже:

```
if(!$fp = fopen("http://www.whatyoumaycallit.com/index.html", "r"))
    die ("Невозможно открыть файл.");

if(!$fp = fopen("ftp://ftp.whatyoumaycallit.com/pub/index.txt", "r"))
    die ("Невозможно открыть файл.");
```

Удаленный файл можно открыть только для чтения — его невозможно модифицировать.

Нотация относительного пути может запутать тех, кто не знаком с файловыми операциями в командной строке Unix или Windows. В относительном пути к файлу точка (.) означает текущий каталог, а две точки (..) — родительский каталог. Например, путь `./data.txt` указывает на файл `data.txt` в том же каталоге, что и сценарий, а путь `../data.txt` указывает на файл `data.txt`, находящийся в каталоге более высокого уровня. Путь `../../../data.txt` указывает на файл `data.txt`, который расположен в каталоге тремя уровнями выше каталога сценария. Абсолютный путь отличается от относительного тем, что начинается с косой черты (/), которая указывает на то, что путь задан относительно корневого каталога файловой системы, а не относительно расположения сценария. Например, запись `C:/Inetpub/wwwroot/index.php` представляет собой абсолютный путь.

Второй аргумент функции `fopen()` (именованный режим) определяет то, как будет использоваться открываемый файл. Задав одно из следующих значений, можно открыть файл только для чтения, для чтения и записи или только для записи.

Значение	Описание
<code>r</code>	Открытие файла только для чтения. Указатель позиции в файле помещается в начало файла
<code>r+</code>	Открытие файла для чтения и записи. Указатель позиции в файле помещается в начало файла
<code>w</code>	Открытие файла только для записи. Любые имеющиеся в файле данные будут утеряны. Если файл не существует, то PHP попытается его создать
<code>w+</code>	Открытие файла для чтения и записи. Любые имеющиеся в файле данные будут утеряны. Если файл не существует, то PHP попытается его создать
<code>a</code>	Открытие файла только для дополнения. Данные записываются в конец существующего файла. Если файл не существует, то PHP попытается его создать
<code>a+</code>	Открытие файла для чтения и дополнения. Данные записываются в конец существующего файла. Если файл не существует, то PHP попытается его создать

Указатель позиции в файле представляет собой внутренний указатель, определяющий в файле точную позицию, в которой должна быть выполнена следующая операция.

Аргумент режима в функции `fopen()` может также принимать значение `b`, которое указывает на то, что открытый файл необходимо интерпретировать как двоичный. В таких платформах, как Linux, которые интерпретируют текстовые и двоичные файлы одинаково, это значение не играет роли, однако в Windows оно может оказаться полезным, потому что Windows (а также Apple Macintosh) по-разному устанавливает символы конца строки для текстовых и двоичных файлов (в Windows используется последовательность CRLF, в Macintosh — CR, а в Linux — LF). По умолчанию в функции `fopen()` (начиная с версии 4.3.2 PHP) используется бинарный режим для всех платформ. Это означает, что преобразование символа конца строки не выполняется. Рекомендуется использовать значение `b` как часть аргумента режима, чтобы код мог использовать те символы конца строки, которые подходят для платформы, на которой он выполняется.

Третий (необязательный) аргумент функции `fopen()` называется `use_include_path` (пути для подключаемых файлов). Если его значение равно 1 и указано только имя файла, но не путь к нему, то функция будет искать данный файл сначала в том каталоге, где расположен сценарий, а затем в каталогах, указанных в параметре `include_path` файла `php.ini`.

Пути для подключаемых файлов особенно полезно указывать, если необходимо задать каталог, в котором расположены подключаемые файлы, часто используемые сценариями сайта. Предположим, что параметру `include_path` файла `php.ini` присвоено значение `/home/apache/inc` и выполняется следующий вызов функции:

```
fopen("data.txt", "r", 1);
```

Если в таком случае функция не сможет найти файл `data.txt` в текущем каталоге, то она продолжит поиск файла в каталоге `/home/apache/inc`.

Функция `fclose()`

Как только работа с файлом закончена, файл необходимо закрыть. Это можно сделать с помощью функции `fclose()`, передав ей в качестве единственного аргумента дескриптор открытого файла:

```
fclose ($fp)
```

Несмотря на то, что РНР должен по завершении работы сценария автоматически закрывать все открытые файлы, хорошей практикой является закрытие файлов из сценария сразу после того, как работа с ними закончена. При этом файлы быстрее освобождаются для использования другими процессами или даже для других запросов в том же сценарии.

Получение информации о файле

Файлы содержат множество сведений о самих себе, например, размер, дату изменения, имя владельца и т.д. В РНР имеется функция `stat()`, позволяющая получить информацию о файле. Для этого ей необходимо передать в качестве аргумента имя файла.

Функция `stat()` возвращает индексированный массив, содержащий статистическую информацию о файле. В возвращаемом функцией массиве имеется 13 элементов:

Индекс массива	Ассоциативное имя	Информация
0	Dev	Номер устройства
1	Ino	Номер индексного узла (inode)
2	Mode	Режим защиты индексного узла
3	nlink	Количество ссылок
4	uid	Пользовательский идентификатор владельца файла (Userid)
5	gid	Идентификатор группы владельца файла
6	rdev	Тип устройства (в случае inode-устройства)
7	size	Размер в байтах
8	atime	Время последнего доступа к файлу
9	mtime	Время последней модификации содержимого файла
10	ctime	Время последнего изменения inode-информации файла
11	blksize	Размер блока ввода-вывода файловой системы
12	blocks	Количество используемых блоков

Следующий фрагмент кода показывает, как использовать функцию `stat()` для получения размера файла в байтах (предполагается, что файл `myfile.txt` существует):

```
$my_file_stats = stat("myfile.txt");
$my_file_size = $my_file_stats[7];
```

В Windows и Macintosh доступна не вся информация, предоставляемая этой функцией. В частности, если используется Web-сервер Personal Web Server на платформе Windows 98, то в массиве, возвращаемом `stat()`, не будет идентификатора группы и идентификатора пользователя, потому что в данной системе они не используются. Подробнее эта тема освещается в данной главе в разделе “Принадлежность и права доступа к файлам”.

Чтение и запись файлов

Рассмотрим несколько РНР-функций, которые можно использовать для чтения данных из файла и записи данных в файл. Несмотря на то, что они весьма просты, их можно использовать вместе, чтобы реализовать элементарный работающий сценарий.

Функция `fread()`

Функцию `fread()` можно использовать для получения строки символов из файла. Функция принимает два аргумента: дескриптор файла `$fp` и целое число `$length`. Функция считывает из файла, указанного посредством дескриптора `$fp`, `$length` байтов и возвращает их в виде строки. Предположим, например, что открывается файл:

```
$fp = fopen("data.txt", "r")
```

а затем используется следующий вызов:

```
$data = fread($fp, 10);
```

Функция `fread()` получит первые 10 байтов из файла `data.txt` и запишет их в переменную `$data` как строку символов.

Функция перемещает указатель позиции в файле на 10 байтов вниз, поэтому, если повторить тот же вызов `fread()`, то в переменную `$data` будут записаны следующие 10 байтов файла. Если в файле осталось менее 10 неп прочитанных байтов, то функция прочтает и возвратит столько байтов, сколько осталось.

Функция `fwrite()`

Функцию `fwrite()` можно использовать для записи данных в файл. Эта функция принимает два обязательных аргумента: дескриптор файла (`$fp`) и строку (строку текста, которая будет записана в файл), после чего записывает содержимое строки в файл, заданный дескриптором. При этом функция возвращает количество записанных байтов (или `-1` в случае ошибки). Например, после открытия файла с помощью вызова `$fp = fopen("data.txt", "w")` можно использовать такой код:

```
fwrite($fp, "ABCxyz");
```

В результате в начало файла `data.txt` будет записана строка `"ABCxyz"`. Поскольку при открытии файла использовался режим только для записи, вся информация, которая была в файле до этого, будет утеряна (а если файл не существовал, то будет создан новый файл `data.txt`); однако повторение данного вызова приведет к тому, что в конец файла будут записаны те же шесть байтов, и в файле будет записана строка `"ABCxyzABCxyz"`. Это также связано с работой указателя позиции в файле.

Если указать в качестве третьего аргумента целое значение `length`, то функция прекратит работу сразу после записи `length` байтов (предполагается, что это случится раньше, чем будет достигнут конец строки). Например, код

```
fwrite($fp, "abcdefghij", 4);
```

запишет в файл, заданный `$fp`, первые четыре байта строки `"abcdefghij"` (т.е. `"abcd"`). Если строка содержит менее чем `length` байтов, то она будет записана в файл полностью.

Рассмотрим практический пример.

Практика Простой счетчик посещений

Очень распространенной разновидностью Web-сценариев является счетчик посещений, который используется для того, чтобы показывать на Web-странице количество

просмотров данной страницы и, следовательно, ее популярность. Существует множество различных форм счетчиков посещений, простейшая из которых — текстовый счетчик. Ниже приведен пример простого сценария-счетчика.

```
<?php
//hit_counter01.php
$counter_file = "./count.dat";
if(!($fp = fopen($counter_file, "r"))){
    die ("Невозможно открыть файл $counter_file.");
}
$counter = (int) fread($fp, 20);
fclose($fp);

$counter++;

echo "Вы - посетитель № $counter.";

$fp = fopen($counter_file, "w");
fwrite($fp, $counter);
fclose($fp);
?>
```

Сохраните данный код как `hit_counter01.php` и вызовите эту страницу в браузере. Пример ее работы показан на рис. 7.1.

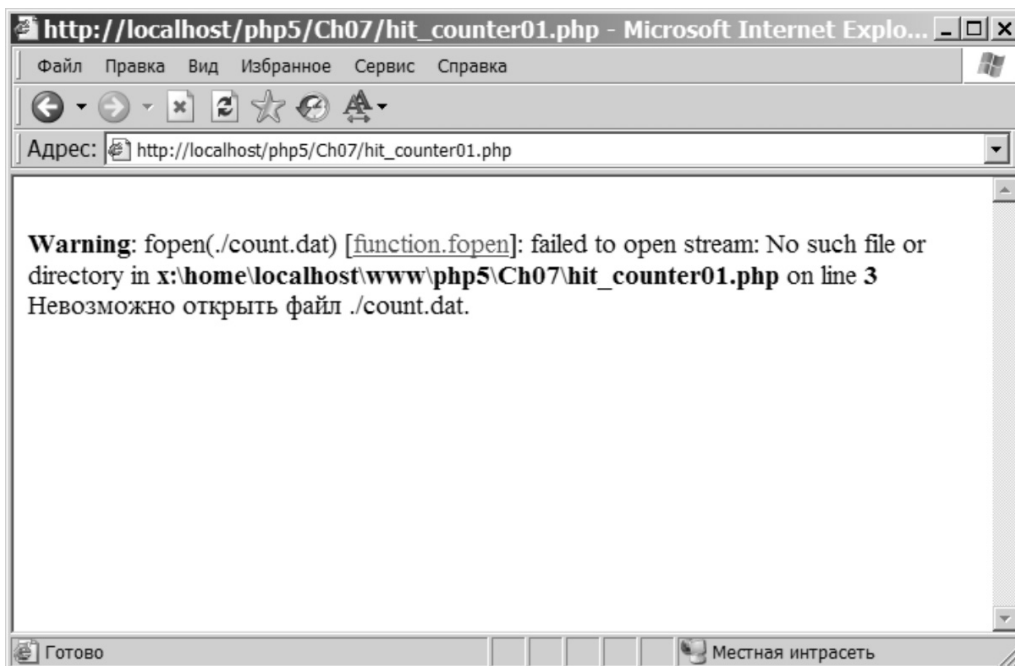


Рис. 7.1.

При запуске сценария возникает ошибка. Это не удивительно. Сценарий предполагает, что файл `count.dat` существует и расположен в текущем каталоге, т.е. в каталоге, из которого запускается сценарий. Если такого файла в данном каталоге нет, то возникает ошибка и сценарий прекращает работу, отображая на экране сообщение об ошибке.

Создать файл можно путем сохранения пустого текстового файла в текстовом редакторе (например, в Notepad в Windows; не забудьте выбрать пункт **Все файлы** в списке **Тип файлов**, в противном случае к имени файла будет добавлено расширение `.txt`) или в Unix-подобной системе, используя команду `touch`:

```
> touch count.dat
```

После создания файла можно снова проверить работу сценария. Примерный результат показан на рис. 7.2.

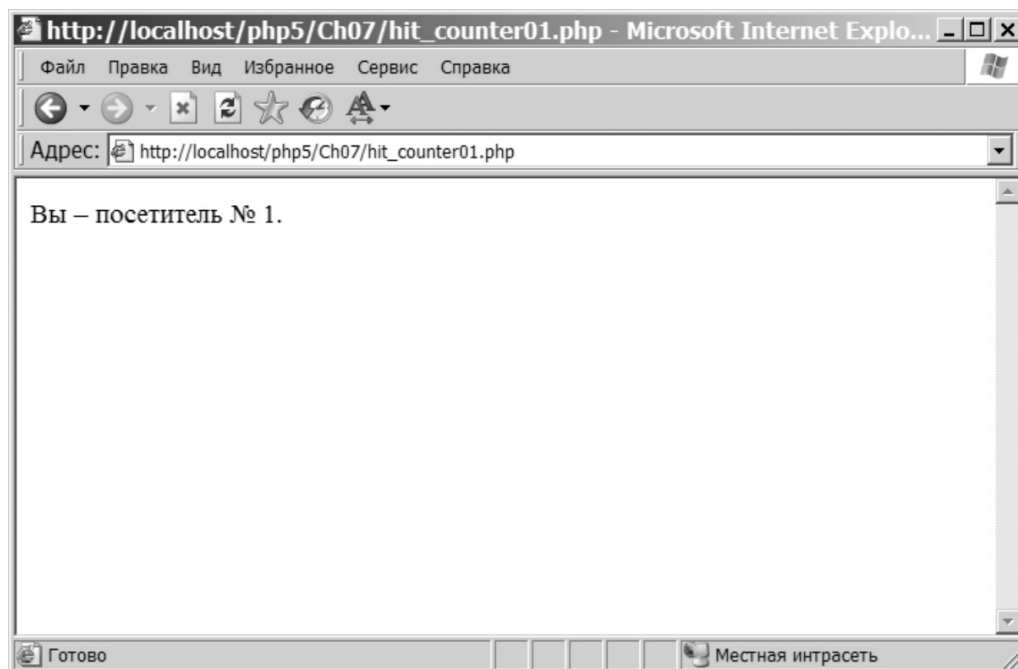


Рис. 7.2.

При желании в файле `count.dat` можно задать начальное количество посещений. Если данный сценарий используется на Linux-платформе, то следует убедиться, что Web-сервер имеет права на чтение и запись данного файла.

Как это работает

Прежде всего, файл `count.dat` (в текущем каталоге) открывается функцией `fopen()` в режиме только для чтения:

```
$counter_file = "./count.dat";  
if(!$fp = fopen($counter_file, "r")){  
    die ("Невозможно открыть файл $counter_file.");  
}
```

Возвращенный функцией `fopen()` дескриптор файла присваивается переменной `$fp`, которую после этого можно использовать для обращения к данному открытому файлу. Функция возвращает значение `False` в случае ошибки, поэтому если заданный файл не существует или во время его открытия возникла какая-либо ошибка, то сценарий завершится с соответствующим сообщением об ошибке.

Затем сценарий использует дескриптор файла для считывания числа посещений из открытого файла. Как видно из кода, сценарий вызывает функцию `fread()` для считывания 20 байтов из файла данных (20 байтов достаточно для хранения, как минимум, одного миллиона посещений):

```
$counter = (int) fread($fp, 20);
```

Так как функция `fread()` возвращает строковое значение, последнее число посещений, считанное из файла, необходимо конвертировать в целое значение, поэтому применяется функция преобразования типа `(int)`.

Затем функция `fclose()` закрывает файл, связанный с дескриптором `$fp`, записывая все не записанные данные в файл:

```
fclose($fp);
```

После закрытия файла сценарий увеличивает счетчик и сообщает посетителю о количестве посещений данной страницы:

```
$counter++;  
echo "Вы - посетитель № $counter.";
```

Теперь необходимо сохранить увеличенное значение счетчика в файл данных, иначе он будет всегда оставаться неизменным. Для этого файл с помощью функции `fopen()` открывается снова, но на этот раз в режиме только для записи, т.е. в режиме, который переписывает существующее содержимое файла.

```
$fp = fopen($counter_file, "w");  
fwrite($fp, $counter);  
fclose($fp);
```

Вызов функции `fwrite()` с переменной `$counter` в качестве второго аргумента гарантирует точную длину строки, записываемой в файл, поэтому аргумент длины указывать необязательно. Затем файл закрывается с помощью функции `fclose()`.

Можно подойти к проблеме подсчета посетителей творчески и создать графический счетчик. Простой тестовый счетчик, используемый совместно с набором изображений, можно превратить в графический счетчик. Например, если для каждой цифры в счетчике имеется изображение (файлы `1.gif`, `2.gif`, `3.gif` и т.д.), то каждое численное значение можно использовать для указания графического файла, связанного с данной цифрой:

```
<?php  
//hit_counter02.php  
$counter_file = "./count.dat";  
$image_dir = "./images";  
if(!($fp = fopen($counter_file, "r"))){  
    die ("Невозможно открыть файл $counter_file.");  
}  
$counter = (int) fread($fp, 20);  
fclose($fp);  
  
$counter++;  
  
for ($i=0; $i < strlen($counter); $i++) {  
    $image_src = $image_dir . "/" . substr($counter, $i, 1) . ".gif";  
    $image_tag_str .= "<img src= \"$image_src\" border=\"0\">";  
}  
  
echo "Вы - посетитель № $image_tag_str";  
  
$fp = fopen($counter_file, "w");  
fwrite($fp, $counter);  
fclose($fp);  
?>
```

В этом сценарии (назовем его `hit_counter02.php`) добавлен цикл `for` для обработки возможных значений переменной `$counter` и связывания каждой цифры с соответствующим изображением. Цикл обеспечивает обработку каждой цифры, проверяя длину строки, содержащейся в переменной `$counter`. На рис. 7.3 показан примерный результат запуска сценария.

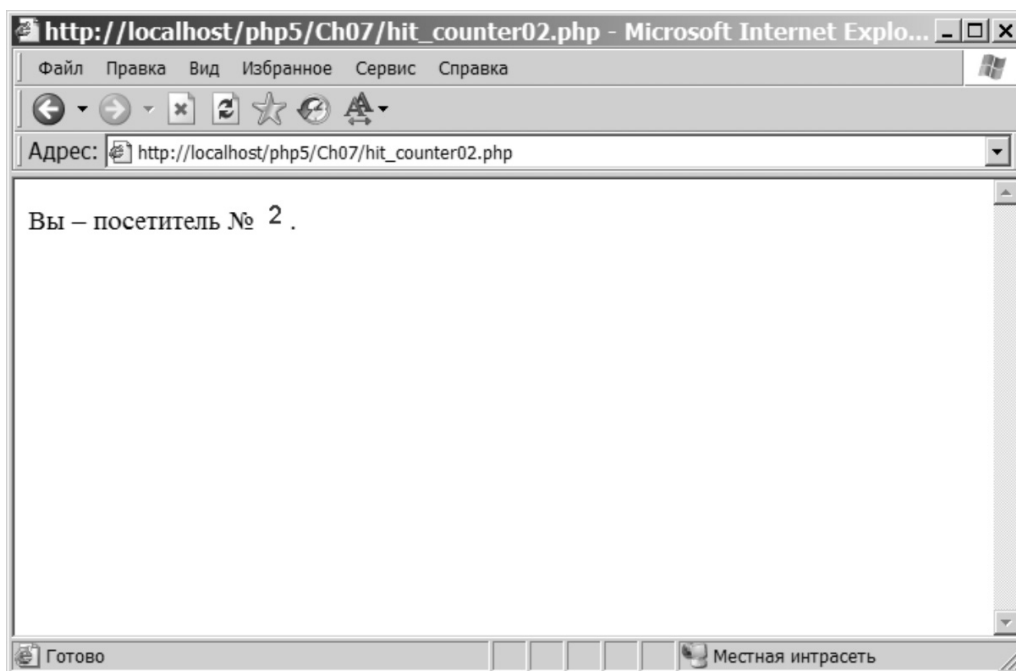


Рис. 7.3.

Чтение и запись символов в файлы

Предположим, что требуется считать весь файл или проанализировать его по одному символу за раз. Обе эти задачи можно легко решить с помощью функции `fread()`, например:

```
do { $one_char = fread($fp, 1); $counter .= $one_char; } while ($one_char);
```

Однако РНР предоставляет несколько функций, которые позволяют решить эти задачи гораздо проще:

- ☐ `fgetc()`
- ☐ `ffeof()`
- ☐ `fgets()`
- ☐ `fgetcsv()`
- ☐ `fputs()`

Функцию `fgetc()` можно использовать для считывания из файла по одному символу за раз. Функция `fgetc()` принимает один аргумент, дескриптор файла `$fp`,

и возвращает только один символ из связанного с данным дескриптором файла; если функция достигает конца файла, то она возвращает `False`. По существу, она работает так же, как и `fread()` — вызов:

```
$one_char = fgetc($fp)
```

эквивалентен вызову:

```
$one_char = fread($fp,1)
```

Модифицируем первоначальный счетчик с использованием в нем функции `fgetc()`. Для этого можно использовать такой код:

```
<?php
//hit_counter03.php
$counter_file = "./count.dat";
if(!($fp = fopen($counter_file, "r"))) die ("Невозможно открыть файл $counter_file.");

do {
    $one_char = fgetc($fp);
    $counter .= $one_char;
} while($one_char);
$counter = (int) $counter;
fclose($fp);
```

Чтобы считать все содержимое файла данных, используется цикл `while`, потому что функция `fgetc()` считывает из файла только один символ за раз. Кроме того, необходимо знать, когда прекращать чтение, поэтому последний считанный символ записывается в холостую переменную, и когда она будет равна `False`, цикл чтения можно прекратить. Однако в таком коде есть один дефект. Как только из файла будет считано значение `"0"` или `" "`, условие цикла не выполнится и цикл чтения прекратится. Если планируется, что значение счетчика посещений будет превышать 9, то данная особенность может оказаться серьезной проблемой.

Существует другой способ определить момент завершения цикла чтения — использование функции `feof()`.

Функция `feof()` служит одной простой цели: она возвращает `True`, когда достигает конца заданного файла (или если возникает ошибка), и `False` — в противном случае. Функция принимает только один аргумент — дескриптор файла:

```
feof($fp)
```

Поэтому в качестве условия цикла можно использовать логическое отрицание и проверить, таким образом, не достиг ли указатель конца данного файла:

```
<?php
//hit_counter04.php
$counter_file = "./count.dat";
if(!($fp = fopen($counter_file, "r"))) die ("Невозможно открыть файл $counter_file.");
while(!feof($fp)) $counter .= fgetc($fp);
$counter = (int) $counter;
fclose($fp);

$counter++;

echo "Вы - посетитель № $counter.";

$fp = fopen($counter_file, "w");
fwrite($fp, $counter);
fclose($fp);
?>
```

Функция `feof()` сообщает оператору `while`, когда необходимо завершить цикл (когда функция `fgetc()` в процессе посимвольного считывания достигает конца

заданного файла). Затем значение переменной `$counter` с помощью оператора `(int)` приводится к целому типу.

Чтение больших файлов с помощью функции `fgetc()` может отнимать очень много времени, потому что данная функция считывает только один символ за раз. Считывать множество символов позволяет функция `fgets()`. Она принимает два аргумента, `$fp` и `$length`, и возвращает прочитанную из файла, заданного дескриптором `$fp`, строку, длина которой не превышает значения `$length` (`$length - 1`). Функция прекращает чтение по одной из трех перечисленных ниже причин:

- ☐ прочитано заданное количество байтов;
- ☐ достигнут конец строки;
- ☐ достигнут конец файла.

Различие между `fgets()` и `fread()` заключается в том, что `fgets()` прекращает чтение после того, как был достигнут конец строки или считано `length - 1` байтов, тогда как функция `fread()` игнорирует конец строки и считывает `$length` байтов. Функцию `fgets()` можно применить в счетчике, используя следующий код:

```
<?php
//hit_counter05.php
$counter_file = "./count.dat";
if(!($fp = fopen($counter_file, "r"))) die ("Невозможно открыть файл $counter_file.");
$counter = (int) fgets($fp, 20);
fclose($fp);

...
```

Сценарий должен считывать из файла данных только одну строку, поэтому вызов `fgets()` возвращает именно то, что нужно, т.е. последнее значение счетчика.

Тот, кто сталкивался с импортированием и экспортированием данных, знаком с форматом CSV (comma-separated-value — значения, разделенные запятыми). (CSV даже имеет собственное расширение файлов: `.csv`.) В CSV-файлах значения данных разделены запятыми, а строковые значения часто заключаются в двойные кавычки между запятыми. Функция `fgetcsv()` считывает данные из файлов, предполагая, что эти данные отформатированы в правильном CSV-формате, и помещает найденные в строке данные в массив. Естественно, после этого можно легко использовать полученный массив с данными.

Функции `fgetcsv()` необходимо передать корректный дескриптор файла — численное значение, которое должно превышать длину каждой строки (включая символы конца строки), а также необязательные разделители данных (по умолчанию запятая) и ограничители данных (по умолчанию двойные кавычки). Следующий фрагмент кода показывает, как можно получить строку значений данных из CSV-файла:

```
$file_handle = fopen("my_csv_text_file.csv", "r");
$my_data_values_array = fgetcsv($file_handle, 1000, ",", "");
```

В результате работы этого кода формируется массив `$my_data_values_array`; если бы первым значением в строке текста было чье-нибудь имя (например, Боб), то в элемент `$my_data_values_array[0]` было бы записано значение "Боб". Следует отметить, что пустая строка в текстовом файле интерпретируется не как ошибка, а как одно NULL-поле.

Функция `fputs()` представляет собой синоним функции `fwrite()`; две эти функции идентичны.

Чтение файлов целиком

Следующие функции предоставляют доступ ко всему содержимому файла за один раз:

- ☐ `file()`
- ☐ `fpassthru()`
- ☐ `readfile()`

Функция `file()` принимает только один аргумент: строку, содержащую имя файла. Например, вызов

```
file("/home/chris/myfile.txt")
```

возвращает все содержимое файла `myfile.txt` (находящегося в каталоге `/home/chris/`) в виде массива, каждый элемент которого представляет собой одну строку файла, включая символ конца строки (CRLF на Windows-платформе). Функция не требует указания дескриптора файла, а позволяет задавать имя файла явно; она автоматически открывает файл, считывает из него данные и по завершении чтения закрывает файл.

Рассмотрим `file()`-версию счетчика посещений:

```
<?php
//hit_counter06.php
$counter_file = "./count.dat";
$lines = file($counter_file);
$counter = (int) $lines[0];

$counter++;

echo "Вы - посетитель № $counter.";

if(!$fp = fopen($counter_file, "w")) die ("Невозможно открыть файл $counter_file.");
fwrite($fp, $counter);
fclose($fp);
?>
```

Все содержимое файла данных считывается в массив `$lines`, после чего извлекается только первый элемент данного массива (первая строка файла). То же самое можно было бы сделать с помощью комбинации функций `feof()` и `fgets()`, например, так:

```
$fp = fopen ("./count.dat", "r");
while (!feof($fp)) $lines[] = fgets($fp, 1024);
fclose ($fp);
```

Однако делать это нет необходимости, потому что функция `file()` делает все это автоматически. Как и `fopen()`, функция `file()` способна открывать файлы на удаленном узле:

```
$file_lines = file("http://www.whatyoumaycallit.com/index.html");
foreach($file_lines as $line) echo $line;
```

Хотя описываемая функция может оказаться очень полезной для чтения всего содержимого файлов, ее следует использовать осторожно — если попытаться с ее помощью прочитать очень большой файл, то она может занять всю выделенную для PHP память.

Функция `fpassthru()` оказывается полезной, если нужно прочесть файл и распечатать его целиком в Web-браузер. Она принимает единственный аргумент, дескриптор файла, который использует для считывания из файла оставшихся данных (т.е. с текущей позиции до конца файла). Затем она записывает результаты в стандартный вывод. Ниже приведена соответствующая версия счетчика посещений.

```
<?php
//hit_counter07.php
$counter_file = "./count.dat";
if(!($fp = fopen($counter_file, "r"))) die ("Невозможно открыть файл $counter_file.");
$counter = (int) fread($fp, 20);
fclose($fp);

$counter++;

if(!($fp = fopen($counter_file, "w"))) die ("Невозможно открыть файл $counter_file.");
fwrite($fp, $counter);
fclose($fp);

if(!($fp = fopen($counter_file, "r"))) die ("Невозможно открыть файл $counter_file.");
echo "Вы - посетитель № ";
fpassthru($fp);
?>
```

В поток стандартного вывода записываются данные, начиная с текущей позиции. Например, если перед вызовом `fpassthru()` прочитать из файла несколько строк, то будет распечатано только содержимое последующих строк. Файл закрывается после того, как функция заканчивает чтение, поэтому нет необходимости вызывать `fclose()` — фактически, если это сделать, то появится предупреждение о некорректном указателе файла.

Функция `readfile()` позволяет распечатывать содержимое файла, даже не вызывая `fopen()`. В качестве своего единственного аргумента она принимает имя файла, считывает этот файл целиком, а затем распечатывает его в стандартный вывод, возвращая при этом количество считанных байтов (или `False` в случае ошибки). Рассмотрим ее применение в счетчике посещений:

```
<?php
//hit_counter08.php
$counter_file = "./count.dat";
if(!($fp = fopen($counter_file, "r"))) die ("Невозможно открыть файл $counter_file.");
$counter = (int) fread($fp, 20);
fclose($fp);

$counter++;

if(!($fp = fopen($counter_file, "w"))) die ("Невозможно открыть файл $counter_file.");
fwrite($fp, $counter);
fclose($fp);

echo "Вы - посетитель № ";
readfile($counter_file);
?>
```

Как и в предыдущем сценарии, все операции чтения и записи данных в файл выполняются до вывода на страницу последнего значения счетчика.

Произвольный доступ к данным файла

Вероятно, было бы более эффективно открывать файл для чтения и записи с помощью одного вызова `fopen()`, однако рассмотренные выше функции лишь позволяют манипулировать данными последовательно, т.е. в том же порядке, в котором эти данные организованы (или будут организованы в файле). Это сильно ограничивает возможности их применения; как только указатель позиции в файле переместился к определенной точке, приходится закрывать файл и открывать его снова, прежде

чем можно будет получить доступ к данным в этой точке. Если доступ к данным не был заблаговременно и идеально организован (что чаще всего и бывает; невозможно предугадать, что потребуется в будущем), то, в конце концов, файл будет открываться столько же раз (если не больше), как представленных ранее в примерах.

Следовательно, нужно найти способ перемещать указатель позиции в файле без необходимости закрывать и заново открывать данный файл. Для этого существует две функции:

- ❑ `fseek()`
- ❑ `ftell()`

PHP предоставляет множество функций, которые предназначены для того, чтобы прочитать данные из определенной позиции или записать их в определенную позицию файла. Если передать функции `fseek()` в качестве аргументов дескриптор файла (`$fp`) и целое смещение (в следующем примере 5), то данная функция переместит указатель позиции в файле, связанном с дескриптором `$fp`, в позицию, которая определяется смещением. По умолчанию смещение отсчитывается в байтах от начала файла. Рассмотрим пример:

```
fseek($fp, 5);
$one_char = fgetc($fp);
```

Данный код помещает указатель позиции в файле, связанном с дескриптором `$fp`, сразу после пятого байта в этом файле. Следовательно, вызов `fgetc()` возвращает содержимое шестого байта. Для расчета относительного смещения с помощью одного из следующих значений можно задать третий необязательный аргумент (который называется в документации *whence* (точка отсчета)):

- ❑ `SEEK_SET`: начало файла плюс смещение;
- ❑ `SEEK_CUR` (используется по умолчанию): текущая позиция плюс смещение;
- ❑ `SEEK_END`: конец файла плюс смещение.

Функция `fseek()` необычна, потому что является целочисленной PHP-функцией, которая возвращает 0, а не 1 в случае успеха (а в случае неудачи возвращает -1). Ее нельзя использовать с файлами на удаленных узлах через HTTP или FTP URL.

Функция `ftell()` принимает дескриптор файла и возвращает текущее смещение (в байтах) указателя позиции в данном файле. Например:

```
$fpi_offset = ftell($fp);
rewind()
```

Функция работает подобно обратной перемотке ленты в магнитофоне — она принимает дескриптор файла и переустанавливает указатель позиции в начало этого файла. Вызов

```
rewind($fp);
```

функционально аналогичен вызову

```
fseek($fp, 0);
```

Как было показано выше, функция `fpassthru()` выводит данные файла, начиная с текущей позиции. Если данные из файла уже считывались, но желательно вывести все содержимое файла, то сначала необходимо вызвать функцию `rewind()`.

С помощью функции `rewind()` можно модернизировать сценарий счетчика так, чтобы он открывал файл только однажды и для чтения и для записи:

```
<?php
//hit_counter09.php
$counter_file = "./count.dat";
if(!($fp = fopen($counter_file, "r+"))) die ("Невозможно открыть файл
$counter_file.");
$counter = (int) fread($fp, 20);
$counter++;

echo "Вы - посетитель № $counter.";
rewind($fp);
fwrite($fp, $counter);
fclose($fp);
?>
```

Очевидно, что в данном случае файл открывается только один раз в режиме чтения-записи. После считывания из файла последнего числа посещений и отображения этого числа вызывается функция `rewind()`, которая возвращает указатель позиции в начало файла.

Практика Навигация по файлу

В следующем примере используются описанные функции `fseek()`, `ftell()` и `rewind()`:

```
<?php
//nav_file.php
$name_field_len = 15;
$country_code_field_len = 2;
$country_field_len = 20;
$email_field_len = 30;

if(!($fp = fopen("./address.dat", "r"))){
    die ("Невозможно открыть файл с адресными данными.");
}

do{
    $address = '';
    $field = fread($fp, $name_field_len);
    $address .= $field;

    $field = fread($fp, $country_code_field_len);
    $address .= $field;
    $field = fread($fp, $country_field_len);
    $address .= $field;
    $field = fread($fp, $email_field_len);
    $address .= $field;
    echo "$address<BR>";
}while($field);

rewind($fp);

echo "<BR>";

fseek($fp, $name_field_len);

do{
    $country_code = fread($fp, $country_code_field_len);
    fseek($fp, ftell($fp) + $country_field_len +
        $email_field_len +
        $name_field_len + 1);
    //Примечание: на Win32-платформах '+1' необходимо заменить на '+2'
```

```

    echo "$country_code<BR>";
}while($country_code);

fclose($fp);
?>

```

Предполагается, что в текущем каталоге существует файл адресной книги, который называется `address.dat` и выглядит следующим образом (необходимо очень внимательно ввести данный текст и даже соблюдать количество пробелов в каждом поле):

```

Wankyu Choi      KRRepublic of Korea  wankyu@whatyoumaycallit.com
James Hetfield   USUnited States      james@headbangers.com
Nomura Sensei    JPJapan              nomura@nosuchsite.com

```

Результат тестового запуска сценария:

```

Wankyu Choi KRRepublic of Korea wankyu@whatyoumaycallit.com
James Hetfield USUnited States james@headbangers.com
Nomura Sensei JPJapan nomura@nosuchsite.com

```

```

KR
US
JP

```

Записи в данном файле отделяются друг от друга символами новой строки. (Как уже говорилось, символом новой строки в Windows-платформах является последовательность CRLF, в Macintosh — CR, а в Linux — LF.) Каждое поле имеет фиксированную длину: 15 символов для имени, 2 символа для кода страны и т.д.

Как это работает

Сценарий начинает свою работу с открытия в текущем каталоге файла `address.dat` в режиме для чтения. Сначала отображаются все записи в том виде, как они есть в файле. Когда функция `fread()` достигает конца файла, переменная `$field` получает значение `False` и первый цикл прекращается:

```

if(!( $fp = fopen("./address.dat", "r"))){
    die ("Невозможно открыть файл с адресными данными.");
}

do{
    $address = '';
    $field = fread($fp, $name_field_len);
    $address .= $field;

    $field = fread($fp, $country_code_field_len);
    $address .= $field;
    $field = fread($fp, $country_field_len);
    $address .= $field;
    $field = fread($fp, $email_field_len);
    $address .= $field;
    echo "$address<BR>";
}while($field);

```

После этого указатель позиции возвращается в начало файла, а затем в конец первого поля с именем. Теперь сценарий готов считывать и отображать значения кодов страны:

```

rewind($fp);
fseek($fp, $name_field_len);

```

Начинается цикл `do...while`. В каждой его итерации сначала с помощью функции `fread()` переменной `$country_code` присваивается код страны. Затем указатель

позиции перемещается в начало следующего поля с кодом страны, и, наконец, полученный код выводится на экран.

```
do{
    $country_code = fread($fp, $country_code_field_len);
```

Точная позиция следующего поля с кодом страны определяется таким образом:

- ❑ с помощью вызова `ftell($fp)` определяется текущая позиция указателя;
- ❑ к данному числу прибавляется общая длина всех оставшихся полей и завершающий символ новой строки.

Используя полученный результат как второй аргумент для функции `fseek()`, можно установить указатель позиции в соответствующую точку файла:

```
fseek($fp, ftell($fp) + $country_field_len +
    $email_field_len +
    $name_field_len + 1);
```

Для Windows-платформ длина комбинации CR LF равна 2, а не 1 (как в Linux или Macintosh). Записанное значение выводится на экран и цикл, который длится, пока переменная `$country_code` содержит значение отличное от `False`, закрывается:

```
echo "$country_code<BR>";
}while($country_code);
```

Наконец, закрывается файл данных:

```
fclose($fp);
```

Получение информации о файлах

В данной главе уже было показано, какие проблемы могут возникнуть, если сценарий попытается открыть несуществующий файл счетчика. Кроме этого, вы уже ознакомились с некоторыми базовыми процедурами проверки ошибок при открытии файла. Такой подход очень прост — если что-либо происходит неправильно, то пользователю необязательно знать, в чем заключается проблема.

PHP предоставляет несколько базовых функций, которые позволяют получить доступ к весьма полезной информации. Например, вместо того, чтобы выдавать стандартное сообщение об ошибке при неудачном открытии файла, можно использовать функцию `file_exists()`, позволяющую проверить существование файла. Если заданного файла не существует, то можно сделать вывод, что данный пользователь является первым посетителем этой страницы, и создать необходимый файл данных. Например:

```
file_exists("/home/chris/count.dat")
```

Данная функция возвращает `True`, если файл `count.dat` существует и находится в каталоге `/home/chris/`, и `False` — в противном случае. Проверка ошибок теперь может иметь следующую форму:

```
<?php
//hit_counter_10.php
$counter_file = "./count.dat";
if(file_exists($counter_file)) {
    if(!($fp = fopen($counter_file, "r+")))
        die("Невозможно открыть файл $counter_file");
    $counter = (int) fread($fp, filesize($counter_file));
    $counter++;
    rewind($fp);
}
```



```

else {
    if(!($fp = fopen($counter_file, "w")))
        die("Невозможно открыть файл $counter_file");
    $counter = 1;
}

echo "Вы - посетитель № $counter.";

fwrite($fp, $counter);
fclose($fp);
?>

```

Это только одна из множества функций, которые возвращают полезную информацию о заданном файле. Подобным образом можно использовать функцию `filesize()`, определяющую точное количество байтов, которые необходимо считать из файла данных. Как и предыдущая функция, `filesize()` вместо дескриптора файла принимает непосредственно его имя:

```
filesize("/home/chris/count.dat")
```

В результате такого вызова возвращается размер заданного файла в байтах или `False` в случае ошибки. Функцию `filesize()` можно использовать для того, чтобы определить, сколько байтов необходимо считать из файла счетчика:

```

<?php
//hit_counter11.php
$counter_file = "./count.dat";
if(file_exists($counter_file)) {
    if(!($fp = fopen($counter_file, "r+")))
        die("Невозможно открыть файл $counter_file");
    $counter = (int) fread($fp, filesize($counter_file));
    $counter++;
    rewind($fp);
}
...

```

Временные свойства файлов

Кроме содержимого, существуют также другие свойства файлов, которые могут сообщать полезную информацию об этих файлах. Эти свойства главным образом зависят от операционной системы, на которой создаются и модифицируются файлы. На Unix-платформах, например, таких как Linux, данные свойства включают в себя дату создания, дату изменения, дату последнего доступа и права пользователей. Добавив немного кода, можно заставить сценарий счетчика отображать время последнего доступа к странице. Функция `fileatime()` возвращает время последнего доступа к файлу в формате временных меток Unix, а также дату изменения файла в Windows-формате.

Временная метка Unix представляет собой длинное целое число, которое можно интерпретировать как количество секунд между началом так называемой Unix-эпохи (1 января 1970 года) и указанной датой и временем.

PHP предоставляет две другие связанные с временными свойствами файлов функции.

- ❑ `filectime()` возвращает время последнего изменения файла в формате временной метки Unix. Изменение файла фиксируется при создании файла, записи данных в файл или изменении прав доступа к файлу.

- ❑ `filemtime()` возвращает время последней модификации файла в формате временной метки Unix. Файл считается модифицированным, если он был создан или его содержимое изменилось.

При работе с временными метками очень полезной является функция `getdate()`. Она возвращает ассоциативный массив, содержащий информацию о представленной в виде временной метки дате. Возвращаемый массив содержит такие значения, как год, месяц, день месяца и др. Можно записать возвращаемое функцией `getdate()` значение в переменную (например, `$my_date`), а затем получить месяц, обратившись к элементу `$my_date['month']`.

Практика Отображение времени последнего доступа к файлу данных счетчика

В данном примере для получения значений даты для файла, имя которого записано в переменную `$counter_file`, используются функции `filemtime()` и `getdate()`.

```
<?php
//last_counter_access.php
$counter_file = "./count.dat";
if(file_exists($counter_file)){
    $date_str = getdate(filemtime($counter_file));
    $year = $date_str["year"];
    $mon = $date_str["mon"];
    $mday = $date_str["mday"];
    $hours = $date_str["hours"];
    $minutes = $date_str["minutes"];
    $seconds = $date_str["seconds"];

    $date_str = "$hours:$minutes:$seconds $mday/$mon/$year";

    if(!($fp = fopen($counter_file, "r+"))){
        die("Невозможно открыть файл $counter_file");
    }

    $counter = (int) fread($fp, filesize($counter_file));
    $counter++;

    echo "Вы - посетитель № $counter.";
    echo "Последнее посещение страницы состоялось $date_str";
    rewind($fp);
}else{
    if(!($fp = fopen($counter_file, "w"))){
        die("Невозможно открыть файл $counter_file");
    }
    $counter = 1;
    echo "Вы - посетитель № $counter.";
}

fwrite($fp, $counter);
fclose($fp);
?>
```

На рис. 7.4 показано, как должна выглядеть эта страница (следует отметить, что на Windows-системах может отображаться время создания файла, а не время последнего доступа к нему).

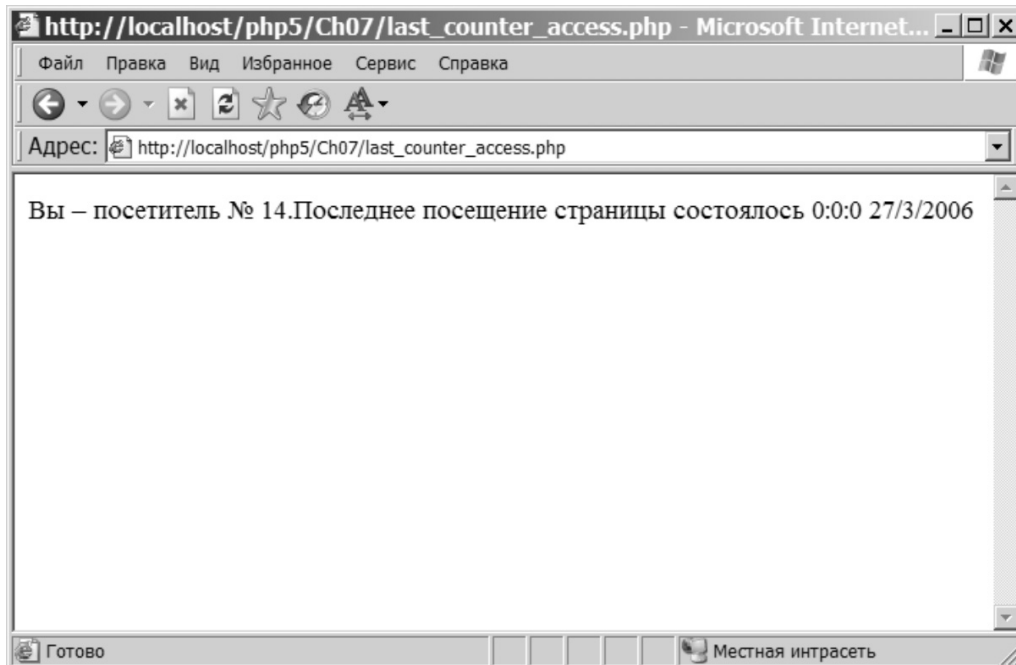


Рис. 7.4.

Как это работает

Если указанный файл, `count.dat`, существует, то сценарий получает в виде временной метки Unix время последнего доступа к данному файлу. Полученное значение с помощью функции `getdate()` преобразовывается в ассоциативный массив, значения данного массива извлекаются и записываются в строковые переменные:

```
$counter_file = "./count.dat";
if(file_exists($counter_file)){
    $date_str = getdate(filemtime($counter_file));
    $year = $date_str["year"];
    $mon = $date_str["mon"];
    $mday = $date_str["mday"];
    $hours = $date_str["hours"];
    $minutes = $date_str["minutes"];
    $seconds = $date_str["seconds"];

    $date_str = "$hours:$minutes:$seconds $mday/$mon/$year";
}
```

При создании, записи или чтении файла операционная система Unix фиксирует доступ к файлу (в Windows изменение файла может фиксироваться при других обстоятельствах и, как уже отмечалось, вместо времени последнего доступа будет показываться время создания файла). По этой причине перед чтением файла вызывается функция `filemtime()`. Как и ранее, при условии существования файла `count.dat` сценарий открывает данный файл, считывает все его содержимое в переменную `$counter`, увеличивает ее значение на единицу и распечатывает полученное значение вместе с отформатированной информацией о времени доступа (или изменения). Затем необходимо вызвать функцию `rewind()` для возвращения указателя позиции

в начало файла (чтобы после этого можно было снова записывать в этот файл данные) и закрыть оператор `if`:

```
if(!($fp = fopen($counter_file, "r+"))){
    die("Невозможно открыть файл $counter_file");
}

$counter = (int) fread($fp, filesize($counter_file));
$counter++;

echo "Вы - посетитель № $counter.";
echo "Последнее посещение страницы состоялось $date_str";
rewind($fp);
}
```

Далее следует оператор `else`; если файл `count.dat` отсутствует, то его необходимо создать с помощью функции `fopen()`, указав для нее режим только для записи. После этого счетчик инициализируется значением 1, которое выводится в сообщении:

```
else{
    if(!($fp = fopen($counter_file, "w"))){
        die("Невозможно открыть файл $counter_file");
    }
    $counter = 1;
    echo "Вы - посетитель № $counter.";
}
```

Наконец, в файл `count.dat` записывается новое значение счетчика, после чего файл закрывается с помощью вызова `fclose()`:

```
fwrite($fp, $counter);
fclose($fp);
```

Принадлежность и права доступа к файлам

Каким же образом можно получить информацию о принадлежности файла и правах доступа к нему? В Unix-подобных системах, таких как Linux, все файлы связываются с определенными пользователями и группами пользователей и им (файлам) назначаются флаги, определяющие пользователей, которые имеют полномочия на чтение, запись или выполнение данных файлов. Некоторые файловые системы Windows (но не все) работают аналогично. Например, Windows NTFS (NT File System) позволяет назначать принадлежность и права доступа, тогда как в Windows 98 такой возможности нет.

В Unix определяются группы пользователей с тем, чтобы можно было легко расширять полномочия на определенные группы пользователей без необходимости назначать эти полномочия каждому пользователю системы по отдельности. Например, несколько пользователей, работающих над одним проектом, могут использовать файлы совместно друг с другом, но только в пределах определенной группы.

Каждое из этих прав (чтение, запись и выполнение) может быть предоставлено (или наоборот отменено):

- ☐ владельцу файла: по умолчанию владельцем файла является пользователь системы, создавший этот файл;
- ☐ группе пользователей: по умолчанию это группа, к которой принадлежит владелец файла;
- ☐ всем пользователям: все пользователи, имеющие учетные записи в системе.

Пользователи и группы в Unix идентифицируются по идентификационным номерам. Если в РНР-сценарии необходимо получить информацию о пользователе по его идентификатору, то это можно сделать с помощью функции `posix_getpwuid()`, которая возвращает ассоциативный массив со следующими данными:

Имя	Описание
<code>name</code>	Имя учетной записи shell данного пользователя
<code>passwd</code>	Зашифрованный пароль пользователя
<code>uid</code>	ID-номер пользователя
<code>gid</code>	Идентификатор группы данного пользователя
<code>gecos</code>	Перечень (поля разделены запятыми), содержащий полное имя пользователя, рабочий телефон, номер офиса и домашний телефон. В большинстве систем доступно только полное имя пользователя
<code>dir</code>	Абсолютный путь к начальному каталогу данного пользователя
<code>shell</code>	Абсолютный путь к пользовательской оболочке (shell) по умолчанию

Другая РНР-функция, `posix_getgrgid()`, возвращает ассоциативный массив информации о группе по идентификатору данной группы. В массиве содержатся следующие элементы:

Имя	Описание
<code>name</code>	Имя группы
<code>gid</code>	Идентификатор (ID) группы
<code>members</code>	Количество пользователей, принадлежащих данной группе

Для получения из РНР-сценариев полезной информации о файлах можно использовать следующие три функции (каждая из них принимает единственный аргумент — имя файла):

- ❑ `fileowner()`: возвращает пользовательский идентификатор владельца заданного файла;
- ❑ `filegroup()`: возвращает идентификатор группы владельца заданного файла;
- ❑ `filetype()`: возвращает тип заданного файла (`fifo`, `char`, `dir`, `block`, `link`, `file` или `unknown`).

Например, чтобы проверить, является ли данный файл файлом или каталогом, можно использовать функцию `filetype()` (убедитесь, что файл `counter.php` существует, а если нет, то создайте его):

```
<?php
//file_type.php
$filename = "./counter.php";
$filegroup = filegroup($filename);
$fileowner = fileowner($filename);
$filetype = filetype($filename);

if ($filetype == 'dir') {
    echo "$filename является каталогом.";
} else if ($filetype == 'file') {
    echo "$filename является файлом.<br>";
} else {
```

```

    echo "$filename не является ни файлом ни каталогом.<br>";
}

echo "$filename принадлежит пользователю № $fileowner и группе № $filegroup.<br>";
echo "<br>Информация о пользователе $fileowner<br>";
$user_info_array = posix_getpwuid($fileowner);

foreach ($user_info_array as $key => $val) {
    echo "$key => $val<br>";
}

echo "<br>Информация о группе $filegroup<br>";
$group_info_array = posix_getgrgid($filegroup);

foreach ($group_info_array as $key => $val) {
    echo "$key => $val<br>";
}
?>

```

На рис. 7.5 показан пример работы сценария на Windows-машине. Обратите внимание на ошибку, возникающую при попытке использовать на Windows-машине функцию, которая работает только на Linux-машинах. Вообще данный сценарий лучше запускать на Linux-машине, а для получения подобной информации о файлах на Windows-машинах можно использовать следующий сценарий.

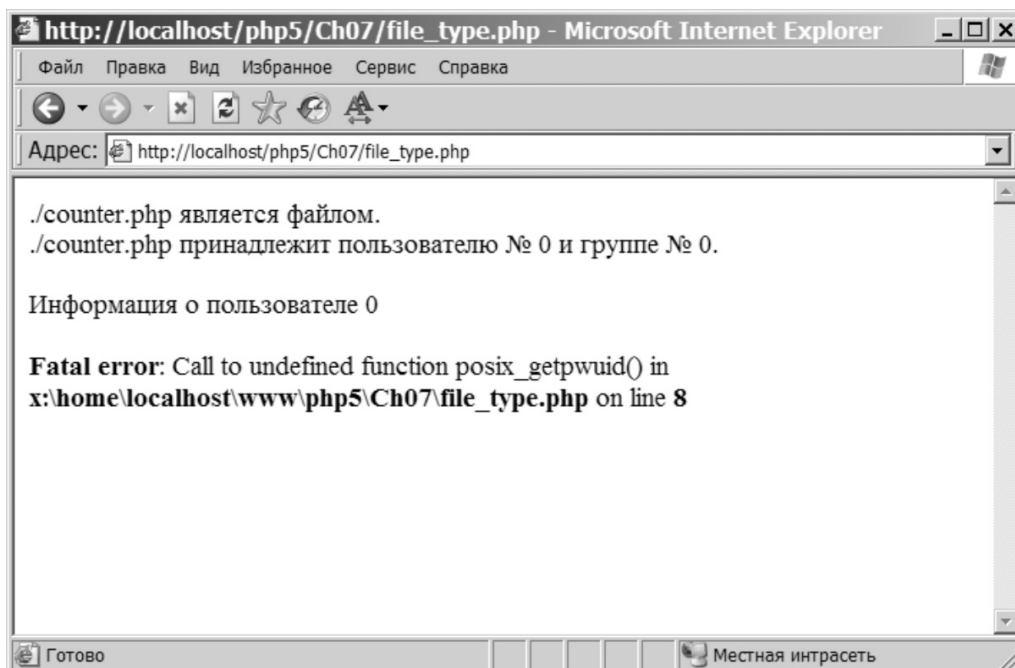


Рис. 7.5.

Функции `is_dir()` и `is_file()`

Часто сценарий должен определить, является ли файл каталогом (как уже отмечалось, каталоги — тоже файлы, но файлы особого типа). Предположим, например, что требуется создать сценарий, который проходит по иерархии каталогов. Для этого необ-

ходимо определить, является ли какой-либо файл каталогом, с тем, чтобы можно было войти в него и продолжить перемещение по иерархии. Точно так же, если требуется просматривать только файлы в каталоге, необходимо определить, какой файл действительно является файлом. Для решения обеих задач в PHP предусмотрены две функции:

- ❑ `is_dir()` специально предназначена для работы с каталогами; она возвращает `True`, если заданное имя файла ссылается на какой-либо каталог;
- ❑ `is_file()` возвращает `True`, если заданное имя файла ссылается на обычный файл.

Предыдущий сценарий очень легко переписать и заставить его выводить те же результаты, но с помощью данных функций:

```
<?php
$filename = "./counter.php";
if(is_dir($filename)) {
    echo "$filename - каталог.";
} else if (is_file($filename)) {
    echo "$filename - файл.";
} else {
    echo "$filename - не файл и не каталог.";
}
?>
```

Как уже отмечалось, функция `fileatime()` на Windows-машине возвращает дату последнего изменения файла; а так как Windows не поддерживает принадлежность файлов, обе функции `filegroup()` и `fileowner()` возвращают нуль.

Практика Получение информации о Windows-файлах

Следующий код сценария отображает некоторые свойства заданного файла.

```
<?php
function date_str($timestamp)
{
    $date_str = getdate($timestamp);
    $year = $date_str["year"];
    $mon = $date_str["mon"];
    $mday = $date_str["mday"];
    $hours = $date_str["hours"];
    $minutes = $date_str["minutes"];
    $seconds = $date_str["seconds"];
    return "$hours:$minutes:$seconds $mday/$mon/$year";
}
function file_info($file)
{
    $file_info_array["filesize"] = number_format(filesize($file)) . " байт.";
    $file_info_array["filectime"] = date_str(filectime($file));
    $file_info_array["filemtime"] = date_str(filemtime($file));
    if(!isset($_ENV["WINDIR"])){
        $file_info_array["fileatime"] = date_str(fileatime($file));
        $file_info_array["filegroup"] = filegroup($file);
        $file_info_array["fileowner"] = fileowner($file);
    }
    $file_info_array["filetype"] = filetype($file);
    return $file_info_array;
}
```

```
$filename = "./count.dat";  
$file_info_array = file_info($filename);  
  
echo "<center>Свойства файла $filename</center>";  
foreach($file_info_array as $key=>$val){  
    echo ucfirst($key) . "=>". $val . "<br>";  
}  
?>
```

На рис. 7.6 показана работа данного сценария на Windows-машине.

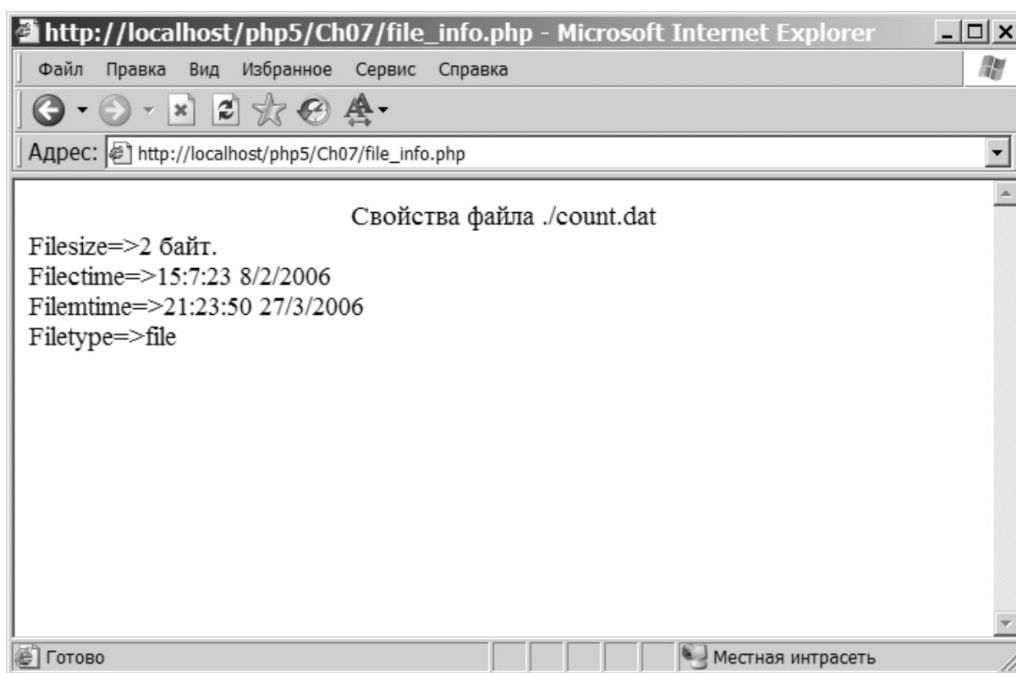


Рис. 7.6.

Как это работает

Несколько строк кода из последней версии счетчика можно поместить в функцию (назовем ее `date_str()`), которая возвращает отформатированную строку с датой:

```
function date_str($timestamp) {  
    $date_str = getdate($timestamp);  
    $year = $date_str["year"];  
    $mon = $date_str["mon"];  
    $mday = $date_str["mday"];  
    $hours = $date_str["hours"];  
    $minutes = $date_str["minutes"];  
    $seconds = $date_str["seconds"];  
  
    return "$hours:$minutes:$seconds $mday/$mon/$year";  
}
```


В следующей функции, `file_info()`, вводится несколько новых функций, которые возвращают полезную информацию о заданных файлах:

```
function file_info($file)
{
    $file_info_array["filesize"] = number_format(filesize($file)) . " байт.";
    $file_info_array["filectime"] = date_str(filectime($file));
    $file_info_array["filemtime"] = date_str(filemtime($file));
    if(!isset($_ENV[WINDIR])){
        $file_info_array["fileatime"] = date_str(fileatime($file));
        $file_info_array["filegroup"] = filegroup($file);
        $file_info_array["fileowner"] = fileowner($file);
    }
    $file_info_array["filetype"] = filetype($file);
    return $file_info_array;
}
```

Переменная среды `$_ENV[WINDIR]` устанавливается только на Windows-платформах. К ней можно получить доступ через `$_ENV` и использовать как флаг для указания того, что сценарий выполняется в Windows. Если эта переменная установлена, то в результирующий массив информации не попадают поля, которые в Windows не поддерживаются. Обратите внимание, чтобы сделать эту переменную видимой внутри функции при условии использования ключевого слова `global`, также можно использовать старую ссылку на нее, `$HTTP_ENV_VARS`. Хотя этот способ работоспособен, он вытесняется использованием суперглобального массива `$_ENV`. Преимущество использования суперглобального массива `$_ENV` заключается в том, что эту переменную не требуется объявлять как глобальную, чтобы получить к ней доступ внутри какой-либо пользовательской функции.

Наконец, задается файл, вызывается функция `file_info()` и распечатывается содержимое каждого элемента возвращаемого ею массива:

```
$filename = "./count.dat";
$file_info_array = file_info($filename);

echo "<center>Свойства файла $filename</center>";
foreach($file_info_array as $key=>$val){
    echo ucfirst($key) . "=>". $val . "<br>";
}
```

Пользовательские функции для работы с файлами

Теперь, зная, как определить владельца файла, можно выполнять некоторые простейшие действия, такие как копирование, переименование и удаление файлов. Чтобы выполнять эти операции, необходимо выяснить точный путь к файлу. Для этого PHP предоставляет несколько очень полезных функций, например, `basename()`.

Разделение имени файла и пути

Часто полезной оказывается возможность отделять имя файла от пути к его каталогу; именно это и делает функция `basename()`, которая принимает полный путь к файлу и возвращает только его имя. Например, в следующем вызове переменной `$filename` присваивается значение `"index.html"`:

```
$filename = basename("home/james/docs/index.html");
```

Можно указать путь к какому-либо каталогу. В таком случае функция возвращает самый правый каталог в пути. В следующем примере переменной `$dirname` присваивается значение "docs":

```
$dirname = basename("home/james/docs");
```

В сущности, функция `basename()` представляет собой средство для получения последней подстроки после крайнего правого символа кривой черты.

Копирование, переименование и удаление файлов

PHP также позволяет копировать, переименовывать и удалять файлы. Для выполнения этих операций предназначены функции `copy()`, `rename()` и `unlink()`.

Функция `copy()` принимает два строковых аргумента, ссылающихся на исходный и конечный файл соответственно. Следующий вызов функции копирует исходный файл `copyme.txt` в конечный файл `copied.txt`:

```
if(!copy("./copyme.txt", "copied.txt")) die("Невозможно скопировать файл  
copyme.txt в copied.txt!");
```

Функция `rename()` используется для переименования файлов:

```
if(!rename("./address.dat", "address.backup"))  
die("Невозможно переименовать файл address.dat в address.backup!");
```

С помощью данных функций можно усовершенствовать счетчик посещений так, чтобы он периодически (например, один раз в час, в день, в неделю и т.д.) сохранял резервную копию файла данных:

```
<?php  
//hit_counter12.php  
$counter_file = "./count.dat";  
$counterbackup_file = "./count.backup";  
$backup_interval = 24*60*60;  
  
if(file_exists($counter_file)) {  
    $date_str = getdate(filemtime($counter_file));  
    $year = $date_str["year"];  
    $mon = $date_str["mon"];  
    $mday = $date_str["mday"];  
    $hours = $date_str["hours"];  
  
    $minutes = $date_str["minutes"];  
    $seconds = $date_str["seconds"];  
    $date_str = "$hours:$minutes:$seconds $mday/$mon/$year";  
  
    if((time() - filemtime($counterbackup_file)) >= $backup_interval) {  
        @copy($counter_file, $counterbackup_file);  
    }  
  
    if(!($fp = fopen($counter_file, "r+")))  
        die("Невозможно открыть файл $counter_file");  
    $counter = (int) fread($fp, filesize($counter_file));  
    $counter++;  
    echo "Вы - посетитель № $counter.  
        Последнее посещение страницы состоялось $date_str";  
    rewind($fp);  
}  
else {  
    if(!($fp = fopen($counter_file, "w")))  
        die("Невозможно открыть файл $counter_file");  
    $counter = 1;  
    echo "Вы - посетитель № $counter.";  
}
```

```
fwrite($fp, $counter);
fclose($fp);
```

```
?>
```

На рис. 7.7 показан результат (в Windows-системах корректное время доступа не отображается).

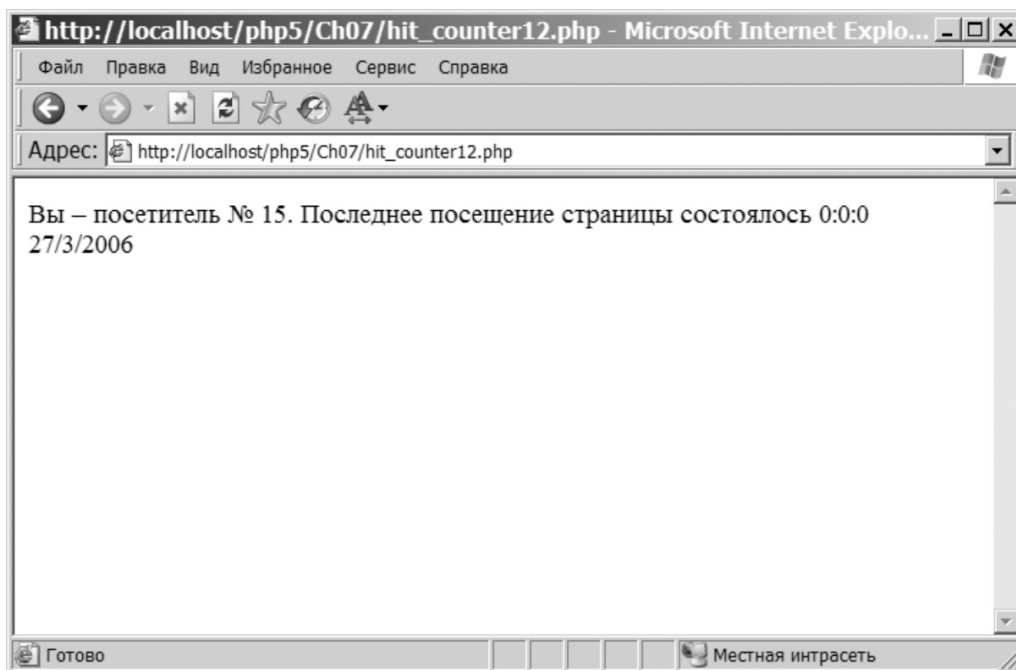


Рис. 7.7.

Переменной `$backup_interval` присваивается значение, равное количеству секунд в сутках ($24 \cdot 60 \cdot 60$). Во время работы сценарий проверяет время последнего доступа к файлу данных счетчика. Кроме того, сценарий проверяет текущее время с помощью функции `time()`, которая возвращает текущую временную метку Unix, т.е. время в секундах, истекшее с 00:00:00 1 января 1970 года.

Если последний доступ к файлу состоялся более `$backup_interval` секунд назад, то с помощью функций `unlink()` и `copy()` создается резервная копия существующего файла данных (чтобы предотвратить удаление старых резервных копий файла данных, к имени каждого файла можно присоединить текущее время):

```
@copy($counter_file, $counterbackup_file);
```

Вывод каких-либо сообщений об ошибках, которые может генерировать функция `copy()`, можно подавить с помощью оператора `@`, предшествующего имени функции. Подавление сообщений об ошибках удобно, потому что иногда они заранее известны, не сообщают ничего нового и их можно проигнорировать.

Функция `unlink()` принимает единственный строковый аргумент — имя файла, который необходимо удалить. Например, если требуется удалить файл `trash.txt` в текущем каталоге, вызов может выглядеть так:

```
if(!unlink("./trash.txt")) die ("Невозможно удалить файл trash.txt!");
```

Почему эта функция называется `unlink()`? Использование такого имени для функции удаления может показаться странным, особенно для тех, кто не сталкивался с Unix-подобными системами. Почему эту функцию не назвали просто `delete`? В файловой системе Unix имеется различие между физическим расположением файлов на носителе и соответствующей структурой каталогов. Вполне возможно, а на практике в Unix это очень распространено, что различные части системы каталогов хранятся на физически обособленных устройствах. При сохранении файла в определенной точке файловой системы эта точка дерева каталогов (которая называется *индексный узел* (*inode*)) связывается с физическим местом хранения данных файла. В Unix путь к файлу фактически представляет собой уникальный идентификатор для одного из этих узлов.

Примечательно то, что в Unix можно связать несколько таких точек (т.е. несколько индексных узлов) с одними и теми же данными. Это делается с помощью так называемых *жестких ссылок* (*hard link*) (подробнее жесткие ссылки описываются в тексте ман-страниц для команды `ln`). Эти данные можно получить до тех пор, пока существует хотя бы одна ссылка на них. Но если все ссылки будут уничтожены, то будут также уничтожены и сами данные. Таким образом, с помощью команды `unlink()` фактически уничтожается одна из этих ссылок на данные.

Обычно существует только одна жесткая ссылка на данные файла, поэтому вызов упомянутой функции на самом деле удаляет файл. Если где-нибудь в системе существует другая ссылка, то файл фактически не удаляется. В Windows связать с одними и теми же данными несколько точек файловой системы невозможно, поэтому вызов `unlink()` для файла во всех случаях эквивалентен его удалению.

Многие версии PHP для Windows вообще не поддерживают функцию `unlink()`. Существуют различные пути решения данной проблемы, но они значительно отличаются в зависимости от используемого Web-сервера. Например, при использовании Web-сервера Apache можно передать команду `del filename` непосредственно Windows посредством функций `system()` или `exec()`. Поэтому в комбинации Windows/Apache можно имитировать функцию `unlink()` с помощью следующего кода:

```
if(!isset($_ENV['WINDIR'])) {
    $userfile = str_replace("/", "", $file);
    exec("del $file");
    if(file_exists("$file")) die("Невозможно удалить файл $file.");
}
else if(!@unlink($file)) {
    die("Невозможно удалить файл $file.");
}
```

Поскольку команда `del` ожидает присутствия в заданном пути символов обратной косой черты, в передаваемом пути с помощью функции `str_replace()` следует заменить все символы кривой черты.

Работа с каталогами

PHP позволяет манипулировать каталогами почти так же, как файлами, и предоставляет для этого множество эквивалентных функций. В некоторых из них используется дескриптор каталога, в других — строка, содержащая имя каталога. Дескриптор каталога подобен дескриптору файла; он представляет собой указывающее на каталог целое значение, которое можно получить, указав каталог в вызове функции `opendir()`:

```
$dp = opendir("/home/james/");
```

В случае возникновения ошибки данная функция возвращает `False`. Закрывать каталог можно с помощью функции `closedir()`, передав ей соответствующий дескриптор:

```
closedir($dp);
```

Функция `readdir()` возвращает следующий пункт в списке открытого каталога. Этот список включает в себя такие пункты, как `.` (используется для указания текущего каталога) и `..` (родительский по отношению к данному каталог). РНР-интерпретатор хранит внутренний указатель, связанный со следующим пунктом в списке. Этот указатель подобен указателю позиции в файле, в которой должна быть выполнена следующая файловая операция.

Практика Создание листинга каталога

Ниже показано, как создать цикл для получения всех пунктов в заданном каталоге:

```
<?php
//dir_list.php
$default_dir = "x:/home/localhost/www/php5/Ch07";
if(!($dp = opendir($default_dir))) die("Невозможно открыть каталог
$default_dir.");
while($file = readdir($dp))
if($file != '.' && $file != '..') echo "$file<br>";
closedir($dp);
?>
```

На рис. 7.8 показан пример выполнения данного сценария.

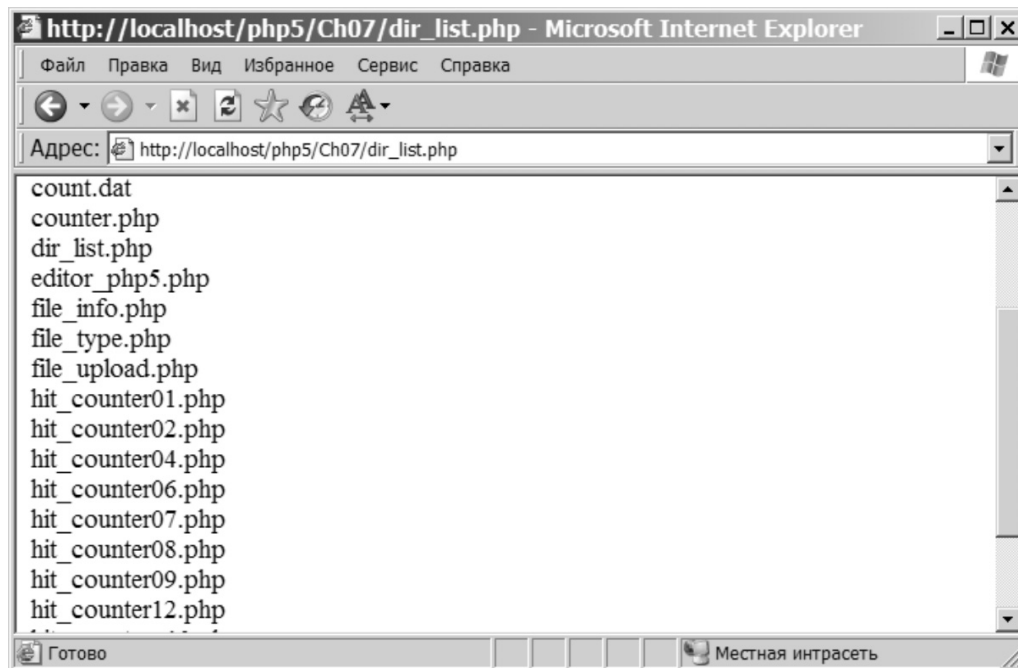


Рис. 7.8.

Как это работает

Сначала сценарий определяет дескриптор заданного каталога (в данном случае `x:/home/localhost/www/php5/Ch07`), а затем начинает цикл, который считывает записи из каталога и (если это не `.` и не `..`) распечатывает их. Цикл обусловлен

возвращаемым функцией `readdir()` значением, которое станет равным `False`, когда список записей в каталоге закончится:

```
while($file = readdir($dp))
    if($file != '.' && $file != '..') echo "$file<br>";
```

Наконец, функция `closedir()` закрывает каталог.

Возвращаемые имена файлов не сортируются. Чтобы отсортировать их, необходимо создать еще два цикла. В первом цикле имена файлов считываются в массив:

```
<?php
$default_dir = "x:/home/localhost/www/php5/Ch07";
if(!($dp = opendir($default_dir))) die("Невозможно открыть каталог
$default_dir.");
while($file = readdir($dp)) $filenames[] = $file;
closedir($dp);
```

Теперь массив `$filenames` содержит все записи листинга каталога. РНР автоматически индексирует массив.

Во втором цикле вызывается функция `sort()`, которая упорядочивает элементы массива по возрастанию и отображает все элементы, кроме ссылок на текущий и родительский каталог:

```
sort($filenames);
for($i=0; $i < count($filenames); $i++)
    if($filenames[$i] != '.' && $filenames[$i] != '..')
        echo $filenames[$i] . "<br>";
?>
```

Другие функции для обработки каталогов

Как и в случае файлов, РНР обеспечивает множество способов для манипуляции с каталогами, включая следующие функции:

- ❑ `rewinddir()`
- ❑ `chdir()`
- ❑ `rmdir()`
- ❑ `mkdir()`
- ❑ `dirname()`
- ❑ `(dir)`

Функция `rewinddir()` переустанавливает внутренний указатель в РНР, когда необходимо вернуться к первой записи в заданном каталоге. Эта функция является аналогом функции `rewind()` для файлов и требует указания соответствующего дескриптора каталога. Остальные функции обработки каталогов вместо дескриптора непосредственно принимают строку, содержащую путь к каталогу.

Функция `chdir()` изменяет текущий каталог:

```
if(chdir("x:/home/localhost/www/php5/Ch07"))
    echo "Текущий каталог: x:/home/localhost/www/php5/Ch07.";
else
    echo "Невозможно перейти в каталог x:/home/localhost/www/php5/Ch07.";
```

Функция `rmdir()` удаляет заданный каталог. Каталог должен быть пустым, а кроме этого, сценарий должен иметь необходимые права для удаления данного каталога. Например:

```
rmdir("/tmp/rubbish/");
```

Функция `mkdir()` создает каталог, указанный в качестве ее первого аргумента. Можно также задать режим доступа к каталогу в виде трехзначного восьмеричного числа. Следующий код сначала проверяет, существует ли каталог `x:/home/localhost/www/php5/Ch07/test`. Если это так, то сценарий удаляет этот каталог, а затем создает новый каталог с тем же именем и предоставляет всем пользователям все права доступа к данному каталогу (права доступа применимы только на Linux-системах):

```
$default_dir = "x:/home/localhost/www/php5/Ch07";
if(file_exists($default_dir)) rmdir($default_dir);
mkdir($default_dir, 0777);
```

Функция `dirname()` возвращает путь к каталогу для заданного имени файла. Например:

```
$filepath = "x:/home/localhost/www/php5/Ch07/index.html";
$dirname = dirname($filepath);
$filename = basename($filepath);
```

В результате в переменной `$dirname` теперь содержится строка `"x:/home/localhost/www/php5/Ch07"`, а в переменной `$filename` строка `"index.html"`.

PHP предоставляет псевдо-объектно-ориентированный механизм для работы с каталогами: класс `dir`. Чтобы использовать этот механизм, необходимо сначала создать объект путем вызова конструктора `dir()` и передачи ему имени каталога, с которым необходимо работать:

```
$dir = dir("x:/home/localhost/www/php5/Ch07");
```

Объект `dir` обладает двумя свойствами: `handle` и `path`. Они ссылаются на дескриптор каталога и путь соответственно:

```
echo $dir->handle; # распечатывает дескриптор каталога
echo $dir->path; # распечатывает строку "x:/home/localhost/www/php5/Ch07"
```

Свойство `handle` можно использовать с другими функциями обработки каталогов, такими как `readdir()`, `rewinddir()` и `closedir()`.

Объект `dir` поддерживает три метода: `read()`, `rewind()` и `close()`, которые функционально эквивалентны PHP-функциям `readdir()`, `rewinddir()` и `closedir()` соответственно. Можно переписать сценарий создания листинга каталога, используя в нем описанный объект:

```
<?php
$default_dir = "x:/home/localhost/www/php5/Ch07";
$dir = dir($default_dir);
while($file = $dir->read()) if($file != '.' && $file != '..')
    echo $file . "<br>";
$dir->close();
?>
```

Обход дерева каталогов

Как уже отмечалось в главе 6, рекурсия особенно полезна, когда сценарий должен выполнять повторяющиеся операции и обрабатывать в цикле определенное множество данных. Очень хорошим примером такого сценария является сценарий, выполняющий обход дерева каталогов. В каталоге могут содержаться подкаталоги, а также файлы. Чтобы создать сценарий, который выводит имена всех файлов и подкаталогов указанного каталога, необходимо заставить сценарий считывать записи в текущем каталоге. Если запись представляет собой имя файла, то ее следует отобразить. Если запись является именем подкаталога, то необходимо отобразить его имя, а затем войти в него и снова вернуться к подпрограмме считывания записей в текущем каталоге.

Очевидно, что второй этап в случае необходимости повторяет весь процесс. Рекурсия продолжается до тех пор, пока не останется подкаталогов для обхода. Рассмотрим пример такого сценария:

```
<?php
//nav_dir.php
$default_dir = "/home/james";
function traverse_dir($dir) {
    echo "Обход каталога $dir....<br>";
    chdir($dir);
    if(!($dp = opendir($dir))) die("Невозможно открыть каталог $dir.");
    while($file = readdir($dp)) {
        if(is_dir($file)) {
            if($file != '.' && $file != '..') {
                echo "$file<br>";
                traverse_dir("$dir/$file");
                chdir($dir);
            }
        }
        else echo "$file<BR>";
    }
    closedir($dp);
}
traverse_dir($default_dir);
?>
```

Функция `traverse_dir()` основывается на идее рекурсии и обходит все дерево каталогов ниже заданного каталога. Во-первых, функция выводит каталог, который она в настоящий момент просматривает. Затем вызов `chdir()` позволяет убедиться, что строка `$dir` соответствует пути к текущему каталогу:

```
function traverse_dir($dir) {
    echo "Обход каталога $dir....<br>";
    chdir($dir);
    if(!($dp = opendir($dir))) die("Невозможно открыть каталог $dir.");
    while($file = readdir($dp)) {
```

Рекурсивный вызов происходит, когда следующий пункт в листинге является подкаталогом. Из возвращаемого списка исключаются файлы `.` и `..` — в данном случае это крайне важно, иначе сценарий вошел бы в бесконечный цикл.

```
        if(is_dir($file)) {
            if($file != '.' && $file != '..') {
                echo "$file<br>";
```

Функция `traverse_dir()` вызывает сама себя, для того чтобы перемещаться вглубь по иерархии каталогов, и делает первоначальный каталог текущим:

```
                traverse_dir("$dir/$file");
                chdir($dir);
            }
        }
        else echo "$file<BR>";
    }
    closedir($dp);
}
traverse_dir($default_dir);
?>
```

Мощь рекурсии, продемонстрированная в данном простом сценарии, позволяет создать собственную версию Linux-команды `find`.

Сценарий для навигации по каталогам

Теперь можно перейти к созданию довольно мощного сценария для навигации по каталогам (назовем его “навигатором”), с помощью которого можно сканировать содержимое существующих каталогов и создавать новые. Для этого необходимо создать файл `common_php5.inc.php` с несколькими разделами, которые помогут приложению выполнять его работу.

Сначала следует установить каталог по умолчанию, имя файла и размер текстового поля:

```
<?php
//определение каталога по умолчанию
$default_dir = "./docs";

//определение стандартного имени для новых файлов
$default_filename = "new.txt";

//определение размера текстового поля
$edit_form_cols = 80;
$edit_form_rows = 25;
```

Определение расширений файлов, которые можно обрабатывать:

```
//определение расширений файлов для обработки
$text_file_array = array( "txt", "htm", "html", "php", "inc", "dat" );
$image_file_array = array( "gif", "jpeg", "jpg", "png" );
```

Три описанные ниже функции создают заголовок, нижний колонтитул и необходимые сообщения об ошибках:

```
function html_header() {
    ?>
    <html>
    <head><title>Добро пожаловать в текстовый редактор на Web-странице</title></head>
    <body>
    <?php
}

function html_footer() {
    ?>
    </body>
    </html>
    <?php
}

function error_message($msg) {
    html_header();
    echo "<script>alert(\"$msg\"); history.go(-1)</script>";
    html_footer();
    exit;
}
```

Обработка информации о датах:

```
function date_str($timestamp) {
    $date_str = getdate($timestamp);
    $year = $date_str["year"];
    $mon = $date_str["mon"];
    $mday = $date_str["mday"];
    $hours = $date_str["hours"];
    $minutes = $date_str["minutes"];
    $seconds = $date_str["seconds"];

    return "$hours:$minutes:$seconds $mday/$mon/$year";
}
```

Следующая функция возвращает информацию о файле, а также проверяет расширение файла:

```
function file_info($file) {
    global $text_file_array;
    $file_info_array["filesize"] =
        number_format(filesize($file)) . " bytes.";
    $file_info_array["filectime"] = date_str(filectime($file));
    $file_info_array["filemtime"] = date_str(filemtime($file));
    if(!isset($_ENV[WINDIR])) {
        $file_info_array["fileatime"] = date_str(fileatime($file));
        $file_info_array["filegroup"] = filegroup($file);
        $file_info_array["fileowner"] = fileowner($file);
    } else {
        $file_info_array["fileatime"] = "недоступно";
        $file_info_array["filegroup"] = "недоступно";
        $file_info_array["fileowner"] = "недоступно";
    }

    $extension = array_pop(explode(".", $file));

    if (in_array($extension, $text_file_array)) {
        $file_info_array["filetype"] = "текстовый";
    } else {
        $file_info_array["filetype"] = "бинарный";
    }

    return $file_info_array;
}
```

Сохраните весь код в файле `common_php5.inc.php`.

В начале сценария-навигатора необходимо подключить только что сохраненный файл:

```
<?php
//navigator.php
include "common_php5.inc.php";
```

Затем необходимо определить несколько функций. Во-первых, функцию `mkdir_form()`, которая отображает форму для создания нового каталога:

```
function mkdir_form() {
    global $dir;
    ?>
    <center>
    <form method="POST"    action="<?php echo
    "$_SERVER[PHP_SELF]?action=make_dir&dir=$dir"; ?>">
    <input type="hidden" name="action" value="make_dir">
    <input type="hidden" name="dir" value="<? echo $dir ?>">

    <?php
    echo "<strong>$dir</strong>"
    ?>

    <br>
    <input type="text" name="new_dir" size="10">
    <input type="submit" value="Создать каталог" name="Submit">

    </form>
    </center>
    <?php
    }
```

Функция `make_dir()` создает заданный каталог:

```
function make_dir() {
    global $dir;
    if(!@mkdir("$dir/${_POST[new_dir]}", 0700)) {
        error_message("Невозможно создать каталог $dir/${_POST[new_dir]}");
    }
```

```

    }
    html_header();
    dir_page();
    html_footer();
}

```

Затем функция `display()` распечатывает в новом окне содержимое заданного файла. Функция сравнивает расширение файла с элементами массивов `$text_file_array` и `$image_file_array`, определяя таким образом тип файла — текстовый, графический или бинарный:

```

function display() {
    global $text_file_array, $image_file_array;
    $extension = array_pop(explode(".", $_GET[filename]));

```

Затем расширение файла сравнивается с типами, которые определены в массивах `$text_file_array` и `$image_file_array`. Сценарий отказывается отображать двоичные файлы:

```

if(in_array($extension, $text_file_array)) {
    readfile("$_GET[dir]/$_GET[filename]");
}
else if(in_array($extension, $image_file_array)) {
    echo "<img src=\"$_GET[dir]/$_GET[filename]\">";
}
else echo "Невозможно отобразить. Файл $_GET[dir]/$_GET[filename]
не является текстовым файлом или корректно сформированным
графическим файлом.";
}

```

`dir_page()` — главная функция, она сканирует иерархию каталогов и создает листинги каталогов, отображая элементы с завершающими символами косой черты. Эта функция представляет собой ядро сценария:

```

function dir_page() {
    global $dir, $default_dir, $default_filename;

    if($dir == "") {
        $dir = $default_dir;
    }
    if (isset($_GET['dir'])) {
        $dir = $_GET['dir'];
    }

    $dp = opendir($dir);

?>
<table border="0" width="100%" cellpadding="0" cellspacing="0">
<?php

```

Создается два цикла для сортировки записей в заданном каталоге: цикл `while` для чтения всех записей в текущем рабочем каталоге и цикл `for` для отображения записей после сортировки:

```

while($file = readdir($dp)) $filenames[] = $file;
sort($filenames);
for($i = 0; $i < count($filenames); $i++)
{
    $file = $filenames[$i];

```

Если следующая запись в каталоге является точкой `"."` (т.е. указывает на текущий каталог), то функция игнорирует данную запись и начинает следующую итерацию

цикла. Кроме того, если текущий рабочий каталог является каталогом по умолчанию, то игнорируются как записи ".", так и записи "..":

```
if($dir == $default_dir && ($file == "." || $file == ".."))
    continue;
if(is_dir("$dir/$file") && $file == ".")
    continue;
if(is_dir("$dir/$file")) {
```

Если следующей является запись ".." (указывающая на родительский каталог), то функция отсекает имя текущего каталога от значения переменной \$dir, чтобы создать гиперссылку на родительский каталог:

```
if($file == ".."){
```

В переменную \$current_dir с помощью функции basename() записывается крайняя правая (после косой черты) подстрока имени каталога:

```
$current_dir = basename($dir);
```

Например, если переменная \$dir содержит значение "/home/apache/htdocs/images", то переменной \$current_dir присваивается значение "images".

При создании гиперссылки функция dir_page() с помощью ereg_replace() удаляет вхождения этого значения в переменную \$dir:

```
$parent_dir = ereg_replace("/$current_dir$", "", $dir);
```

После этого переменная \$parent_dir содержит значение "/home/apache/htdocs", так как образец /\$current_dir\$ совпадает с подстрокой "/images" в конце строки.

```
echo "<tr><td width=\"100%\" nowrap>
    <a href=\"$_SERVER[PHP_SELF]?dir=$parent_dir\">$file/
    </a></td></tr>\n";
}
```

Если следующая запись является подкаталогом, то функция создает гиперссылку на этот подкаталог:

```
else echo "<tr><td width=\"100%\" nowrap>
    <a href=\"$_SERVER[PHP_SELF]?dir=$dir/$file\">
    $file/</a></td></tr>\n";
}
```

Если следующая запись является файлом, то функция создает гиперссылку для отображения содержимого данного файла в новом окне браузера:

```
else echo "<tr><td width=\"100%\" nowrap>
    <a href=\"$_SERVER[PHP_SELF]?action=display&dir=$dir&filename=$file\"
    target=\"_blank\">$file</a></td></tr>\n";
}
?>
</table>
<?php
    mkdir_form();
}
```

Когда сценарий запускается, сначала он проверяет, не находится ли заданный каталог выше каталога по умолчанию (по причинам безопасности пользователь не должен иметь доступа к файлам, которые находятся выше его каталога по умолчанию). Функция сравнения с образцом ereg() возвращает False, если переменная \$dir не содержит значения \$default_dir, и если это так, то переменной \$dir присваивается значение \$default_dir:

```
if(empty($dir) || !ereg($default_dir, $dir)) {
    $dir = $default_dir;
}
```

Наконец, вызываются функции, соответствующие значению переменной `$action`. По умолчанию вызывается функция `dir_page()`:

```
if (!empty($_POST['action'])) {
    $action = $_POST['action'];
}
if (!empty($_GET['action'])) {
    $action = $_GET['action'];
}
switch ($action) {
    case "make_dir":
        make_dir();
        break;
    case "display":
        display();
        break;
    default:
        html_header();
        dir_page();
        html_footer();
        break;
}
?>
```

Примерный результат работы сценария показан на рис. 7.9. В данном случае сценарий распечатал имена двух файлов и одного нового каталога в текущем каталоге.

На рис. 7.10 показано содержимое файла `new01.txt`.

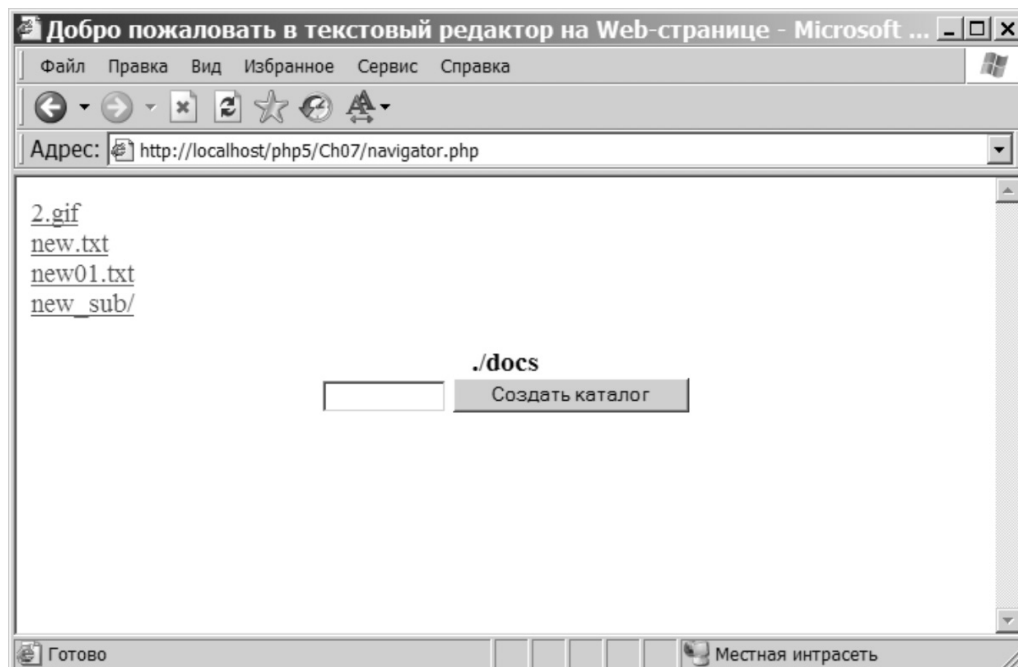


Рис. 7.9.



Рис. 7.10.

Создание текстового редактора

Зная базовые возможности PHP по обработке файлов и каталогов, можно приступить к созданию простого текстового редактора. Описанный ниже сценарий в качестве аргумента будет принимать имя файла, редактировать, а затем сохранять данный файл. Если сценарий не получает имя существующего файла, то он создает новый файл.

Во-первых, следует продумать, как в итоге будет выглядеть сценарий. Эта картина в воображении разработчика в конечном итоге трансформируется в пользовательский интерфейс. Так как это текстовый редактор, необходимо определить место, в котором сценарий сможет сохранять текст и давать пользователю возможность работать с этим текстом. Нет необходимости беспокоиться о реализации таких специфических функций редактирования, как ввод и удаление символов, перемещение курсора и т.д., поскольку HTML-дескриптор `<textarea>` способен поддерживать все необходимые функции редактирования. Все, что требуется сделать, это создать форму, на которой будет отображаться область ввода с полосой прокрутки, текстовое поле для ввода имени файла и кнопка для сохранения файла.

Затем необходимо решить, как уведомлять пользователя об ошибках и как получать от него подтверждение на выполнение возможных деструктивных действий, таких как перезапись существующего файла. Добиться этой цели позволяют весьма простые JavaScript-сценарии. В HTML-страницу можно поместить фрагменты JavaScript-кода, заключив их в следующую комбинацию тегов:

```
<script> JavaScript-код </script>
```

Например, если требуется сообщить пользователю об ошибке, можно использовать JavaScript-метод `alert()`:

```
<script> alert("Предупреждение! Возникла ошибка!"); </script>
```

Данный код открывает небольшое окно, в котором отображается заданное сообщение об ошибке. Чтобы пользователь после возникновения ошибки мог вернуться на предыдущую страницу, можно использовать метод `history.go(-1)`:

```
<script> alert("Предупреждение! Возникла ошибка! Вернитесь на предыдущую страницу!");  
history.go(-1);  
</script>
```

Получить подтверждение от пользователя также несложно. Для этой цели служит метод `confirm()`:

```
<script> result = confirm("Предупреждение! Вы уверены?");  
if(!result) history.go(-1);  
</script>
```

Метод `confirm()` возвращает значение в зависимости от решения пользователя. Если пользователь нажал кнопку **ОК**, то возвращается значение `True` и выполняется заданное действие. Если пользователь нажал кнопку **Отмена**, то переменной `result` присваивается значение `False` и браузер переходит на предыдущую страницу. Некоторые из таких приемов используются в примере сценария.

Сначала вводятся глобальные переменные, а часто используемые функции помещаются в подключаемый файл `common.inc`. Как было сказано выше, помещать в общий подключаемый файл наиболее употребительные элементы, которые повторно используются в разных частях сценария, — хорошая Практика. Назовем новую версию подключаемого файла `common_php5_02.inc.php`. Сначала сценарий реализует функцию для обработки сообщений об ошибках. Она выдает пользователю сообщение, а затем направляет пользовательский браузер обратно на предыдущую страницу:

```
<?php  
function error_message($msg) {  
    echo "<script>alert(\"$msg\"); history.go(-1)</script>";  
    exit;  
}
```

Следующий код представляет собой копию созданной ранее функции `date_str()`:

```
function date_str($timestamp) {  
    $date_str = getdate($timestamp);  
    $year = $date_str["year"];  
    $mon = $date_str["mon"];  
    $mday = $date_str["mday"];  
    $hours = $date_str["hours"];  
    $minutes = $date_str["minutes"];  
    $seconds = $date_str["seconds"];  
  
    return "$hours:$minutes:$seconds $mday/$mon/$year";  
}
```

Последней в файле `common_php5_02.inc.php` является функция `file_info()`:

```
function file_info($file) {  
    global $text_file_array;  
    $file_info_array["filesize"] = number_format(filesize($file)) . " bytes.";  
    $file_info_array["filectime"] = date_str(filectime($file));  
    $file_info_array["filemtime"] = date_str(filemtime($file));  
    if(!isset($_ENV['WINDIR'])) {  
        $file_info_array["fileatime"] = date_str(fileatime($file));  
        $file_info_array["filegroup"] = filegroup($file);  
    }
```

```
    $file_info_array["fileowner"] = fileowner($file);
} else {
    $file_info_array["fileatime"] = "недоступно";
    $file_info_array["filegroup"] = "недоступно";
    $file_info_array["fileowner"] = "недоступно";
}

$extension = array_pop(explode(".", $file));

if (in_array($extension, $text_file_array)) {
    $file_info_array["filetype"] = "текстовый";
} else {
    $file_info_array["filetype"] = "двоичный";
}
return $file_info_array;
}
```

Обратите внимание, как с помощью функции `explode(".", $file)` из заданного имени файла выделяется расширение. Функция разделяет имя файла, используя в качестве разделителя точку (`.`), и возвращает в виде массива полученные в результате части имени. Последний элемент извлекается из массива с помощью функции `array_pop()` и сохраняется в переменной `$extension`. В данном случае расширение не является вторым элементом.

В последнем блоке файла `common_php5_02.inc.php` создается массив расширений файлов:

```
// задаем расширения обрабатываемых файлов
$text_file_array = array( "txt", "htm", "html", "php", "inc", "dat" );
```

Теперь можно создавать структуру главного файла обработки, `editor_php5.php`. Сначала вводится заголовок для обычной HTML-страницы:

```
<html>
<head><title>Текстовый редактор</title></head>
<body>
<form method="POST" action="<?php echo $_SERVER['PHP_SELF']; ?>">
<input type="hidden" name="posted" value="true">
```

Затем начинается PHP-код и подключается (с помощью функции `require()`) файл `common_php5.inc.php`. После чего задается используемый каталог:

```
require("common_php5_02.inc.php");
//определение каталога по умолчанию
$dir = "./docs";
```

Следующий фрагмент кода проверяет, нажал ли пользователь кнопку, и если это так, то устанавливает соответствующее значение для окончательного оператора `select..case`:

```
if (isset($_POST['save_edited_file'])) {
    $action_chosen = "save_file";
} elseif (isset($_POST['create_new_file'])) {
    $action_chosen = "save_file"; } elseif (isset($_POST['edit_existing_file'])) {
    $action_chosen = "edit_existing_file";
}
```

Далее начинается оператор `select..case` и создается `case`-блок `save_file`. Этот код проверяет существование файла, и если файл существует, то используется JavaScript-сценарий, запрашивающий у пользователя подтверждение на перезапись данного файла:

```
switch ($action_chosen) {
    case "save_file";
```



```

        if (file_exists("$dir/${_POST[filename]}")) {
            echo "<script>result = confirm(\"Перезаписать файл
            '$dir/${_POST[filename]}'? \"); if(!result) history.go
            (-1);</script>";
        }
    }

```

Для отправки пользователю сообщения об ошибке, в случае если попытка открыть (или создать) файл оказалась неудачной, используется функция `error_message()`:

```

        if ($file = fopen("$dir/${_POST[filename]}", "w")) {
            fputs($file, $_POST[filebody]);
            fclose($file);
        } else {
            error_message("Невозможно сохранить файл $dir/${_POST[filename]}.");
        }
    }

```

Последний фрагмент кода в `case`-блоке `save_file` использует HTML для отображения простых кнопок, позволяющих пользователю выполнять доступные в приложении операции. Обратите внимание на отличие кода создания листинга каталогов для наполнения выпадающего списка. Тот же код используется в `case`-блоке по умолчанию. В конце этого блока необходимо поместить ключевое слово `break`:

```

//отображение основных кнопок
?>
<table border="1" align="center"><tr><td>
<strong>Создать (или перезаписать) новый файл или редактировать
    существующий файл</strong>
</td></tr>
<tr><td>
<input type="submit" name="create_new_file" value="Создать новый файл">
<input type="text" name="filename" value="new.txt">
</td></tr>
<tr><td>
<input type="submit" name="edit_existing_file" value="Редактировать
    существующий файл">
<select name="existing_file">
<?php

if($dp = opendir($dir)) {
    while($file = readdir($dp)) {
        if($file != '.' && $file != '..' &&
            is_file($dir."/".$file)) {
            ?>
            <option value="<?php echo $file; ?>">
                <?php echo $file; ?></option>
            <?php
        }
    }
} else {
    error_msg("Невозможно открыть каталог $dir.");
}

    closedir($dp);

    ?>
</select>
</td></tr></table>
<?php

break;

```

Рассмотрим case-блок `edit_existing_file`. Сценарий устанавливает свойства файла, включая `filebody`, и помещает результаты работы функции `file_info()` в массив `$file_info_array`:

```
case "edit_existing_file";

    $filepath = "$dir/${_POST[existing_file]}";
    $filebody = implode("",file($filepath));
    $file_info_array = file_info("$filepath");
```

Затем необходимо проверить, является ли данный файл текстовым, и если это не так, то записать в свойство `filebody` уведомление о том, что данный файл редактированию не подлежит:

```
if ($file_info_array["filetype"] != "text") {
    $filebody = $filepath . " не является текстовым файлом.
    Редактирование невозможно.";
    $editable = 0;
} else {
    $editable = 1;
}
```

Отображение информации о файле:

```
?>
<table border="1" width="70%" align="center">
<tr><th width="100%" colspan="2">
<center><strong>Информация о существующем файле <?php echo
    "$dir/${_POST[existing_file]}"; ?>
</td></tr>
<?php
$file_info_array = file_info("$dir/${_POST[existing_file]}");

foreach ($file_info_array as $key=>$val) {
    echo "<tr><th width=\"30%\">". ucfirst($key)
        . "</td><td width=\"70%\">". $val
        . "</td></tr>\n";
}
?>
</table>
```

Последние строки case-блока `edit_existing_file` позволяют пользователю отредактировать и сохранить файл:

```
<br>

<table border="1" align="center"><tr><td>
<strong>Редактирование существующего файла <?php echo
    $_POST['existing_file']; ?></strong>
</td></tr>
<tr><td>
<?php

if ($editable == 0) {
    echo $filebody;
} else {
    ?>
    <input type="hidden" name="filename" value="<?php echo
        $_POST['existing_file']; ?>">
    <textarea rows="4" name="filebody" cols="40" wrap="soft">
    <?php
    echo "$filebody";
    ?>
    </textarea><br>
    <input type="submit" name="save_edited_file" value="Сохранить файл">
    <?php
```

```

    }
    ?>
</td></tr></table>
<?php

    break;

Блок default представляет собой копию последней части case-блока save_file:
default;
    //отображение основных кнопок
    ?>
    <table border="1" align="center"><tr><td>
    <strong>Создать (или перезаписать) новый файл или отредактировать
    существующий</strong>
    </td></tr>
    <tr><td>
    <input type="submit" name="create_new_file" value="Создать новый файл">
    <input type="text" name="filename" value="new.txt">
    </td></tr>
    <tr><td>
    <input type="submit" name="edit_existing_file" value="Редактировать
    существующий файл">
    <select name="existing_file">
    <?php

        if($dp = opendir($dir)) {
            while($file = readdir($dp)) {
                if($file != '.' && $file != '..' && is_file
($dir."/".$file)) {
                    ?>
                    <option value="<?php echo $file; ?>"><?php echo $file; ?></option>
                    <?php
                    }
                } else {
                    error_msg("Невозможно открыть каталог $dir.");
                }
            }
            closedir($dp);

        ?>
    </select>
    </td></tr></table>
    <?php
    break;

```

В последней части сценария закрывается оператор `select..case`, а также завершается HTML-код для страницы, включая закрывающий тег `</form>`.

```

}
?>
    </form>
</body>
</html>

```

Данный сценарий очень хорошо иллюстрирует использование функций для работы с файловой системой.

Загрузка файлов на сервер

Созданный только что “навигатор” можно улучшить, добавив в него возможность загружать файлы с локальной машины на сервер. PHP предоставляет простой способ для внедрения в приложение подобной функциональности. Загружать файлы

с помощью браузера позволяет элемент HTML-формы `<input>`, имеющий тип `file`. Для загрузки файлов также необходимо установить в теге `<form>` атрибут `enctype` (со значением `multipart/form-data`), который при создании обычной формы, как правило, опускается.

Ниже приведен примерный код формы для загрузки файлов:

```
<form method="POST" enctype="multipart/form-data"
      action="<?php echo "$_SERVER[PHP_SELF]?action=upload_file&dir=$dir"; ?>">
  Локальный файл <input type="file" name="userfile">
  <input type="submit" name="submit" value="Загрузить">
</form>
```

Атрибут `action` данной формы должен указывать на PHP-сценарий, который будет обрабатывать загружаемый файл. Вводить второе текстовое поле необходимо только в том случае, если желательно предоставить пользователю возможность задавать имя, с которым загружаемый файл будет сохраняться на сервере.

Как только файл загружается, через суперглобальный массив `$_FILES` становятся доступны следующие переменные:

Переменная	Описание
<code>\$_FILES[userfile]</code>	Массив в <code>\$_FILES</code> , содержащий другие значения
<code>\$_FILES[userfile][name]</code>	Исходный (на пользовательской машине) путь и имя загруженного файла
<code>\$_FILES[userfile][size]</code>	Размер загруженного файла в байтах
<code>\$_FILE[userfile][type]</code>	Тип файла (если браузер предоставляет данную информацию)

Предположим, пользователь с помощью формы только что загрузил zip-файл `C:\bsdocs\projects.zip` размером 20 000 байт. В таком случае указанные переменные содержат следующие значения:

- ❑ `$_FILES[userfile][tmp_name]`: путь к файлу во временном каталоге (который устанавливается в `php.ini`) и временное имя файла в формате `"php-###"` (где `"###"` — номер, автоматически генерируемый PHP), например, `"/tmp/php-512"`.
- ❑ `$_FILES[userfile][name]`: `"C:\bsdocs\projects.zip"`.
- ❑ `$_FILES[userfile][size]`: `20000`.
- ❑ `$_FILES[userfile][type]`: `"application/x-zip-compressed"`.

Эти значения при желании можно записать в обычные переменные, например, так:

```
//определение параметров пользовательского файла
$userfile_name = $_FILES['userfile']['name'];
$userfile_tmp_name = $_FILES['userfile']['tmp_name'];
$userfile_size = $_FILES['userfile']['size'];
$userfile_type = $_FILES['userfile']['type'];
if(isset($_ENV['WINDIR'])) {
    $userfile = str_replace("\\\\", "\\", $_FILES['userfile']['name']);
}
```

Загружаемый файл сохраняется во временном каталоге, заданном в файле `php.ini`, и уничтожается по окончании данного запроса, поэтому его необходимо скопировать в какой-либо другой каталог. По соображениям безопасности для этого рекомендуется использовать функцию `move_uploaded_file()` вместо `copy()`:

```
$archive_dir = "./docs";
$filename = basename($userfile_name);
if(!@move_uploaded_file($userfile, "$archive_dir/$filename"))
    echo "Ошибка: невозможно скопировать файл $filename.";
else echo "Файл $filename успешно загружен.";
```

Можно установить ограничение на размер загружаемых файлов и с помощью переменной `$userfile_size` проверять, не превышает ли размер загружаемого файла заданное ограничение:

```
$archive_dir = "./docs";
$max_filesize = 200000;
$filename = basename($userfile_name);
if($userfile_size > $max_filesize)
    echo "Ошибка: файл $filename слишком велик. " .
        . "Максимальный размер загружаемого файла: " .
        . number_format($max_filesize) . " байтов.";
else if(!copy($userfile, "$archive_dir/$filename"))
    echo "Ошибка: невозможно скопировать файл $filename.";
else echo "Файл $filename успешно загружен.";
```

Еще один способ ограничить размер загружаемых файлов заключается в использовании скрытого поля формы:

```
<input type="hidden" name="max_file_size" value="200000">
...
if($userfile_size > $max_file_size)
    echo "Ошибка: файл $filename слишком велик. " .
        . "Максимальный размер загружаемого файла: " .
        . number_format($max_filesize) . " байт.";
```

Установка данного ограничения внутри сценария — гораздо более безопасный метод, потому что злоумышленник может отредактировать содержимое Web-страницы, содержащей форму для загрузки, и установить любое ограничение.

Практика Загрузка файлов

Ниже приведен завершенный пример реализации функции загрузки файлов:

```
<?php
//file_upload.php
function upload_form() {
    ?>
    <table border="1" align="center">
    <tr><td>
    <form method="POST" enctype="multipart/form-data"
        action="<? echo $_SERVER['PHP_SELF'] ?>">
        Выберите файл для загрузки
        <input type="file" name="userfile">
        <input type="submit" name="action" value="Загрузить">
    </form>
    </td></tr>
    </table>
    <?
}
function upload_file() {
    //установка архивного каталога
    $archive_dir = "./docs";
    //определение параметров пользовательского файла
    $userfile_name = $_FILES['userfile']['name'];
    $userfile_tmp_name = $_FILES['userfile']['tmp_name'];
    $userfile_size = $_FILES['userfile']['size'];
    $userfile_type = $_FILES['userfile']['type'];
```

```

if(isset($_ENV['WINDIR'])) {
    $userfile = str_replace("\\\\", "\\", $_FILES['userfile']['name']);
}
$filename = basename($userfile_name);
if($userfile_size <= 0) die ("Файл $filename пуст.");
if(!@move_uploaded_file($userfile_tmp_name, "$archive_dir/$filename"))
    die("Невозможно скопировать файл $userfile_name в $filename.");
if(isset($_ENV['WINDIR']) && !@unlink($userfile))
    die ("Невозможно удалить файл $userfile_name.");
echo "Файл $filename был успешно загружен.<BR>";
echo "Размер файла: " . number_format($userfile_size) . "<BR>";
echo "Тип файла: $userfile_type<BR>";
}
?>
<html>
<head><title>Загрузка файла</title></head>
<body>
<?php
if($_POST[action] == 'Загрузить') {
    upload_file();
} else {
    upload_form();
}
?>
</body>
</html>

```

На рис. 7.11 изображена форма для загрузки (в данном случае для загрузки выбран файл 2.gif).

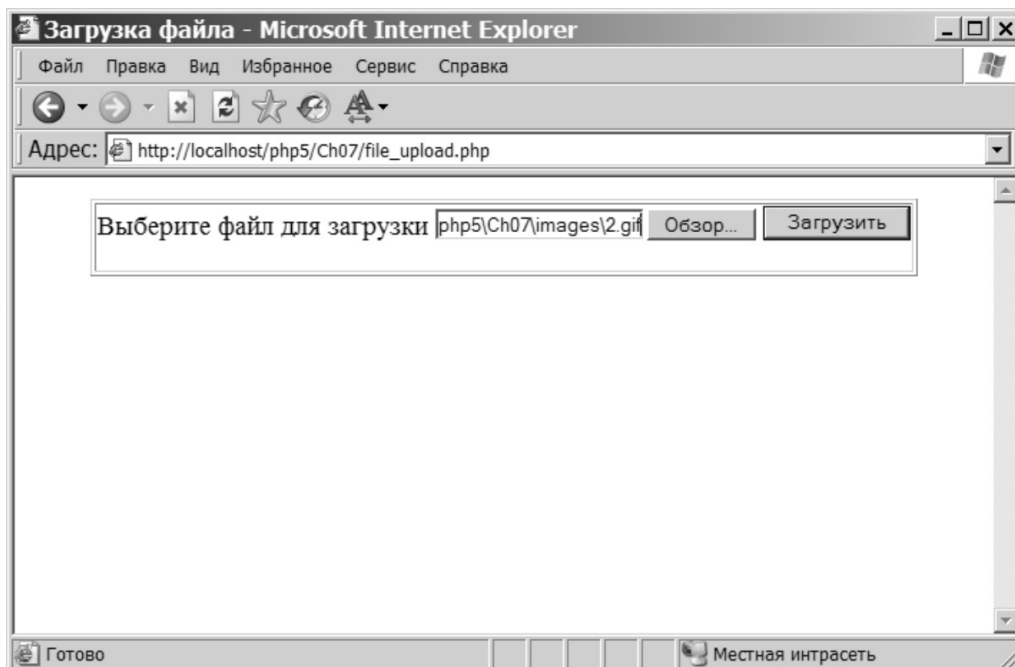


Рис. 7.11.

Пользователь, загружающий данный файл, в случае успешной загрузки увидит страницу, показанную на рис. 7.12.

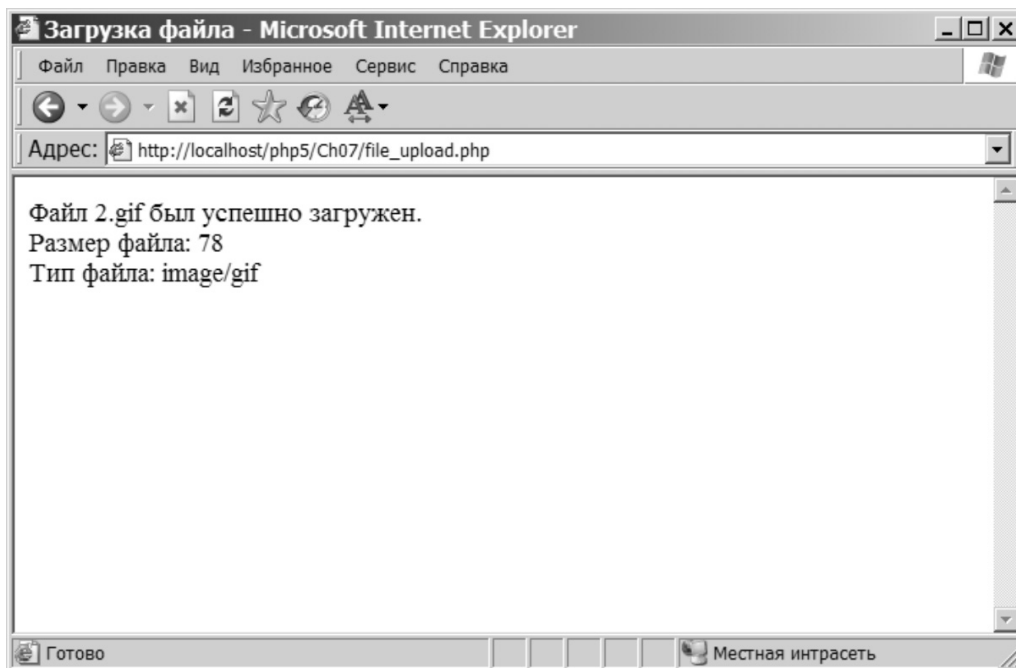


Рис. 7.12.

Как это работает

Когда пользователь загружает файл со своей локальной машины, сценарий сначала получает значения массива `$_FILES`, а затем определяет платформу, на которой работает. Если таковой является Windows, то все вхождения двойной обратной косой черты необходимо заменить на одинарную обратную косую черту:

```
//установка архивного каталога
$archive_dir = './docs';
//определение параметров пользовательского файла
$userfile_name = $_FILES['userfile']['name'];
$userfile_tmp_name = $_FILES['userfile']['tmp_name'];
$userfile_size = $_FILES['userfile']['size'];
$userfile_type = $_FILES['userfile']['type'];
if(isset($_ENV['WINDIR'])) {
    $userfile = str_replace("\\\\", "\\", $_FILES['userfile']['name']);
```

Если размер файла меньше нуля, то пользователь отправил форму, не указав загружаемый файл:

```
if($userfile_size <= 0) die ("Файл $filename пуст.");
```

Если все в порядке, то сценарий копирует загруженный файл в архивный каталог и определяет платформу, на которой выполняется. Если это не Windows, то для удаления временного файла можно использовать функцию `unlink()`. После этого выводится сообщение о том, что файл загружен, а также размер и тип данного файла.

Резюме

В настоящей главе рассматривались функции для работы с файлами, в том числе функции для чтения и записи файлов. На примерах было продемонстрировано создание простого текстового редактора на Web-странице, а также работа с каталогами и создание сценария для обхода дерева каталогов.

Данная глава охватывает большинство основных PHP-функций для работы с файлами и каталогами. В некоторых из них используется дескриптор файла или каталога — указатель, ссылающийся на открытый файл или каталог. Работа с такими функциями состоит из следующих этапов:

1. Открытие файла или каталога с помощью функции `fopen()` или `opendir()`.
2. Работа с открытым файлом или каталогом с помощью функций `fread()`, `fgets()`, `fpasssthru()` и `readdir()` др.
3. Закрытие файла или каталога с помощью функций `fclose()` или `closedir()`.

Наконец, в главе рассматривалось создание текстового редактора, который может сканировать дерево каталогов и манипулировать записями в нем.

В главе не обсуждались некоторые редко применяемые в Web-приложениях функции. Полный перечень функций для работы с файлами и каталогами можно просмотреть на Web-сайте PHP <http://php.net/manual/ref.filesystem.php>.

Упражнение

Создайте PHP-приложение, которое можно использовать для поиска определенного каталога в правильно заданном родительском каталоге. Приложение должно просматривать указанный каталог, а также все каталоги, которые в нем находятся.

8

XML

XML (eXtensible Markup Language — расширяемый язык разметки) — сравнительно новая идея. Первоначально XML разрабатывался как удобный для человека способ обмена структурированными данными, но впоследствии весьма быстро получил широкое распространение также как средство хранения структурированной информации. Хотя XML во многом отличается от баз данных, он, как и базы данных, представляет собой метод форматирования и/или постоянного хранения данных, а также имеет свои преимущества и недостатки.

XML на самом деле является не языком, а спецификацией для создания собственных языков разметки. XML представляет собой подмножество языка SGML (Standard Generalized Markup Language — стандартный обобщенный язык разметки, предшественник HTML). XML предназначен для упрощения обмена данными между несовместимыми приложениями. Форматирование XML-документов очень похоже на форматирование HTML-документов. В отличие от HTML, который имеет фиксированное множество дескрипторов и атрибутов, определенных в спецификации HTML, XML позволяет писать собственные дескрипторы (которые также называются элементами) и атрибуты и, таким образом, предоставляет возможность определения собственного языка в XML- или любом другом определении. В сущности, с помощью XML можно форматировать любые данные.

В дополнение к этому XML предоставляет средства для создания определений данных, которые будут доступны другим приложениям в сети. Таким образом, два приложения, которые ничего “не знают” друг о друге, но оба способны анализировать XML-документы, смогут обмениваться данными друг с другом.

По этим причинам XML быстро становится стандартом обмена данными, а в PHP5 имеется множество новых средств и функций, которые ускоряют работу с XML-данными и делают ее эффективнее и понятнее. В PHP5 включены все XML-функции PHP4, а также новые функции, основанные на simpleXML (встроенное PHP-расширение, поддерживающее общие XML-операции).

В данной главе рассматриваются основы XML, а также чтение и запись XML-документов с помощью функций семейства `xml_parser` и `simpleXML`-функций. Здесь также представлена документная объектная модель (Document Object Model, DOM) и описано использование XML в Web-службах. Наконец, на практике рассматривается обработка XML-документов с помощью `simpleXML`-функций.

Что такое XML

XML представляет собой спецификацию для создания собственного языка разметки. Документ, созданный согласно правилам XML-спецификации, выглядит аналогично HTML-документу — он содержит представленные в виде тегов дескрипторы и атрибуты. Тем не менее, браузеры не могут непосредственно форматировать и отображать XML-документы, если они так или иначе не получили информацию о форматировании.

XML-документы удобочитаемы, кроме того, существует множество приложений, предназначенных для разбора XML-документов и эффективной работы с их содержимым. В PHP5 имеются новые связанные с XML функции, которые можно легко использовать для работы с XML-документами или преобразования не-XML-данных в XML-документы.

Создать XML-документ довольно просто:

```
<?xml version="1.0" ?>
<php_programs>
  <program name="cart">
    <price>100</price>
  </program>
  <program name="survey">
    <price>500</price>
  </program>
</php_programs>
```

В первой строке документа указывается используемая версия XML (следует отметить, что разделители `<?xml` и `?>` очень похожи на PHP-разделители, поэтому необходимо убедиться, что в начале PHP-кода используется полный тег `<?php`). Во второй строке определяется корневой элемент документа (который называется `php_programs`). В XML-документе может быть только один корневой элемент. В третьей строке определяется дочерний по отношению к корневому элемент, который называется `program`. Элемент содержит атрибут с именем `name`, значение которого равно `cart`.

Очевидно, что созданный здесь XML-документ описывает PHP-программы (`cart` и `survey`) и что цена программы `cart` равна 100, а цена программы `survey` — 500. Корневой элемент может содержать множество элементов (в данном примере используется два элемента `program`, входящих в состав корневого элемента). Как и HTML-документ, XML-документ по существу составляется из элементов и атрибутов.

Другой особенностью XML-документов является то, что в качестве содержимого элементов могут использоваться другие элементы, а также простой текст, тогда как атрибутам значения могут присваиваться непосредственно. XML-документы, как уже отмечалось, довольно легко читать, однако существует множество программ-анализаторов, упрощающих программную обработку этих документов.

Написание XML-документов не составляет труда, многие программисты создают приложения и собственные языки программирования для обработки XML-документов: как для чтения существующих документов, так и для создания новых. XML-спецификацию можно использовать свободно; Консорциум W3 (World Wide Web

Consortium; сайт www.w3.org, на котором также поддерживаются XHTML- и XML-спецификации) выпустил и поддерживает последние версии данного стандарта.

Писать XML-документы можно, комбинируя любые элементы и атрибуты, однако часто необходимо использовать предопределенные элементы и атрибуты, так чтобы при обмене данными с другим человеком или приложением обе стороны транзакции точно знали, что означают названия элементов и атрибутов. Предопределенные элементы и атрибуты указываются в определении типа документа (document type definition, DTD) или с помощью языка XML Schema. Оба эти способа обсуждаются в данной главе далее. Часто обнаруживается, что перед написанием XML-документа необходимо либо найти, либо написать собственное DTD-определение или XML-схему. DTD можно опубликовать в Web. Это позволит всем желающим писать (или получать и обрабатывать) XML-документы, совместимые с данным определением.

Структура XML-документа

При обсуждении XML-документов часто употребляются понятия *правильно сформированный* и *корректный*. Правильно сформированный XML-документ организован согласно основным синтаксическим правилам (которые будут описаны далее), а корректный документ кроме этого соответствует правилам DTD или XML Schema.

Базовое требование для XML-документов заключается в том, чтобы они были правильно сформированными; неправильно сформированный XML-документ, по сути, не является XML-документом. Такой документ подобен сценарию, содержащему критическую синтаксическую ошибку; внешне такой сценарий похож на другие РНР-сценарии, но на самом деле не является таковым до тех пор, пока все синтаксические ошибки не будут устранены. Правильно сформированный XML-документ может содержать любые элементы, атрибуты или другие допускаемые XML-спецификацией конструкции. Однако не существует правил, определяющих возможные имена элементов и атрибутов (кроме основных правил именования, которые в действительности не очень строги) или возможное содержимое элементов. Именно в этой расширяемости и заключается большая часть эффективности и пользы XML; поэтому при условии соблюдения базовых правил XML-спецификации ограничений на то, что можно добавить или изменить, нет.

Правильно сформированный документ не обязательно является корректным, но корректный документ всегда правильно сформирован, иначе его невозможно было бы прочесть. Если документ правильно сформирован и содержит ссылку на DTD-определение или XML Schema, то XML-анализатор (parser) может обратиться к этому DTD или Schema и определить, является ли документ корректным. XML-документ корректен, если его элементы, атрибуты и другие конструкции соблюдают правила DTD- или Schema-определения. По определению DTD или Schema содержит правила, регламентирующие содержащиеся в документе элементы или атрибуты, допустимые данные для этих элементов или атрибутов и т.д. По сути, главной целью наличия DTD или Schema является точное определение того, какие элементы и атрибуты допускаются и какие данные они могут содержать.

Ссылка на DTD или Schema ограничивает пары имя/значение (элементы, атрибуты и многие другие XML-конструкции), которые могут присутствовать в XML-документе. Однако важное преимущество заключается в том, что приложения, ничего “не знающие” друг о друге, могут эффективно обмениваться данными при условии, что они в состоянии анализировать XML (т.е. могут прочесть правильно сформированный документ и распознать его содержимое, если оно корректно). Другая особен-

ность, которая делает XML столь мощной технологией, заключается в том, что удобочитаемость XML-документов и свойства DTD или XML Schema позволяют узнать, что в частности означают определенные элементы и атрибуты.

Основные части XML-документа

XML-документ может содержать необязательный пролог, а затем обязательный корневой элемент (включая любое его содержимое и другие элементы и атрибуты), а также необязательный раздел в конце для других данных. В следующем перечне определены требования для основных разделов.

- ❑ XML-документ должен содержать строку `xml version`, которая может включать в себя объявление кодировки символов.
- ❑ Корректный XML-документ содержит DTD или XML Schema либо ссылку на одно из этих определений, если они хранятся вне документа.
- ❑ XML-документ обычно содержит один или несколько элементов, каждый из которых может иметь один или несколько атрибутов. Между начальным и конечным тегом атрибута могут находиться или отсутствовать другие элементы или данные.
- ❑ XML-документы могут содержать дополнительные компоненты, такие как инструкции для определенных приложений; CDATA-разделы, которые могут содержать специальные символы, входящие в сценарии, но не допустимые в обычных XML-данных; записи, комментарии, ссылки (например, псевдонимы для специальных символов), текст и другие объекты.

Ниже приведен пример правильно сформированного и корректного XML-документа:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE Client SYSTEM "http://www.example.com/dtds/client.dtd">
<clients>
  <client>Joe</client>
  <client>Jim</client>
</clients>
```

Обратите внимание, что ссылка на внешнее DTD-определение задана с помощью URL. Это означает, что документ может быть проверен путем чтения указанного DTD и проверки документа на соответствие DTD-определению. Конечно, можно самому прочитать документ и сравнить его с элементами, атрибутами и другими компонентами, определенными в DTD, однако существует множество приложений, которые способны автоматически проверять корректность XML-документа по DTD-определению или XML-схеме. А поскольку DTD или схема доступны непосредственно в документе или в Web, приложения могут легко выполнять функцию проверки автоматически во время синтаксического разбора документа.

Правильно сформированные XML-документы

В правильно сформированном XML-документе соблюден синтаксис, описанный в спецификации XML. Синтаксис, естественно, определяется несколькими базовыми правилами; наиболее распространенные из них перечислены ниже.

- ❑ Существует только один родительский элемент, содержащий все остальные элементы в документе.

- ❑ XML-документ должен (хотя это и не является строгим правилом) начинаться с XML-объявления, содержащего номер используемой версии XML. Например:

```
<xml version="1.0">
```
- ❑ В строку с номером версии XML можно включать объявления кодировок символов, а если используется кодировка, отличная от UTF-8 или UTF-16, то это обязательно. Код может выглядеть так:

```
<xml version="1.0" encoding="UTF-8">
```
- ❑ Если XML-документ содержит DTD или ссылку на XML Schema, то они должны появляться перед первым элементом в документе.
- ❑ XML-документ может состояться из начальных и конечных тегов (которые очень похожи на HTML-теги) или из одинарных тегов с терминатором (как тег `
` в XHTML). В отличие от HTML не допускается использовать элементы, имеющие только начальный тег без терминатора. Элементы с начальными и конечными тегами считаются не пустыми (т.е. они могут заключать в себе какое-либо содержимое), тогда как в пустых тегах содержимого нет (пустые элементы иногда сами по себе обозначают что-либо, как, например, тег `
` в XHTML):

```
<client>John Doe</client>
<break />
```
- ❑ XML-атрибуты могут записываться внутри не пустых XML-элементов и должны иметь имя и значение; значение должно быть заключено в разделители, например, в двойные кавычки. В одном элементе имя одного атрибута может появляться только однажды.
- ❑ Должны соблюдаться правила вложения элементов: начальный и конечный теги какого-либо определенного элемента должны находиться снаружи от начального и конечного тега дочернего элемента и между начальным и конечным тегами родительского элемента. Например:

```
//неправильно
<parent><child></parent></child>
//правильно
<parent><child></child></parent>
```
- ❑ CDATA-разделы (например, разделы данных, которые составляют сценарии) должны быть заключены в теги `[CDATA и]`.
- ❑ XML-элементы не могут иметь имя, в котором используется комбинация “xml”, “XML” или любая другая комбинация букв верхнего и нижнего регистров в данной последовательности. Имена должны начинаться с буквы, символа подчеркивания или двоеточия, но на практике двоеточие не используется. Имена чувствительны к регистру. В именах также можно использовать цифры, дефис и точку, но только после первого символа имени.
- ❑ Комментарии заключаются в такие же теги, как и HTML-комментарии (`<!-- и -->`).

Использование XML-элементов и атрибутов

XML-элементы и их атрибуты формируют иерархическую структуру XML-документа и заключают в себе его содержимое (содержимым XML-документа являются его данные). Хотя в документе может быть только один корневой элемент. Корневой

элемент может содержать множество элементов с одинаковыми именами (эти элементы часто называются дочерними элементами), в свою очередь дочерние элементы тоже могут содержать множество элементов с одинаковыми именами. Поэтому документ может выглядеть так:

```
<clients>
  <client ID="1">Joe</client>
  <orders>
    <order ID="1">ProductA</order>
    <order ID="2">ProductB</order>
  </orders>
  <client ID="2">Jim</client>
  <orders>
    <order ID="1">ProductA</order>
    <order ID="2">ProductB</order>
  </orders>
</client></client>
</clients>
```

Очевидно, что одна часть данных документа помещена в элементы (имя каждого клиента находится между начальным и конечным элементами `client`), а другая часть данных представлена в виде значений атрибутов (идентификаторы клиентов и их заказов указываются в атрибутах элементов `client` и `order`).

Когда же следует использовать атрибут и когда элемент, содержащий данные? На этот счет нет жесткого правила. Однако на практике рекомендуется использовать элемент тогда, когда существует вероятность, что одни и те же данные придется указывать несколько раз (например, в настоящее время у клиента может быть только один заказ, но можно ожидать, что в дальнейшем у этого клиента будет несколько заказов). Использовать атрибут следует, когда есть уверенность, что данные появляются только однажды (например, каждый клиент может иметь один и только один идентификатор).

Корректные XML-документы: DTD-определения и XML-схемы

DTD-определения представляют собой специальные документы, написанные в формате EBNF (Extended Backus Naur Format — расширенный формат Бэкуса-Наура), который не является XML-языком и поэтому его синтаксический анализ весьма сложен. В DTD-определениях задаются ограничения на XML-элементы, атрибуты, содержимое и т.д. XML-схемы служат той же цели, но написаны на языке XML Schema; их можно легко анализировать и обрабатывать в том же приложении, которое используется для чтения XML-документа. XML-схемы также более удобны для определения деталей элементов и атрибутов (таких как типы данных, диапазоны значений и т.д.), чем DTD. По этой причине многие создатели XML-документов предпочитают использовать XML-схемы. Перед первым элементом XML-документа могут задаваться ссылки, как на DTD-определения, так и на XML-схемы. Кроме того, существуют другие способы включения DTD-определений и XML-схем в XML-документы (эти способы рассматриваются ниже).

Корректность некоторых или всех элементов и содержимого документа можно проверить по DTD или схеме, если DTD или схема присутствует в документе, либо на них задана ссылка. Первостепенное значение проверенного XML-документа заключается в том, что обрабатывающее его приложение получает некоторые сведения о содержимом документа, например, о том, сколько раз определенный элемент может встречаться в другом элементе, какие значения и атрибуты допускаются и т.д.

Как уже отмечалось, написать XML-документ несложно. Также не составляет труда определить DTD или XML-схему, согласно которой будет проверяться созданный

XML-документ. Консорциум W3C преобразовал следующую версию HTML в XHTML, используя существующее DTD-определение для HTML (HTML всегда основывался на формальном DTD) с очень небольшими изменениями, касающимися определения всех элементов, атрибутов и других компонентов, допустимых в XHTML. Главное отличие между HTML и XHTML заключается в том, что XHTML-документ должен соответствовать XML-спецификации, а для HTML-документов это не обязательно.

Существуют другие сложности: браузеры отображают HTML-документы, даже если они неправильно сформированы. Но правильно сформированный XHTML-документ они отображают как XML, если файл имеет расширение .xml, и как обычную Web-страницу, когда файл имеет расширение .htm или .html. Безусловно, для отображения XHTML-документа как обычной Web-страницы ссылка на XHTML DTD должна быть корректной, а документ должен быть правильно сформированным. В нескольких последующих разделах рассматривается часть DTD-определения для XHTML, ссылки на DTD в XHTML-документах и то, как подобные документы отображаются в браузерах при разных расширениях файла (.xml и .htm).

DTD-определение для XHTML

Для стандарта XHTML1 имеется три DTD-определения. Они расположены на страницах:

- ❑ www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd
- ❑ www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd
- ❑ www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd

Эти DTD-определения дополняют свои HTML-аналоги и, по сути, являются очень простыми. Если ввести в браузере указанные ссылки, то можно получить DTD-определение в виде обычного текста.

Следующий код показывает, как пишется DTD (строгая версия) для языка XHTML, но только для элемента IMG. DTD-определение для HTML также используется и для XHTML (с очень небольшими отличиями, которые гарантируют, что XHTML-документы соответствуют стандарту XML), хотя фактически только XHTML соответствует спецификации XML. Это означает, что в XHTML присутствуют все HTML-элементы и атрибуты, но в XHTML-документе необходимо строго придерживаться правил, продиктованных спецификацией XML (например, о правильном вложении и закрытии элементов).

```
<!--
  Чтобы не создавать проблем для пользователей, браузеры
  которых не отображают изображения, следует предоставлять
  текстовое описание, используя атрибуты alt и longdesc.
  Кроме того, следует избегать использования серверных
  карт-изображений.
  Замечание: в данном DTD-определении отсутствует
  атрибут name. Такой элемент доступен только в переходном и
  фреймовом DTD.
-->

<!ELEMENT img EMPTY>
<!ATTLIST img
  %attrs;
  src      %URI;          #REQUIRED
  alt      %Text;         #REQUIRED
  longdesc %URI;          #IMPLIED
  height   %Length;       #IMPLIED
  width    %Length;       #IMPLIED
  usemap   %URI;          #IMPLIED
```

```

    ismap          (ismap)          #IMPLIED
  >
<!-- usemap указывает на элемент карты-изображения,
      который может присутствовать в текущем документе
      или во внешнем документе, хотя последний случай
      поддерживается не так широко -->

```

В приведенном выше примере (код написан в формате EBNF) в первой строке, которая следует за комментариями, имеется идентификатор `ELEMENT`, имя элемента `img` и свойство `EMPTY` (элемент не содержит данных между несуществующими начальным и конечным тегами). Однако несмотря на то, что элемент формально пуст, его атрибут `src` *содержит* данные в форме URI-адреса (в данном случае это то же самое, что и URL), указывающего на месторасположение графического файла.

После идентификатора `ELEMENT` следует перечень атрибутов, которые можно включать в тег `img` в XHTML-документе. Читатели, знакомые с HTML и XHTML, несомненно догадаются, что атрибут `src` представляет собой URL (или URI), который указывает на месторасположение файла изображения, а также на то, что данный атрибут является обязательным (`REQUIRED`).

Таким образом, данная часть DTD для XHTML-документов указывает на то, что в такие документы допускается включать элемент `IMG`. Если в каком-либо XHTML-документе указана ссылка на данное DTD-определение (на все определение, а не только на эту часть) и документ включает в себя элемент `img` с соответствующим атрибутом `src`, то можно сказать, что данный документ корректен (по крайней мере, если рассматривать элемент `img`). Однако если попытаться включить в документ элемент с именем `imgе`, `image` или `images`, то XML-анализатор, проверяющий данный документ, сгенерирует ошибку, потому что в DTD данные элементы не определены и, таким образом, документ не является корректным. Следует также отметить, что хотя в HTML-документе элемент `img` можно не закрывать, в XHTML-документе правильное закрытие этого элемента является обязательным.

Ссылки на DTD-определения и XML-схемы

Для проверки XML-документа в нем должна присутствовать ссылка на внешний файл, содержащий DTD-определение или XML-схему, либо в данный документ должно быть включено само DTD-определение или XML-схема. Ссылки на XML-схемы более сложны, поэтому сначала рассмотрим ссылки на DTD-определения.

Для создания ссылки на внешнее DTD-определение используется объявление `DOCTYPE`. Данное объявление предоставляет некоторую информацию о том, как обнаружить DTD, а также имя данного DTD. Например, следующий код показывает, как с помощью URL-адреса задается ссылка на DTD:

```

<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

```

Идентификатор `html` после `<!DOCTYPE` в первой строке указывает, что корневой элемент имеет имя `html` и является обязательным. Если DTD — внешний документ, то это определение может находиться где угодно и идентифицируется по URI-идентификатору, который понятен и доступен приложению, а не только по URL-адресу в Internet.

Значительное ограничение DTD заключается в том, что в документе можно сослаться только на одно внешнее DTD-определение, здесь также отсутствует DTD-поддержка для добавления префиксов к элементам или именам атрибутов, хотя для документа, ссылающегося на DTD, можно вызывать пространство имен.

Пространство имен представляет собой определение источника имен для элементов и атрибутов (поскольку рассматривается XML). Обозначение источника элемента или имени атрибута означает, что одно и то же имя можно использовать для представления различных данных в одном документе. В XML-документе пространство имен может быть идентифицировано с помощью ссылки на него посредством специального зарезервированного в XML ключевого слова `xmlns` (XML Namespace):

```
xmlns = "http://www.w3.org/1999/xhtml"
```

Данный URL-адрес представляет официальное пространство имен XHTML. Имена элементов и атрибутов для данного пространства имен определены внутри XHTML DTD, а атрибут `xmlns` служит только для определения пространства имен для корневого элемента XHTML-документа (тега `html`). Такой способ определения пространства имен для корневого элемента также подходит для определения пространства имен для остальных элементов и атрибутов документа.

Внешние XML-схемы

Сослаться на XML-схему можно, указав с помощью URI расположение документа XML-схемы. Обычно для этого в XML-документе используется атрибут `xmlns` как часть корневого элемента. Значением данного атрибута является месторасположение схемы. Так определяется пространство имен и программа-анализатор “знает”, где искать данную XML-схему.

Чтобы задать ссылку на XML-схему, в корневой элемент можно добавить атрибут `xmlns`:

```
<?xml version="1.0" encoding="UTF-8"?>
<customer xmlns="http://www.example.com/customer.xsd" cust_id="1">
  <cust_name>John Doe</cust_name>
</customer>
```

Естественно, в данном случае предполагается, что документ XML-схемы уже написан, расположен в корневом каталоге Web-сайта `http://www.example.com` и называется `customer.xsd`, а также что в нем определены элементы `customer` (с атрибутом `cust_id`) и `cust_name`. Хотя в данной книге подробности написания XML-схем не рассматриваются, достаточно сказать, что XML Schema — гораздо более развитый язык для определения элементов, атрибутов и других компонентов языка, соответствующего XML. Кроме того, он проще в обработке, поскольку XML Schema написан согласно главным принципам XML-спецификации.

Для документов, которые можно сверить с XML-схемой, с помощью атрибута `xmlns` можно объявлять любое количество пространств имен, каждое из которых будет связано с внешней XML-схемой. Например, можно использовать одну XML-схему, в которой элемент `farm` означает площадь, отведенную под сельскохозяйственные нужды, и другую схему, в которой элемент `farm` означает несколько серверов, выполняющих одну и ту же задачу. Если понадобится создать XML-документ, использующий оба элемента (например, описывающий, как ферма управляет своими серверами), то придется найти какой-либо способ, чтобы отличать эти элементы.

Так в одном документе допускаются ссылки на обе XML-схемы: можно использовать атрибут `xmlns` для идентификации их по URL-адресу, а также создать префиксы, с помощью которых можно будет отличать имена элементов друг от друга. Для этого подойдет следующий код:

```
xmlns:agri = "http://www.example.com/agricultural.xml"
xmlns:serv = "http://www.example.com/server.xml"
```

Впоследствии любой элемент с префиксом `agri:` будет определяться схемой `agricultural.xml`, а любой элемент с префиксом `serv:` будет определяться схемой `server.xml`. Это предотвратит путаницу в значении элементов.

Написание XML-документов с помощью XHTML

Для XHTML-документа также требуется указывать пространство имен. DTD-определения не допускают множественных ссылок, но задать одно пространство имен все-таки можно. Более того, этого требует спецификация XHTML.

Чтобы написать XHTML-документ, сначала необходимо указать номер используемой версии XML, ввести объявление DOCTYPE, которое ссылается на XHTML DTD, а затем вставить атрибут `xmlns`, указывающий на пространство имен для данного документа. Вставка атрибута `xmlns` в корневой элемент приводит к тому, что корневой элемент и все его дочерние элементы по умолчанию определяются заданным DTD. Рассмотрим пример:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <HEAD>
    <title>Пример XHTML-документа</title>
  </head>
  <body>
    <p>Данный документ представляет собой пример XHTML-документа.
      Документ может содержать изображения , а также
      гиперссылки <a href="http://example.com/">example.com</a>
      и любые другие HTML-элементы.
    </p>
  </body>
</html>
```

Данный документ выглядит почти так же, как и обычный HTML-документ, а если сохранить его с расширением `.htm` или `.html`, то в большинстве браузеров он будет отображаться как обычная Web-страница, написанная на HTML. Вместе с тем данный документ соответствует XML-спецификации и является не только правильно сформированным, но и корректным. Если сохранить данный документ с расширением `.xml`, то в большинстве браузеров он будет отображаться в XML-формате.

Web-службы

Еще одним примером эффективности XML является конструкция Web-служб. Web-службой (Web Service) называется модуль программной логики, доступный через Internet, а название “XML Web-служба” применимо к Web-службам, доступ к которым можно получить посредством XML-языков.

Как работают Web-службы и почему они считаются такими полезными? Вспомним, как определяется, а затем вызывается функция в PHP-программе. Сначала программист пишет функцию, назначает ей имя, перечень параметров и добавляет всю необходимую для ее работы программную логику. Затем при необходимости функцию можно вызывать, просто указав ее имя и передав ей соответствующие параметры.

Однако предположим, что то же самое можно было бы сделать через Internet, обратившись к предопределенным функциям (и, следовательно, к другим хранилищам информации, включая базы данных), указывая функции по их URL-адресам и именам и передавая им соответствующие параметры. Это означало бы, что можно создать

приложение, теоретически распределенное в Internet (т.е. независимое от того, где расположена программная логика или хранятся данные).

Это именно то, что можно сделать с помощью Web-служб. Вызов функций, написанных другим разработчиком, использование созданных другими программистами баз данных или даже использование множества функций и баз данных в любой точке Internet — вот для чего предназначены Web-службы. Однако для доступа к Web-службам требуется специальное средство, потому что Web-службы могут работать на разных платформах, использовать любые языки и базы данных; кроме того, имеются некоторые проблемы при передаче данных. Здесь полезными оказываются технологии SOAP и WSDL.

- SOAP (Simple Object Access Protocol — простой протокол доступа к объектам) — XML-язык, позволяющий определять для отправки и получения конверт, тело и другие компоненты вызовов Web-служб. Вызовы Web-служб помещаются в SOAP-конверт.
- WSDL (Web Service Description Language — язык описания Web-служб) — другой XML-язык, который используется для определения имени, типа и аргументов, связанных с вызовом Web-службы.

Web-службы являются одним из наиболее важных аспектов применения XML. Существует множество доступных связанных с Web-службами приложений, которые облегчают разработку на PHP как Web-служб, так и обращающегося к этим службам клиентского кода. (Тема Web-служб выходит за рамки данной книги, более подробная информация по этой теме представлена в книге *Professional PHP Development*.)

PHP и XML

Почти с самого начала существования PHP в данном языке присутствуют встроенные функции для подключения, получения данных и манипуляции данными в базах данных. Позднее, по мере того как спецификация XML приобретала широкую популярность как средство обмена данными и хранения данных, в PHP появлялись функции, облегчающие работу с XML-документами.

Ввиду специфической природы и формата XML-документов большая часть работы по добавлению XML-функций в PHP сосредоточена на правильном синтаксическом анализе XML-документов и их обработке с сохранением соответствия со спецификацией XML. Чтобы эффективно анализировать и обрабатывать XML-документы, данные функции должны быть способны работать с именами и значениями элементов и атрибутов, а также со многими другими компонентами XML-документов.

Далее рассматриваются XML-функции, которые были встроены в PHP в течение нескольких последних лет, включая такие нововведения, как расширение `simpleXML`. Сначала обсуждаются XML-функции, которые были встроены в PHP4, затем рассматриваются расширения `simpleXML` и DOM (Document Object Model — документная объектная модель) и, наконец, расширения PHP5.

XML-функции в PHP4

PHP5 сохраняет обратную совместимость с многими функциями PHP4, поэтому данный раздел начинается с обсуждения некоторых XML-функций PHP4, после чего рассматриваются новые XML-функции, доступные в PHP5. Функции семейства `xml_parser`

в PHP4 задействуют программу Джеймса Кларка (James Clark) `expat` (синтаксический анализатор XML 1.0, написанный на языке C). `Expat`-анализаторы могут определить, является ли XML-документ правильно сформированным, но не проверяют корректность XML-документов. Поэтому анализаторам, созданным на основе `expat`, необходимо передавать правильно сформированные XML-документы. В противном случае будет сгенерирована ошибка (место возникновения ошибки можно локализовать).

Ниже перечислены некоторые наиболее распространенные функции.

- ❑ `xml_parser_create`: базовая функция для создания XML-анализатора, который затем можно использовать с другими XML-функциями для чтения и записи данных, определения ошибок, а также для многих других полезных задач. По окончании работы с созданным анализатором рекомендуется использовать функцию `xml_parser_free` для освобождения ресурсов.
- ❑ `xml_parse_into_struct`: разбирает XML-данные в массив. Эту функцию можно использовать для передачи содержимого правильно сформированного XML-файла в PHP-массив, и впоследствии работать с элементами этого массива.
- ❑ `xml_get_error_code`: определяет код ошибки XML-анализатора (определяется как константа, например, `XML_ERROR_NONE` и `XML_ERROR_SYNTAX`). Чтобы с помощью данного кода получить текстовое описание ошибки, используется функция `xml_error_string`.
- ❑ `xml_set_option`: существует несколько параметров, которые можно задавать для XML-анализатора: `XML_OPTION_CASE_FOLDING` и `XML_OPTION_TARGET_ENCODING`. По умолчанию используется первый параметр. Его включение означает, что имена элементов будут записываться в верхнем регистре. Вторым параметром позволяют указать используемую кодировку для целевого документа; по умолчанию действует кодировка, используемая функцией `xml_parser_create` — `ISO-8859-1`. Чтобы определить, какие параметры для XML-анализатора установлены в текущий момент, можно использовать функцию `xml_parser_get_option`.

Существует также большое число других, связанных с XML-анализаторами функций для установки обработчиков различных типов (для распространенных XML-компонентов, например инструкций, символьных данных и т.д.).

Практика Создание XML-документа

В следующем примере для создания XML-документа используются обычные строки и строковые функции PHP. Примечательно то, что сценарий работает вообще без XML-функций. Он получает введенные имена и значения для нескольких элементов и атрибутов, а затем использует эти данные для создания XML-документа, после чего сохраняет этот документ в файл. Недостаток такого сценария в том, что он предоставляет очень слабые возможности по созданию или обработке документов произвольной сложности; можно использовать только два элемента с двумя атрибутами каждый, кроме того, нет никакой проверки корректности документа. Но иногда все, что нужно, это вывести XML-код (как показано ниже, это можно сделать без использования XML-функций).

Для работы данного сценария необходимо создать подкаталог (в Web-каталоге, содержащем файл сценария) с именем `xml_files`. В данном каталоге будут сохраняться созданные XML-файлы (убедитесь в том, что задан правильный каталог по умолчанию, `$default_dir`, если он отличается от указанного в примере). Запустите HTML-редактор, создайте файл `php_xml.php` и введите в него следующий код:

```

<html>
<head>
<title>XML-функции PHP</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body bgcolor="#FFFFFF">
<form method="POST" action="php_xml.php">
<input type="hidden" name="posted" value="true">
<table width="100%" border="1" cellpadding="10">
  <tr><td><h2>Использование XML-средств PHP</h2>
<?php
if (isset($_POST['posted'])) {
    $cmdButton = $_POST['cmdButton'];
    $root_element_name = $_POST['root_element_name'];
    $element01_name = $_POST['element01_name'];
    $att0101_name = $_POST['att0101_name'];
    $att0101_value = $_POST['att0101_value'];
    $att0102_name = $_POST['att0102_name'];
    $att0102_value = $_POST['att0102_value'];
    $element0101_name = $_POST['element0101_name'];
    $att010101_name = $_POST['att010101_name'];
    $att010101_value = $_POST['att010101_value'];
    $att010102_name = $_POST['att010102_name'];
    $att010102_value = $_POST['att010102_value'];
    switch ($cmdButton) {
        case "Создать XML-документ";
            //формирование XML-документа
            $xml_dec = "<?xml version='1.0' encoding='UTF-8'?">";
            $doc_type_dec = "";
            $root_element_start = "<" . $root_element_name . ">";
            $root_element_end = "</" . $root_element_name . ">";
            $xml_doc = $xml_dec;
            $xml_doc .= $doc_type_dec;
            $xml_doc .= $root_element_start;
            $xml_doc .= "<" . $element01_name . " "
                . $att0101_name . "=" . "'" . $att0101_value . "'" . " "
                . $att0102_name . "=" . "'" . $att0102_value . "'" . ">";
            $xml_doc .= "<" . $element0101_name . " "
                . $att010101_name . "=" . "'" . $att010101_value . "'" . " "
                . $att010102_name . "=" . "'" . $att010102_value . "'" . ">";
            $xml_doc .= "</" . $element0101_name . ">";
            $xml_doc .= "</" . $element01_name . ">";
            $xml_doc .= $root_element_end;
            //открыть файл, скопировать в него XML-текст, а затем закрыть его
            $default_dir = "./xml files";
            $fp = fopen($default_dir . "\\\" . $_POST['xml_file_name'] . ".xml", 'w');
            $write = fwrite($fp, $xml_doc);
            $response_message = "XML-документ создан";
            break;
        default;
            break;
    }
}
?>

<table width="100%" border="1"><tr>
  <td><font face="Arial, Helvetica, sans-serif" size="-1"><b>
    Создание правильно сформированного XML-документа
  </b></font></td>
</tr><tr><td><table width="100%" border="1"><tr>
  <td colspan="3"><font size="-1"><b><font face="Arial,
    Helvetica, sans-serif">Ответ:
  <?php echo $response_message; ?></font></b></font></td>
</tr><tr>
  <td><font size="-1"><b><font face="Arial, Helvetica, sans-serif">

```

```

        Элемент или атрибут</font></b></font></td>
        <td><font size="-1"><b><font face="Arial, Helvetica,
        sans-serif">Имя</font></b></font></td>
        <td><font size="-1"><b><font face="Arial, Helvetica,
        sans-serif">Значение</font></b></font></td>
    </tr><tr>
        <td><font size="-1"><b><font face="Arial, Helvetica,
        sans-serif">Корневой элемент:</font></b></font></td>
        <td><input type="text" name="root_element_name">
    </td><td>&nbsp;</td></tr><tr>
        <td><font size="-1"><b><font face="Arial, Helvetica,
        sans-serif">Элемент01:</font></b></font></td>
        <td><input type="text" name="element01_name">
    </td><td>&nbsp;</td></tr><tr>
        <td><font size="-1"><b><font face="Arial, Helvetica,
        sans-serif">Атрибут0101:</font></b></font></td>
        <td><input type="text" name="att0101_name">
        <td><input type="text" name="att0101_value">
    </td></tr><tr>
        <td><font size="-1"><b><font face="Arial, Helvetica,
        sans-serif">Атрибут0102:</font></b></font></td>
        <td><input type="text" name="att0102_name">
        <td><input type="text" name="att0102_value">
    </td></tr><tr><td>
<font size="-1"><b><font face="Arial, Helvetica, sans-serif">Элемент0101:
        </font></b></font></td>
        <td><input type="text" name="element0101_name">
    </td><td>&nbsp;</td></tr><tr>
        <td><font size="-1"><b><font face="Arial, Helvetica,
        sans-serif">Атрибут010101:</font></b></font></td>
        <td><input type="text" name="att010101_name">
        <td><input type="text" name="att010101_value">
    </td></tr><tr>
        <td><font size="-1"><b><font face="Arial, Helvetica,
        sans-serif">Атрибут010102:</font></b></font></td>
        <td><input type="text" name="att010102_name">
        <td><input type="text" name="att010102_value">
    </td></tr><tr>
        <td><b><font face="Arial, Helvetica, sans-serif">Текущие
        XML-файлы</font></b></td><td><td>
<?php
$default_dir = "./xml_files";
if (!(($dp = opendir($default_dir))) {
    die("Невозможно открыть каталог $default_dir.");
}
while ($file = readdir($dp)) {
    if ($file != '.' && $file != '..') {
        echo "$file<hr>";
    }
}
closedir($dp);
?>
<font size="-1"><b><font face="Arial, Helvetica, sans-serif">Имя
        файла нового XML-документа:</font></b></font>
    </td><td colspan="2" valign="bottom">
        <input type="text" name="xml_file_name" size="30">
    </td></tr><tr><td>&nbsp;</td>
        <td colspan="2">
            <input type="submit" name="cmdButton" value="Создать XML-документ">
        </td></tr></table></td></tr>
</td></tr></table>
</form>
</body>
</html>

```

Сохранив данный файл как `php_xml.php`, запустите его в браузере и с помощью формы (рис. 8.1) создайте правильно сформированный XML-документ.

The screenshot shows a Microsoft Internet Explorer window titled "XML-функции PHP - Microsoft Internet Explorer". The address bar shows "http://localhost/php5/Ch08/php_xml.php". The main content area has the heading "Использование XML-средств PHP" and a sub-heading "Создание правильно сформированного XML-документа". Below this is a form with the following sections:

- Ответ: XML-документ создан**
- Элемент или атрибут** table with columns: Элемент или атрибут, Имя, Значение.

Элемент или атрибут	Имя	Значение
Корневой элемент:	<input type="text"/>	<input type="text"/>
Элемент01:	<input type="text"/>	<input type="text"/>
Атрибут0101:	<input type="text"/>	<input type="text"/>
Атрибут0102:	<input type="text"/>	<input type="text"/>
Элемент0101:	<input type="text"/>	<input type="text"/>
Атрибут010101:	<input type="text"/>	<input type="text"/>
Атрибут010102:	<input type="text"/>	<input type="text"/>
- Текущие XML-файлы**
 - first_xml.xml
- Имя файла нового XML-документа:**
- Создать XML-документ** button

The status bar at the bottom shows "Готово" and "Местная интрасеть".

Рис. 8.1.

Имя первого XML-документа, созданного с помощью данной формы — `first_xml.xml`. Если в сценарии использовались те же имена элементов и значения, что и в книге, то файл `first_xml.xml` будет выглядеть так:

```
<?xml version="1.0" encoding="UTF-8" ?><rootelement><element01
attr0101="attr0101 val" attr0102="attr0102 val"><element0101
attr010101="attr010101 val" attr010102="attr010102 val"
/></element01></rootelement>
```

Откройте данный `.xml`-файл в браузере. Результат должен выглядеть аналогично рис. 8.2 при условии, что используемый браузер способен анализировать XML.

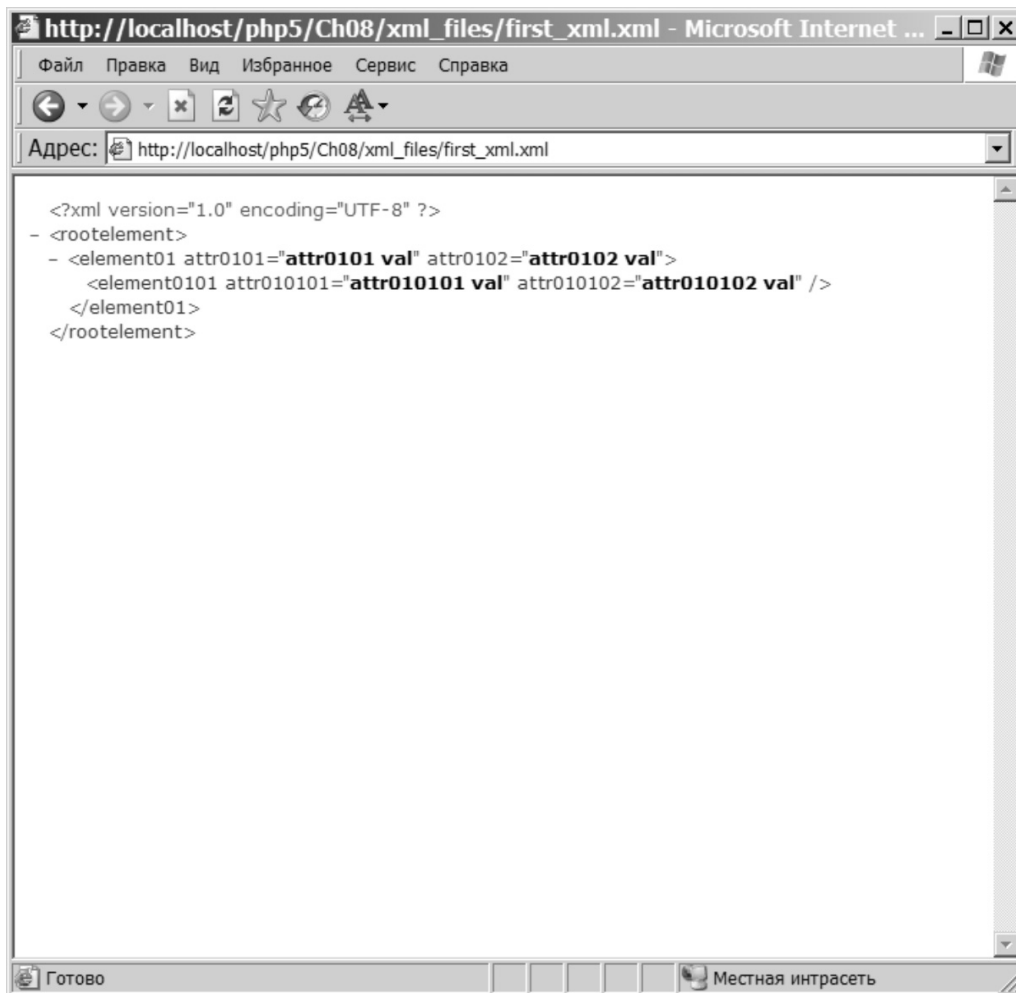


Рис. 8.2.

Как это работает

Сценарий генерирует стандартную Web-страницу, состоящую из HTML-кода. На странице отображается таблица с формой, в которую пользователь может вводить имена и значения для фиксированного набора XML-элементов и атрибутов. Функция `isset()` позволяет определить факт отправки формы, и если форма отправлена, то запускается код. После извлечения значений из `$_POST`-переменной используется блок `switch case`, определяющий следующий выполняемый блок сценария. Хотя в данном случае существует только один `case`-блок, при необходимости такие блоки можно добавить:

```
<?php
if (isset($_POST['posted'])) {
    $cmdButton = $_POST['cmdButton'];
    $root_element_name = $_POST['root_element_name'];
    $element01_name = $_POST['element01_name'];
```



```

$att0101_name = $_POST['att0101_name'];
$att0101_value = $_POST['att0101_value'];
$att0102_name = $_POST['att0102_name'];
$att0102_value = $_POST['att0102_value'];
$element0101_name = $_POST['element0101_name'];
$att010101_name = $_POST['att010101_name'];
$att010101_value = $_POST['att010101_value'];
$att010102_name = $_POST['att010102_name'];
$att010102_value = $_POST['att010102_value'];
switch ($cmdButton) {
    case "Создать XML-документ";

```

В данном примере в case-блоке выполняется форматирование введенных пользователем данных в строку, содержащую все необходимые компоненты правильно сформированного XML-документа:

```

//формирование XML-документа
$xml_dec = "<?xml version='1.0' encoding='UTF-8'?">";
$doc_type_dec = "";
$root_element_start = "<" . $root_element_name . ">";
$root_element_end = "</" . $root_element_name . ">";
$xml_doc = $xml_dec;
$xml_doc .= $doc_type_dec;
$xml_doc .= $root_element_start;
$xml_doc .= "<" . $element01_name . " "
    . $att0101_name . "=" . "'" . $att0101_value . "'" . " "
    . $att0102_name . "=" . "'" . $att0102_value . "'" . ">";
$xml_doc .= "<" . $element0101_name . " "
    . $att010101_name . "=" . "'" . $att010101_value . "'" . " "
    . $att010102_name . "=" . "'" . $att010102_value . "'" . ">";
$xml_doc .= "</" . $element0101_name . ">";
$xml_doc .= "</" . $element01_name . ">";
$xml_doc .= $root_element_end;

```

Затем открывается файл и в него записывается сформированный XML-документ (необходимо правильно указать каталог, используемый по умолчанию, если он отличается от указанного в книге):

```

//открыть файл, скопировать в него XML-текст, а затем закрыть его
$default_dir = "./xml_files";
$fp = fopen($default_dir . "\\\" . $_POST['xml_file_name'] . ".xml", 'w');
$write = fwrite($fp, $xml_doc);

```

Наконец, создается сообщение и данный блок кода заканчивается оператором break:

```

$response_message = "XML-документ создан";
break;

```

XML-анализаторы

Итак, очевидно, что XML-документы можно создавать с помощью обычных, строковых PHP-функций. Однако ясно также и то, что эти функции не представляют простого способа работы с XML-документом. Можно было бы написать какие-либо регулярные выражения и специальные функции, упрощающие работу с XML, но разработчики PHP уже создали множество полезных функций.

В следующем примере демонстрируется использование функций `xml_parser_create()` и `xml_parse_into_struct()`. Кроме того, в приведенном ниже сценарии используется функция `file_get_contents()`, которая позволяет получить содержимое файла и передать его в строку.

Практика Чтение XML-документа

Для изучения данного примера можно скопировать файл `php_xml.php` под именем `php_xml02.php`. Затем в него необходимо будет вставить новый `case`-блок и новую таблицу для отображения результатов.

1. Создайте копию файла `php_xml.php`, сохраните ее с именем `php_xml02.php` и откройте в HTML-редакторе.
2. Вставьте следующий код вместо прежнего блока, который начинается с `if (isset` и заканчивается `switch ($cmdButton)`:

```
if (isset($_POST[posted])) {

    $xml_file_chosen = $_POST[xml_file_chosen];
    $cmdButton = $_POST[cmdButton];
```

```
    switch ($cmdButton) {
```

3. Теперь вместо прежнего `case`-блока вставьте следующий код:

```
    case "Проанализировать XML-документ";

        //найти заданный файл
        $default_dir = "./xml_files";
        $xml_string = file_get_contents ($default_dir . "/" . $xml_file_chosen, "rb");

        //Прочитать имеющиеся данные и превратить их в массивы
        $xp = xml_parser_create();
        xml_parse_into_struct($xp, $xml_string, $values, $keys);
        xml_parser_free($xp);

        break;
```

4. Вместо прежней HTML-таблицы вставьте новую, код которой показан ниже:

```
<table width="100%" border="1">
  <tr><td><font face="Arial, Helvetica, sans-serif" size="-1">
  <b>Проанализировать XML-документ</b></font></td></tr><tr><td>
    <table width="100%" border="1"><tr><td>
      <font face="Arial, Helvetica, sans-serif" size="-1">
  <b>Выберите XML-файл</b></font></td></tr><tr>
    <td><select name="xml_file_chosen">

      <?php
        $default_dir = "./xml_files";
        if (!($dp = opendir($default_dir))) {
          die("Невозможно открыть каталог $default_dir.");
        }
        while ($file = readdir($dp)) {
          if ($file != '.' && $file != '..') {
            echo "<option value='$file'>$file</option>\n";
          }
        }
        closedir($dp);
      ?>

    </select></td></tr><tr>
    <td><font face="Arial, Helvetica, sans-serif" size="-1">
    <b>Содержимое XML-файла</b></font><hr>

    <?php
      if ($cmdButton == "Проанализировать XML-документ") {
```

```

        echo "Массив ключей<BR><BR><PRE>";
        print_r($keys);
        echo "</PRE><BR><BR>Массив значений<BR><BR><PRE>";
        print_r($values);
        echo "</PRE>";
    }
    ?>
</td><td>
<input type="submit" name="cmdButton" value="Проанализировать XML-документ">
</td></tr></table>
</td></tr></table>

```

5. Сохраните данный файл и вызовите его в браузере. Значение атрибута action необходимо изменить на новое имя файла, `php_xml02.php`. На странице в выпадающем списке должен отобразиться XML-файл, созданный с помощью исходного сценария `php_xml.php` (а также остальные файлы, если было создано несколько XML-файлов). Выберите файл и нажмите кнопку Проанализировать XML-документ. На рис. 8.3 показаны выводимые на экран массивы ключей и значений.

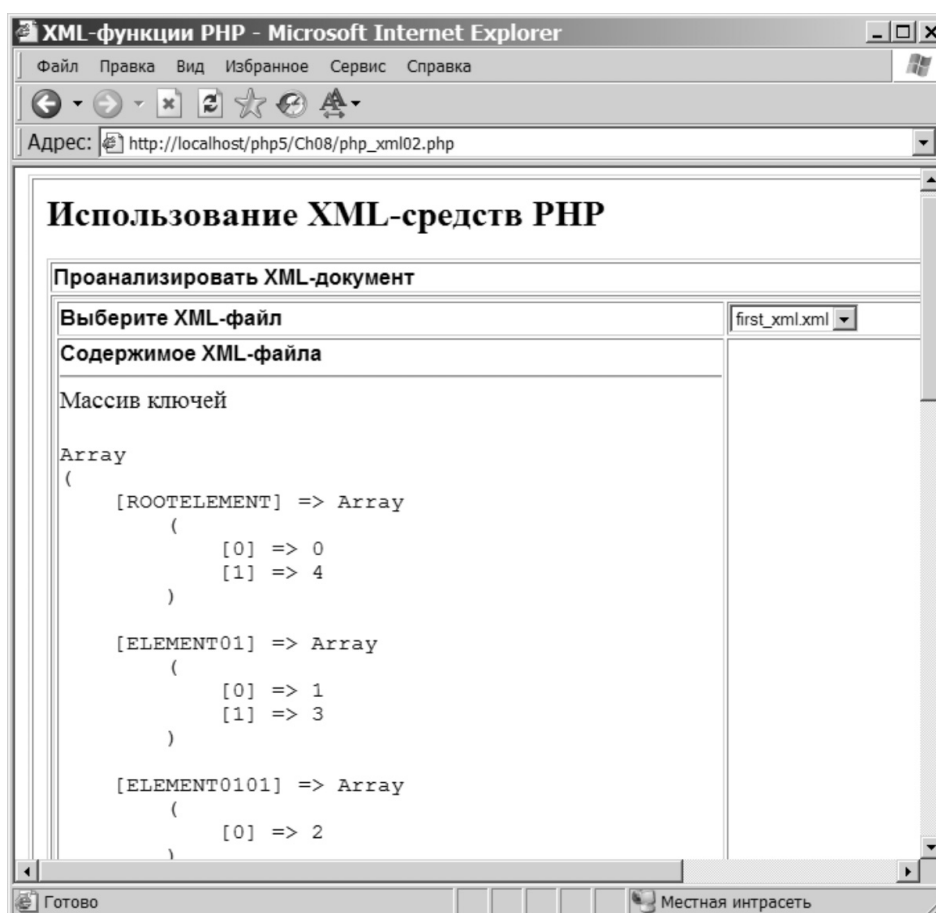


Рис. 8.3.

Как это работает

Case-блок начинается с поиска выбранного пользователем файла и считывания из него данных в переменную `$xml_string`. После этого для создания нового объекта-анализатора (с именем `$xp`) используется функция `xml_parser_create`, а функция `xml_parse_into_struct()` превращает XML-документ в два массива: массив ключей и массив значений.

Функция `xml_parse_into_struct()` принимает в качестве аргументов XML-анализатор (объектная переменная `$xp`), строковую переменную (содержимое XML-документа), а также имена двух массивов, которые необходимо создать:

```
//найти заданный файл
$default_dir = "./xml_files";
$xml_string = file_get_contents ($default_dir . "/" .
$xml_file_chosen,"rb");

//Прочитать имеющиеся данные и превратить их в массивы
$xpr = xml_parser_create();
xml_parse_into_struct($xpr, $xml_string, $values, $keys);
xml_parser_free($xpr);

break;
```

HTML-таблица предоставляет средство для выбора необходимого .xml-файла. Затем, если была нажата кнопка Проанализировать XML-документ, сценарий посредством функции `print_r()` распечатывает на экране два массива. Для того чтобы данные отображались в удобочитаемом виде, используется HTML-тег `<PRE>`:

```
<table width="100%" border="1">
  <tr><td><font face="Arial, Helvetica, sans-serif" size="-1">
<b>Проанализировать XML-документ</b></font></td></tr><tr><td>
  <table width="100%" border="1"><tr><td>
    <font face="Arial, Helvetica, sans-serif" size="-1">
<b>Выберите XML-файл</b></font></td></tr><tr><td>
      <td><select name="xml_file_chosen">

        <?php
          $default_dir = "./xml_files";
          if (!($dp = opendir($default_dir))) {
            die("Невозможно открыть каталог $default_dir.");
          }
          while ($file = readdir($dp)) {
            if ($file != '.' && $file != '..') {
              echo "<option value='$file'>$file</option>\n";
            }
          }
          closedir($dp);
        ?>

      </select></td></tr><tr><tr>
      <td><font face="Arial, Helvetica, sans-serif" size="-1">
<b>Содержимое XML-файла</b></font><hr>

      <?php
      if ($cmdButton == "Проанализировать XML-документ") {
        echo "Массив ключей<BR><BR><PRE>";
        print_r($keys);
        echo "</PRE><BR><BR>Массив значений<BR><BR><PRE>";
        print_r($values);
        echo "</PRE>";
      }
    }
  }
</td></tr></table>
```

```

}
    ?>
</td><td>
<input type="submit" name="cmdButton" value="Проанализировать XML-документ">
</td></tr></table>
</td></tr></table>

```

Объектная модель документа

Объектная модель документа (Document Object Model, DOM) представляет собой иерархическую модель для взаимодействия с документами. Она делает возможным доступ к частям документа путем непосредственного обращения к ним.

XML-документы можно моделировать с помощью DOM, так как между частями любого XML-документа существуют специфические отношения. Как уже было показано, может существовать только один корневой элемент, и он является родительским для всех остальных элементов в XML-документе. Это означает, что корневой документ находится внизу (отсюда и название) иерархии или дерева, из которого “произрастают” остальные элементы. Следовательно, отношения между компонентами XML-документа могут быть выведены программно (именно это и делают функции DOM-расширения PHP; это расширение подробнее рассматривается далее).

Элементы, расположенные внутри другого элемента, являются дочерними для этого элемента, тогда как элемент, внутри которого он расположен, является для него родительским. Поэтому элементы можно представлять как родителей и детей или как дерево с корнем, ветвями и листьями. Внутри модели DOM допустимы обе абстракции. Элементы и другие компоненты XML-документа внутри модели DOM считаются *узлами*.

DOM-расширение

DOM-расширение в PHP строго придерживается рекомендации Консорциума W3C, касающейся второго уровня DOM, которая гласит: “Модель DOM представляет собой API (application programming interface — программный интерфейс приложений) для корректных HTML- и правильно сформированных XML-документов”. DOM позволяет программно создавать и осуществлять навигацию в любом правильно сформированном XML-документе, а также добавлять, редактировать и удалять из него узлы.

Чтобы данное расширение было доступно в сценариях, необходимо скомпилировать PHP с параметром `--with-dom=dom_dir`. Для инсталляции PHP в Windows необходимо скопировать файл `libxml2.dll` или `iconv.dll` в каталог `System32`. Дальнейшие инструкции приведены в документации.

Поддержка функций DOM-расширения на момент написания книги остается экспериментальной (поэтому в данной книге нет примеров его использования), в книге *Профессиональное программирование на PHP* (ИД “Вильямс”, 2006 г) DOM-расширение PHP освещается более подробно.

Использование функций DOM-расширения PHP

В PHP имеются следующие DOM-функции: `domxml_new_doc()` для создания новых XML-документов, `domxml_open_file()` для открытия файла XML-документа в виде DOM-объекта, а также `domxml_open_mem()` для создания DOM-объекта из уже имеющегося в памяти XML-документа. Функции возвращают DOM-объект, а не строку или данные другого типа. Чтобы использовать PHP DOM-функции, обычно сначала необходимо создать объект `DOMDocument`, а затем манипулировать им, используя методы, которые

являются частью класса созданного объекта. Существует множество доступных классов объектов. Для начала можно использовать DOMDocument-объект и получить новые объекты, отражающие такие компоненты XML-документа, как элементы, атрибуты и т.д.

Например, чтобы открыть созданный ранее XML-файл, можно использовать следующий код, создающий DOM-объект с именем `$my_dom_obj`:

```
$my_dom_obj = domxml_open_file("first_xml.xml");
```

Затем, чтобы создать переменную, представляющую найденный в документе корневой элемент, можно использовать такой код:

```
$the_root_element = $my_dom_obj->document_element();
```

Манипулировать созданными DOM-объектами можно также с помощью нескольких других PHP DOM-функций, включая `create_element()`, которая создает новый элемент, и `append_child()`, которая добавляет какой-либо элемент в качестве дочернего для существующего элемента.

Например, если в XML-документе, скомпонованном согласно DOM, требуется найти определенный элемент, то можно воспользоваться функцией `get_element_by_tagname` DOMElement-объекта, позволяющей находить элемент как узел в DOM, и он будет найден как дочерний узел в DOM-иерархии корень-ветвь-лист.

В DOM-расширении доступно множество других классов объектов, а также большое количество функций для создания и манипулирования XML-документами. Объектно-ориентированная природа данных функций широко раскрывается в книге *Профессиональное программирование на PHP* (ИД “Вильямс”, 2006 г), в которой можно найти более подробную информацию о DOM и PHP.

XML-функции PHP5

PHP4 включает в себя некоторые базовые функции для синтаксического анализа XML, а в PHP5 имеется множество новых функций, основанных на библиотеке libxml2. Поддержка расширения simpleXML в PHP5 включена автоматически, поэтому нет необходимости подключать какие-либо дополнительные расширения. PHP5 также поддерживает проверку корректности XML-документов при обращении к DTD или XML-схеме.

Расширение SimpleXML

Расширение SimpleXML также является экспериментальным, но в PHP5 оно полностью переработано. Данное расширение устанавливается по умолчанию (параметр `--enable-simplexml`). Оно включает в себя функции для работы с XML-документами, которые значительно упрощают распространенные операции. Например, расширение позволяет сценарию принимать строку, преобразовывать ее в XML-документ и отображать этот документ. Главное преимущество заключается в том, что XML-документ становится объектом, который можно обрабатывать, как любые другие объекты в PHP, причем доступ к элементам и атрибутам документа, а также к их данным осуществляется с помощью обычных объектных операций (объекты более подробно рассматриваются в последующих главах).

В состав SimpleXML-расширения входят следующие функции:

- ❑ `simplexml_load_file`: принимает в качестве аргумента путь к файлу и, если содержимое файла является правильно сформированным XML-документом, то выгружает данные файла в виде объекта;

- ❑ `simplexml_load_string`: принимает в качестве аргумента строку, которая должна быть правильно сформированным XML-документом, и преобразовывает эту строку в объект;
- ❑ `simplexml_import_dom`: принимает узел из DOM-документа и превращает его в `simplexml`-узел;
- ❑ `simplexml_element->asXML`: возвращает правильно сформированную XML-строку из `simpleXML`-объекта;
- ❑ `simplexml_element->attributes`: предоставляет определения атрибутов и значений внутри правильно сформированной XML-строки;
- ❑ `simplexml_element->children`: данный метод предоставляет дочерние элементы заданного элемента;
- ❑ `simplexml_element->xpath`: метод выполняет XPath-запрос по `simpleXML`-узлу.

Использование функции `simplexml_load_string()`

В PHP-программе можно написать (или получить как результат какого-либо выражения или функции) строку, представляющую собой правильно сформированный XML-документ. Эту строку с помощью функции `simplexml_load_string()` можно превратить в `simpleXML`-объект. Чтобы преобразовать значение строковой переменной `$string` в `simpleXML`-объект на основании XML-кода внутри PHP-программы, необходимо использовать код, аналогичный следующему:

```
<?php
$string = <<<XML
<?xml version='1.0'?> <root_element>
  <child01>First element</child01>
  <child02>Second element</child02>
</root_element>
XML;
$xml_string = simplexml_load_string($string);
?>
```

Сначала рассмотрим чтение XML-строки с помощью функции `simplexml_load_string`. Все что требуется сделать, это передать в данную функцию строку. Как только `simpleXML`-объект создан, можно использовать его метод `asXML()` для отображения правильно сформированной XML-строки.

Ниже приведен пример использования функции `simplexml_load_string` и метода `asXML` объекта `simplexml_element`. Откройте текстовый редактор и создайте .php-документ, содержащий следующий код:

```
<?php
//Метод asXML форматирует данные родительского объекта в
//строку XML версии 1.0. Создаем XML-строку
$my_xml_string = <<<XML
<a>
  <b>
    <c>text content</c>
    <c>more text content</c>
  </b>
  <d>
    <c>even more text content</c>
  </d>
</a>
XML;
```

```
//загрузка строки в объект  
$xml_object = simplexml_load_string($my_xml_string);  
//отображение содержимого XML-объекта  
echo $xml_object->asXML();  
?>
```

Сохраните сценарий как `create_xml_doc.php` и откройте его в браузере. Примерный результат показан на рис. 8.4.

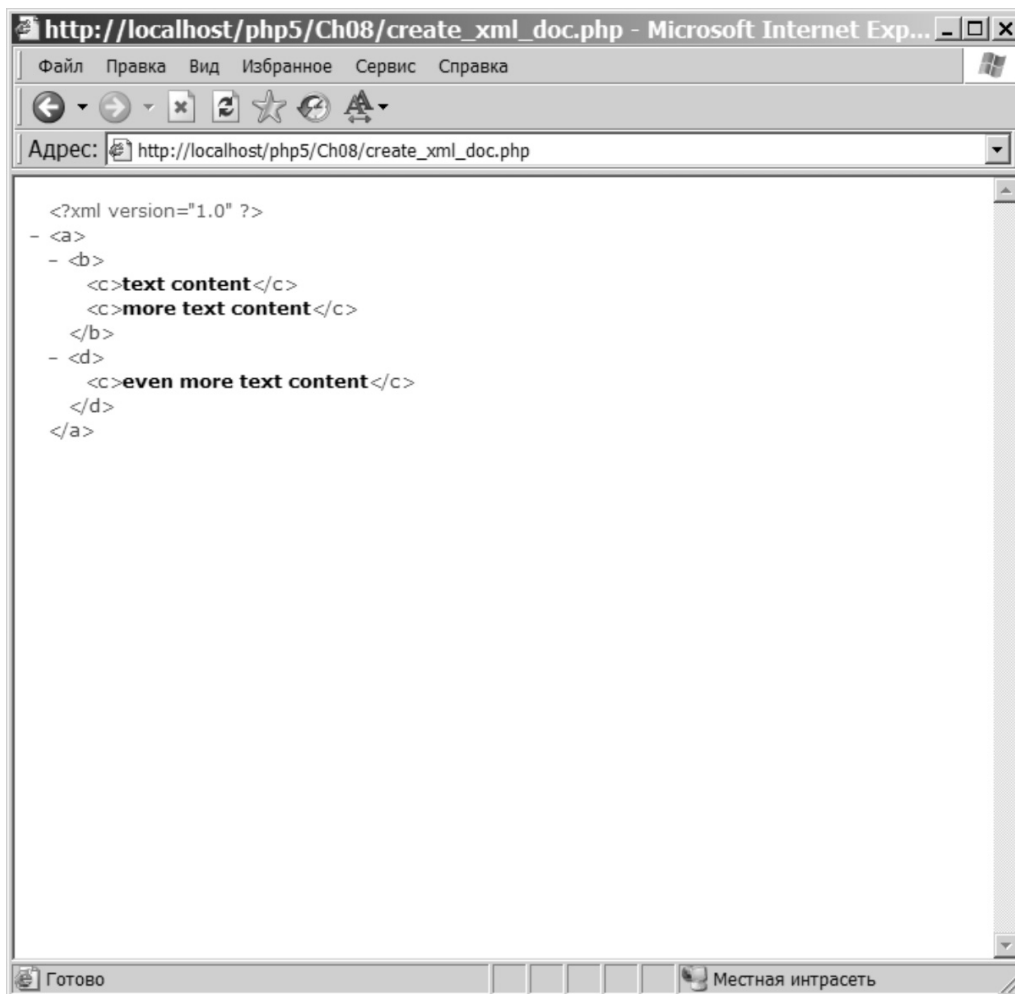


Рис. 8.4.

Использование функции `simplexml_load_file()`

Если данные уже записаны в XML-файл, то с помощью функции `simplexml_load_file()` содержимое этого файла можно преобразовать в `simpleXML`-объект. Как бы ни был создан `simpleXML`-объект, им можно манипулировать с помощью других `simpleXML`-функций.

В последующем примере для простоты используется внешний файл, содержащий XML-строки. Данные строки считываются с помощью функции `simplexml_load_file()`. Например, чтобы прочитать XML-файл, такой как был создан в начале данной главы, можно использовать функцию `simplexml_load_file()`.

Ниже приводится тот же XML-файл (его следует сохранить как `php_programs.xml`):

```
<?xml version="1.0" ?>
<php_programs>
  <program name="cart">
    <price>100</price>
  </program>
  <program name="survey">
    <price>500</price>
  </program>
</php_programs>
```

Чтобы прочитать имена и значения элементов и атрибутов в данном XML-документе, используется `simpleXML`-функция для загрузки файла — `simplexml_load_file()`. Теперь следует создать PHP-документ (и назвать его, например, `simplexml_01.php`), а затем присвоить переменной результат выполнения функции `simplexml_load_file()`.

```
<?php
$php_programs = simplexml_load_file('php_programs.xml');
```

Данная переменная содержит объект, который можно использовать почти так же, как обычный массив. Чтобы из этой переменной извлечь имена и значения, используется цикл `foreach`, который возвращает ключи и значения точно так же, как и в случае обычного массива:

```
foreach ($php_programs->program as $program_key => $program_val) {
echo "Корневой элемент <B>php_programs</B> содержит элемент с именем
<B>$program_key</B><BR>";
```

Однако вывести содержимое переменной `$program_val` непосредственно на экран не удастся; в переменной содержится подобный массиву объект, и поэтому снова придется использовать цикл `foreach` как для дочерних элементов, так и для их атрибутов:

```
foreach($program_val->children() as $child_of_program_key => $child_of_program_val)
{
  if ($child_of_program_key == "price") {
    foreach($program_val->attributes() as $att => $val) {
      if ($att == "name") {
        foreach($program_val->price as $the_price) {
          echo "Элемент <B>$program_key</B> имеет атрибут с именем <B>$att</B>
и значением <B>$val</B>.<BR>";
          echo "Элемент <B>$program_key</B> имеет дочерний элемент с именем
<B>$child_of_program_key</B> и значением <B>$the_price</B>.<BR>";
          echo "Следовательно, можно сказать, что ключ <B>$child_of_program_key</B>
<B>$val</B> <B>$program_key</B> равен
<B>\$the_price</B>.<BR><BR>";
        }
      }
    }
  }
}
```

Результат работы данного сценария представлен на рис. 8.5.

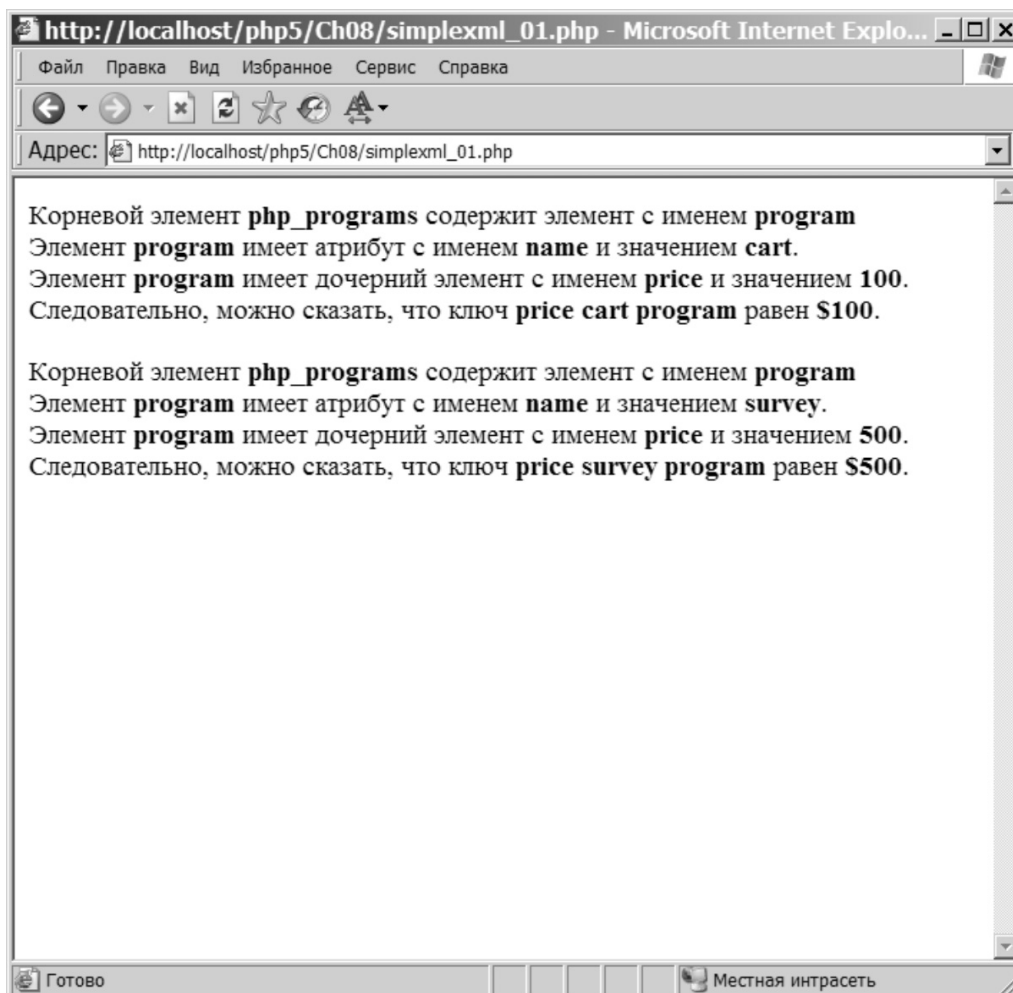


Рис. 8.5.

Изменение значений с помощью simpleXML

Можно не только считывать части XML-документа в simpleXML-объект. Кроме этого, можно изменять данные в документе, загруженном в память. Все что требуется сделать, это правильно обратиться к данным, которые нужно изменить, а затем записать вместо них необходимые значения.

Практика Изменение значения узла

Создайте новый файл, сохраните его как `simplexml_change_value.php` и введите в него следующий код:

```
<?php
$xmlstr = <<<XML
```

```
<php_programs>
  <program name="cart">
    <price>100</price>
  </program>
  <program name="survey">
    <price>500</price>
  </program>
</php_programs>
XML;

$first_xml_string = simplexml_load_string($xmlstr);
$first_xml_string->program[0]->price = '250';

echo "<pre>";
var_dump($first_xml_string);
echo "</pre>";
?>
```

Сохраните файл, а затем вызовите его в браузере. Результат работы сценария показан на рис. 8.6.

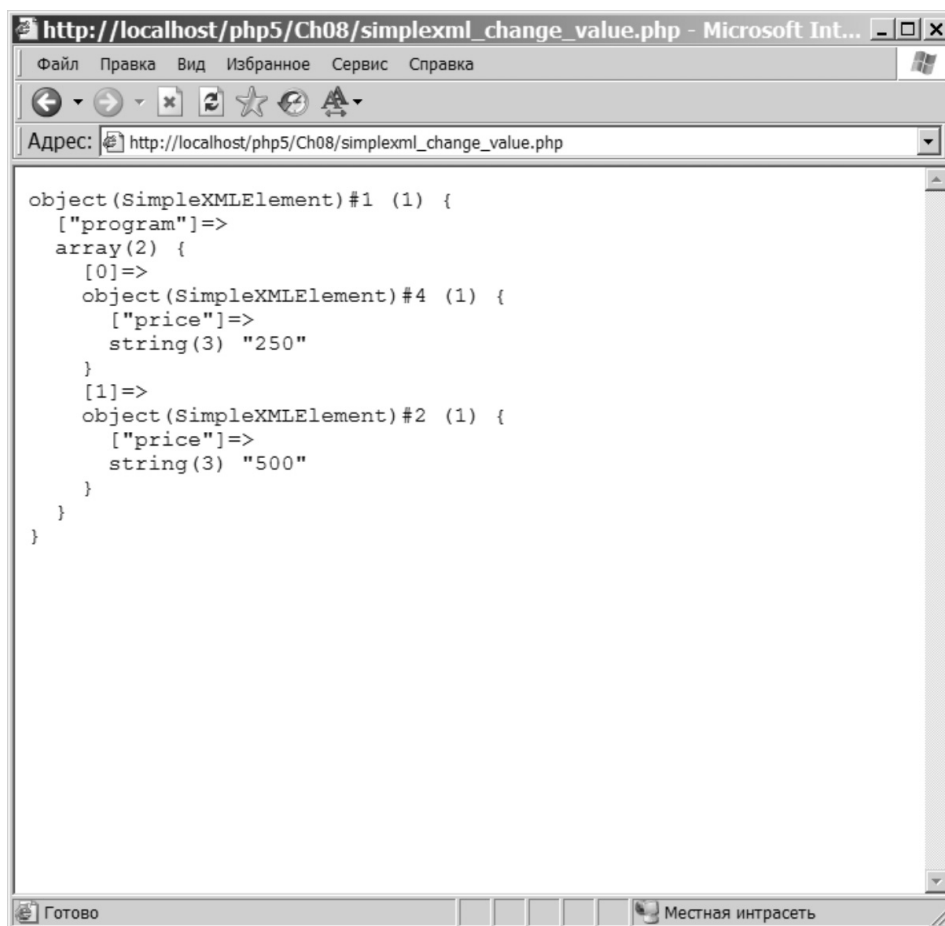


Рис. 8.6.

Как это работает

После того как XML-строка считана в переменную `$xmlstr`, обращаться к частям документа можно почти так же, как если бы документ был массивом, т.е. можно использовать имена и индексы для каждого уровня. Очевидно, что `program` и `price` — это элементы и что первая цена должна быть равна 100. Однако во второй строке приведенного здесь кода значение изменено на 250, а содержимое элементов распечатывается на экране с помощью функции `var_dump` (`var_dump` — PHP-функция, которая отображает структуру и данные PHP-переменных независимо от их типа). Для сохранения форматирования распечатываемых на экране данных используется HTML-тег `<PRE>`.

```
$first_xml_string = simplexml_load_string($xmlstr);
$first_xml_string->program[0]->price = '250';
echo "<PRE>";
var_dump($first_xml_string);
echo "</PRE>";
```

Практика Импорт DOM с помощью simpleXML

Используя расширение `simpleXML`, можно получать и создавать DOM-объекты. Создайте файл `simplexml_dom.php` со следующим кодом:

```
<?php
$xmlstr = <<<XML
<php_programs>
  <program name="cart">
    <price>100</price>
  </program>
  <program name="survey">
    <price>500</price>
  </program>
</php_programs>
XML;

$dom = new domDocument;
$dom->loadXML($xmlstr);
$s_dom = simplexml_import_dom($dom);

echo "Цена первой программы: <B>$" . $s_dom->program[0]->price . "</B>";
?>
```

Сохраните созданный файл и откройте его в браузере. Результат работы сценария показан на рис. 8.7.

Как это работает

Сначала с помощью ключевого слова `new` из класса `domDocument` создается новый XML-документ (более подробно создание объекта с помощью ключевого слова `new` рассматривается в главе 12, “Введение в объектно-ориентированное программирование”). Затем с помощью функции `loadXML` этого объекта XML-строка передается в `DOMDocument`-объект. Наконец, на экран выводится содержимое элемента `price`.

```
$dom = new domDocument;
$dom->loadXML($xmlstr);
$s_dom = simplexml_import_dom($dom);

echo "Цена первой программы: <B>$" . $s_dom->program[0]->price . "</B>";
```

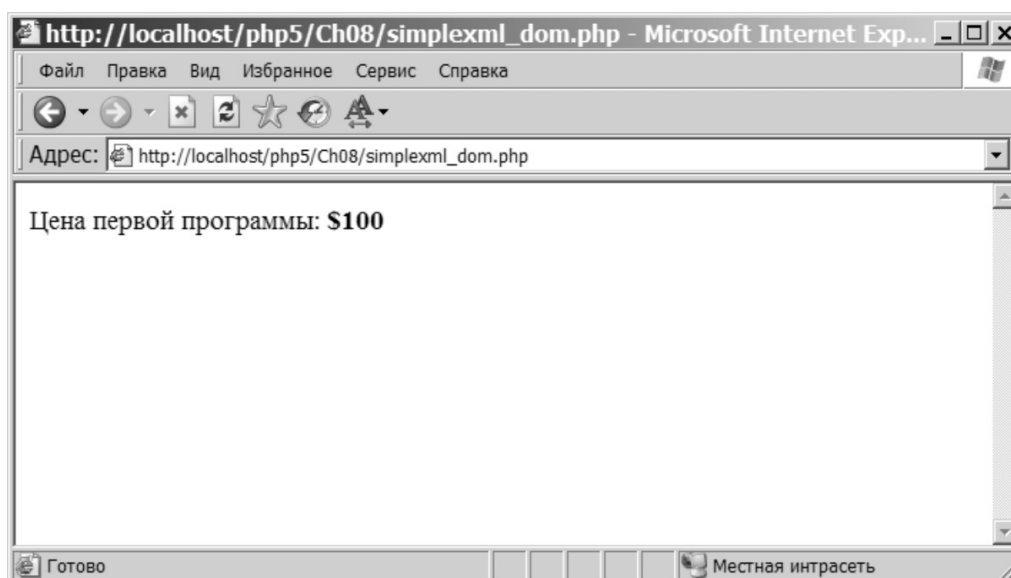


Рис. 8.7.

Резюме

В данной главе рассматривались основы технологии XML, включая правила формирования XML-документа, а также определение типа документа (DTD), написанное в формате EBNF, и обращение к DTD для проверки корректности XML-документа. В главе обсуждалась поддержка пространств имен в XML и ссылки на несколько XML-схем из одного XML-документа.

В главе также рассматривалась документная объектная модель (DOM) — иерархическая модель структуры XML-документов, и некоторые PHP-функции, связанные с DOM. Также здесь были описаны некоторые функции PHP4 для использования XML-анализаторов, а также с новое расширение simpleXML, которое появилось в PHP5.

Примеры сценариев помогают понять процесс создания и чтения XML-документов, работы с их содержимым с помощью simpleXML- и DOM-функций. Несмотря на то, что в книге пока не рассматривались все объектно-ориентированные функции расширений simpleXML и DOM, читатель уже, несомненно, уяснил способы эффективного создания и обработки XML-документов и их компонентов.

Упражнение

Создайте приложение, которое считывало бы XML-файл с несколькими элементами и атрибутами в simpleXML-объект, изменяло бы значения некоторых или всех элементов или атрибутов файла, а затем записывало бы модифицированный документ обратно в файл.

9

Введение в базы данных и SQL

В двух последних главах было показано, как PHP-сценарии могут использовать внешние файлы для хранения и получения данных. Хотя непосредственный доступ к файлам во многих обстоятельствах позволяет решать серьезные задачи, по мере увеличения решений для хранения данных такой доступ становится все более ограниченным — что видим, то и получаем, ни больше ни меньше. В конце концов, на плечи разработчика и его сценария ложится реализация выборки и сортировки данных, а также других операций с данными, и чем больше данных приходится обрабатывать, тем дольше выполняется сценарий. Это не такая уж большая проблема, если иметь дело с несколькими килобайтами данных, но что произойдет, если эти килобайты превратятся в мегабайты (или даже гигабайты)? Разработчик, намеревающийся создать успешный, популярный, управляемый данными Web-сайт, действительно должен учитывать такую возможность.

Подумать только: если бы в крупном коммерческом сайте, таком как Amazon, все данные извлекались непосредственно из простых текстовых файлов, то поиск какой-либо одной книги, не говоря уже о размещении заказа, занял бы несколько месяцев.

Базы данных специально предназначены для решения описанной проблемы. Учитывая их собственные специальные возможности по организации и безупречному хранению данных, базы данных можно представить себе как огромные библиотеки мира IT. Нет необходимости часами просматривать полки пыльных томов, одно только слово, неясное описание — и на столе немедленно появляется нужная книга.

Эта глава — первая из трех глав, в которых рассматриваются базы данных и возможности их применения для создания мощных, эффективных (и потенциально очень больших) PHP-приложений. Эти главы познакомят читателя с преимуществами хранения информации с помощью баз данных вместо файлов, а также с популярными базами данных (и их отличиями), с которыми сталкиваются разработчики PHP-приложений.

В главе рассматриваются реляционные базы данных, а также связанные с ними понятия, такие как нормализация и индексирование. Читатель узнает, как устанавли-

вать MySQL — свободно доступную и широко используемую систему управления базами данных, познакомится с некоторыми PHP-функциями для подключения к MySQL, а также для получения и модификации информации в базах данных.

В данной главе представлено много теоретических сведений, поэтому она может показаться более сложной по сравнению с остальными главами книги. Практика разработчика предполагает использование баз данных и их запуск на тестовой машине, поэтому в дальнейшем будет гораздо проще понять излагаемый в данной книге материал. По этой причине в течение нескольких следующих разделов рассмотрение самого PHP отодвигается на задний план. Это позволит подготовиться к двум последующим главам. Читатели, которые уже знакомы с базами данных и их работой, могут пропустить эту главу.

Хранение данных

В начале работы над управляемым данными приложением разработчик вынужден определиться с тем, какое хранилище данных будет использоваться в этом приложении. Иначе говоря, как и где приложение будет сохранять данные. Правильный ответ на этот вопрос всегда зависит от требований, предъявляемых к разрабатываемому приложению. В самом простом случае следует задать себе следующие вопросы:

- ☐ Велико ли количество используемых приложением данных?
- ☐ Высока ли частота доступа приложения к данным?
- ☐ Высока ли частота модификаций данных?
- ☐ Много ли пользователей могут одновременно попытаться получить данные?
- ☐ Увеличивается ли количество данных со временем?
- ☐ Велики ли будут потери в случае искажения, похищения или неумышленного уничтожения данных?

Если ответ на любой из этих вопросов положительный, то, вероятно, хранения данных в простых текстовых файлах придется избегать.

Это не означает, что текстовые файлы бесполезны — например, когда посетители пытаются войти в защищенную паролем область, Web-сервер Apache (если не задана иная конфигурация) аутентифицирует их, сверяя введенные ими данные с текстовым файлом, содержащим полный список пользовательских идентификаторов и паролей. Невелика проблема, если требуется обслуживать небольшую группу пользователей, но если понадобится одновременно проверять сотни пользовательских идентификаторов, то системе придется сканировать текстовый файл строку за строкой, пока не будет найдено совпадение. Пользователю, который записан в таком файле последним, придется очень долго ждать, пока Apache найдет его идентификационные данные и разрешит вход в защищенную область.

Часто наиболее эффективной альтернативой этому способу является использование для хранения, получения и модификации информации базы данных или, точнее, системы управления базами данных (СУБД). Хорошая система управления базами данных отлично справляется с ролью эффективного посредника между пользователем и данными. Она организывает, каталогизирует, резервирует и выполняет целый ряд других операций, которые направлены на то, чтобы ускорить и облегчить работу с данными.

Как база данных сохраняет информацию? Это в некоторой степени зависит от используемой системы управления базами данных, хотя в конечном итоге данные хранятся в виде потоков битов и байтов во множестве файлов — именно файлов. В действительности избавиться от файлов не удастся. Однако хорошие базы данных обладают средствами, которые позволяют использовать эти файлы как можно более эффективно.

Базы данных

Что же означает понятие *база данных*? Строго говоря, база данных представляет собой эффективно организованную совокупность данных, точно так же как библиотека представляет собой не более чем упорядоченную совокупность книг. Однако когда речь заходит о библиотеке, мы чаще всего подразумеваем учреждение в целом — не только книги, но и персонал, практику работы, а также здание. То же можно сказать и о базах данных, которая представляет собой совокупность: систему управления базами данных *и* сами данные.

Это не удивительно, потому что в обычных условиях контакт с “сырыми” данными осуществляется только через СУБД или, говоря менее формальным языком, через базу данных. В большинстве случаев пользователь не видит “связи” этих понятий, поэтому реальных практических различий не существует. Далее в главе обсуждаются темы установки базы данных, подключения и обращения к ней и т.д., при этом понятие “база данных” используется в его наиболее употребительном, обобщающем смысле.

Архитектуры баз данных

Прежде всего необходимо определиться с используемой базой данных и типом ее архитектуры. Существует два основных варианта: архитектура встроенных баз данных и клиент/серверная архитектура. Ниже кратко рассматриваются оба типа.

Встроенные базы данных

Встроенная база данных работает и хранит данные на той же машине, что и программа, которая использует эти данные (в данном случае PHP). Такая база данных недоступна в сети и в определенный момент времени к ней может подключиться только одна программа. Кроме того, база данных не может совместно использоваться несколькими машинами, потому что каждая из них в конечном итоге сохраняет и использует свою собственную отдельную версию данных. Продолжая аналогию с библиотеками, можно сказать, что такая база данных подобна персональной библиотеке, к которой пользователь имеет исключительный доступ. Часто в мелких приложениях использовать встроенные базы данных более предпочтительно. В более крупных системах чашу весов перевешивают большие реляционные системы управления базами данных (СУРБД) и поэтому большинство коммерческих предприятий используют клиент/серверные архитектуры баз данных.

В число давно существующих, популярных примеров встроенных баз данных включаются dBase и DBM. PHP предоставляет связь с обеими системами. Недавно в этот перечень добавилась база данных SQLite, которая доступна не только как расширение для PHP, а фактически встроена в дистрибутив PHP5. SQLite стоит изучить хотя бы только по этой причине. А впечатляющая статистика по производительности определенно подтверждает статус этой системы как перспективной технологии баз данных в PHP. Более подробная информация об SQLite представлена в приложении В, “Использование SQLite”.

Клиент/серверные базы данных

Клиент/серверные базы данных предназначены для использования через сеть и позволяют множеству пользователей (которые могут находиться в разных уголках планеты) работать одновременно с одними и теми же данными. Сама база данных (или СУБД) работает как сервер в режиме, очень похожем на режим работы Web-сервера, который обсуждался в первых главах книги. В принципе, такая база данных способна обрабатывать запросы, поступающие через сетевое соединение из любой точки мира и от любой подходящей клиентской программы. Вместе с тем, нет причин, препятствующих запуску сервера и клиентской программы на одной машине.

Клиент/серверные базы данных близки к метафоре публичной библиотеки. Публичная библиотека открыта для всех, кто имеет членский билет (т.е. зарегистрирован в библиотеке), а множество работающих в ней библиотекарей могут одновременно обслуживать запросы от большого количества посетителей. Посетители могут прийти лично или позвонить в библиотеку по телефону.

Фактическая работа по поиску и возвращению книг на полки может быть делегирована помощнику, и посетители могут об этом не беспокоиться. Каждый библиотекарь, принимающий посетителей, может обслуживать несколько посетителей одновременно, но если он хорошо выполняет свою работу, то, вероятно, уделяет каждому читателю достаточно времени, и все посетители довольны таким обслуживанием. Аналогично, когда несколько пользователей одновременно получают доступ к клиент/серверной базе данных, сервер баз данных периодически переключается между ними, создавая у каждого иллюзию одновременного обслуживания всех пользователей.

Описанный выше вид баз данных наиболее распространен в крупных компаниях, где большое количество данных должно использоваться совместно многими сотрудниками, где доступ может осуществляться из любой точки сети и где наличие централизованного хранилища данных значительно упрощает такие важные задачи, как администрирование и резервное копирование информации. Любые приложения, которые нуждаются в доступе к базам данных, используют специализированные, легковесные клиентские программы для сообщения с сервером.

Установка и администрирование некоторых СУРБД (систем управления реляционными базами данных) часто оказывается очень дорогим и сложным мероприятием. Общеизвестными представителями этого типа баз данных являются Oracle, DB/2 (производства IBM) и SQL Server (Microsoft) — массивные, многофункциональные системы, способные хранить и обрабатывать практически любые данные, которые могут понадобиться на современном предприятии. Обратная сторона этой медали — тяжеловесность и дороговизна таких систем, к тому же они могут содержать больше функций, чем от них требуется.

К счастью, существуют альтернативы, такие как системы PostgreSQL и MySQL, каждая из которых представляет собой клиент/серверную систему баз данных с открытым исходным кодом и вот уже много лет является весьма популярной среди РНР-разработчиков. Эти системы быстры, стабильны и легко удовлетворяют потребности большинства мелких и средних проектов, и, что не менее важно, они бесплатны.

Выбор базы данных

В принципе, в РНР-приложениях можно использовать любую из этих систем. Можно даже привязать одно приложение к нескольким различным системам баз данных. Чтобы сохранить разумные размеры этой и двух последующих глав, авторы решили сосредоточиться на одной СУБД — MySQL.

По сравнению с другими данная система предоставляет ряд преимуществ:

- ❑ это одна из самых популярных систем управления базами данных, используемых в настоящее время в Web-среде;
- ❑ она свободно доступна для загрузки из Internet и установки на практически любой машине;
- ❑ ее легко установить на многих операционных системах (включая Windows и Unix);
- ❑ эта система доступна как сравнительно недорогая функция во многих пакетах Web-хостинговых услуг;
- ❑ она проста в использовании и включает в себя несколько удобных инструментов администрирования;
- ❑ это быстрая, мощная клиент/серверная система, которая справляется с очень крупными, сложными базами данных и хорошо проявляет себя, когда дело касается крупных проектов.

Если последний критерий не является определяющим для какого-либо разрабатываемого приложения (и если нести дополнительные затраты, связанные с функциональностью баз данных на Web-сервере, нежелательно), то можно с успехом использовать встраиваемую базу данных, такую как SQLite.

MySQL — свободно доступная СУБД, разработчики которой лишь недавно присоединились к сообществу открытого исходного кода, выпустив MySQL под лицензией GPL (GNU Public License — общедоступная лицензия GNU). Даже до того как она стала бесплатной, для некоммерческого ее использования или использования не на Windows-платформах (Windows-версия MySQL была условно бесплатной) не требовалось покупать лицензию. Уже одно только то, что за использование MySQL теперь не нужно платить деньги, делает эту систему серьезным кандидатом для разработки приложений с базами данных. Если лицензия GPL по какой-либо причине не устраивает разработчика или MySQL требуется внедрить в коммерческое приложение, то все-таки придется приобрести коммерческую лицензионную версию у разработчиков продукта на сайте www.mysql.com.

Здесь и в последующих главах вводится много новых понятий, которые выходят за рамки одной определенной системы управления базами данных, поэтому для начала остановимся на MySQL.

Установка MySQL

В качестве примеров баз данных в книге используются базы данных MySQL, поэтому читателю следует научиться устанавливать и запускать эту систему.

К моменту написания книги была доступна стабильная версия MySQL 4.0, а файлы (как исходные коды так и скомпилированные бинарные файлы для различных платформ) были доступны на сайте MySQL по адресу www.mysql.com/downloads/mysql-4.0.html.

Подробности инсталляции (например, последовательность установки и местоположение ключевых файлов) могут значительно отличаться в зависимости от таких факторов, как используемая операционная система и тип загруженного пакета MySQL (исходный код или скомпилированные бинарные файлы), поэтому рассмотрим несколько вариантов процесса установки.

Установка на Windows

Бинарный установочный пакет для Windows поставляется в виде zip-архива (например, `mysql-4.0.16-win.zip`), содержащего множество файлов, включая исполняемый файл `setup.exe`. Для установки MySQL необходимо запустить этот файл, после чего мастер инсталляции предложит несколько параметров установки. Эти параметры определяют, какие файлы будут установлены и где они будут храниться впоследствии. Конфигурация по умолчанию (установка в каталог `C:\MySQL`) должна удовлетворить многих пользователей.

Все основные программы MySQL хранятся в каталоге `bin` инсталляционного пакета. Если при установке были сохранены значения по умолчанию, то путь к файлам этих программ будет `C:\mysql\bin`. Большинство данных программ (например, `mysql.exe`, `mysqladmin.exe` и `mysqld.exe`) являются инструментами командной строки, поэтому их не следует запускать посредством двойного щелчка мышью на пиктограмме. Их можно будет увидеть в действии позднее.

Программа с информативным названием `winmysqladmin.exe`, которая поставляется в стандартной Windows-версии MySQL, предоставляет удобный интерфейс для управления MySQL-сервером. Эта программа имеет графический пользовательский интерфейс, поэтому ее можно запускать как любую другую программу двойным щелчком мыши.

При первом запуске программа потребует ввести имя пользователя и пароль и создаст учетную запись пользователя для работы со средством администрирования баз данных. После этого запускается программа MySQL-сервера, которая в случае правильной установки должна запуститься впервые.

После запуска сервера программа `winmysqladmin` сворачивается в пиктограмму с изображением светофора в панели задач (обычно в правом нижнем углу экрана). Зеленый огонек показывает, что сервер запущен, а красный — что сервер остановлен.

Щелчок правой кнопкой мыши на пиктограмме вызывает контекстное меню, позволяющее запустить или остановить сервер, а также вывести окно `winmysqladmin` (рис. 9.1), в котором можно просматривать и редактировать различные параметры сервера.

Установка MySQL на Linux

Пользователи Linux могут установить MySQL из бинарного пакета или из исходного кода. В случае использования RPM-пакета необходимо убедиться, что на машине имеются инсталляционные пакеты сервера, клиента, подключаемые файлы и библиотеки, а также клиентские общие библиотеки для используемой платформы. Должны присутствовать следующие четыре файла (хотя точные имена могут отличаться в зависимости от версии MySQL и используемой операционной системы):

- ☐ `MySQL-4.0.16.i386.rpm`
- ☐ `MySQL-client-4.0.16.i386.rpm`
- ☐ `MySQL-devel-4.0.16.i386.rpm`
- ☐ `MySQL-shared-4.0.16.i386.rpm`

В случае использования исходного кода понадобится только один архив, `mysql-4.0.18.tar.gz`.

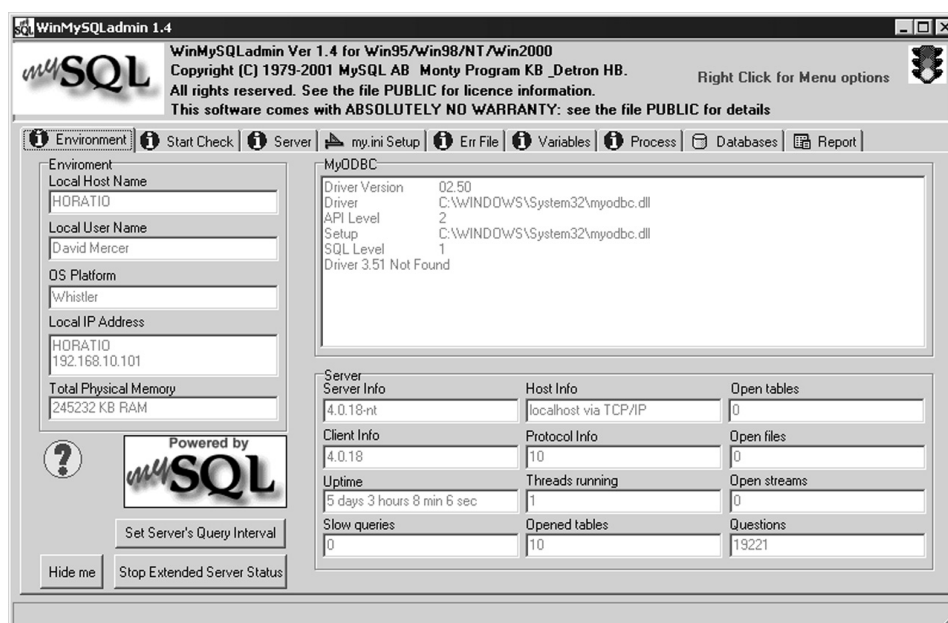


Рис. 9.1.

Установка MySQL из RPM-пакетов

RPM-пакеты устанавливаются с помощью следующей команды:

```
> rpm -Uvh filename.rpm
```

Пакеты необходимо устанавливать в том же порядке, в котором они перечислены в предыдущем разделе.

При установке первого пакета, содержащего MySQL-сервер, на экране появится следующий текст:

```
PLEASE REMEMBER TO SET A PASSWORD FOR THE MySQL root USER !
This is done with:
/usr/bin/mysqladmin -u root password 'new-password'
See the manual for more instructions.
```

Please report any problems with the /usr/bin/mysqlbug script!

```
The latest information about MySQL is available on the web at http://www.
mysql.com
Support MySQL by buying support/licenses at http://www.tcx.se/license.htm.
Starting mysqld daemon with databases from /var/lib/mysql
```

```
ПОЖАЛУЙСТА, НЕ ЗАБУДЬТЕ УСТАНОВИТЬ ПАРОЛЬ ДЛЯ MySQL-ПОЛЬЗОВАТЕЛЯ root!
Это делается с помощью команды:
/usr/bin/mysqladmin -u root password 'новый-пароль'
Дальнейшие инструкции см. в документации.
```

```
Пожалуйста, создавайте отчеты о проблемах с помощью сценария
/usr/bin/mysqlbug
```

```
Последняя информация о MySQL доступна на Web-сайте
http://www.mysql.com
Чтобы получить техническую поддержку, необходимо приобрести лицензию
```

<http://www.tcx.se/license.htm>.

Запускайте `mysqld`-демон с базами данных из каталога `/var/lib/mysql`

Программа `mysqladmin` является одним из клиентских инструментов, поэтому необходимо повременить с установкой пароля до инсталляции клиентского пакета, которая описана в этой главе далее. RPM немедленно запускает программу MySQL-сервера `mysqld` (MySQL-демон). Кроме того, создается сценарий запуска и останова `/etc/rc.d/init.d/mysql`, который гарантирует, что MySQL-сервер будет запускаться каждый раз при загрузке и останавливаться при выключении компьютера. Следующий сценарий можно использовать для запуска и останова `mysqld`:

```
> /etc/init.d/mysql start
```

```
> /etc/init.d/mysql stop
```

Теперь следует установить пакеты `MySQL-client`, `MySQL-devel` и `MySQL-shared`, после чего можно переходить к разделу “Конфигурирование MySQL” данной главы.

Установка MySQL из исходного кода

Устанавливать MySQL из исходного кода довольно просто. Для этого необходимо загрузить архив с Web-сайта MySQL и начать процесс инсталляции:

```
> tar -zxvf mysql-4.0.18.tar.gz
> cd mysql-4.0.18
> ./configure --prefix=/usr
> make
```

Если программа `make` аварийно завершается, то, скорее всего, это происходит из-за недостатка памяти. Такое возможно даже на довольно мощных машинах. В этой ситуации можно сделать следующее:

```
> rm -f config.cache
> make clean
> ./configure --prefix=/usr --with-low-memory
> make
```

Если и это не помогает, обратитесь к документу www.mysql.com/doc/en/Compilation_problems.html.

Предположим, что сценарии `configure` и `make` выполнились без сбоев, тогда для завершения инсталляции достаточно ввести следующие команды:

```
> make install
> mysql_install_db
```

В дальнейшем для запуска и останова MySQL-сервера, `mysqld` понадобятся специальные сценарии. Ниже приводится пример сценария для запуска:

```
#!/bin/bash
/usr/bin/mysqld_safe &
```

и соответственно для останова:

```
#!/bin/bash
kill 'cat /usr/var/$HOSTNAME.pid'
```

Эти сценарии имеются в пакете исходных кодов, который можно загрузить с Web-страницы данной книги (www.wrox.com). Тем, кто предпочитает писать сценарии запуска и останова самостоятельно, рекомендуется создать в текстовом редакторе соответствующие файлы, а затем с помощью команды `chmod` присвоить им права на выполнение. Например, если сценарий был сохранен как `startmysqld`, то следует ввести такую команду:

```
> chmod ugo+rx startmysqld
```

Конфигурирование MySQL

Установив и запустив на локальной тестовой машине сервер баз данных MySQL, следует уделить время конфигурированию системы.

Для конфигурирования MySQL, как и в случае большинства сетевых систем (включая другие клиент-серверные базы данных, операционной системы и т.д.), необходимо войти в систему с определенной учетной записью. Такая мера предосторожности ограничивает доступ к данным путем назначения специфических прав доступа для каждой учетной записи. Например, один пользователь может иметь право только на просмотр существующих данных, тогда как другому разрешено добавлять новые данные и даже, возможно, изменять привилегии других пользователей.

Главному пользователю в системе, который автоматически получает права на просмотр и изменение *всех* данных и настроек, обычно назначается имя *root*. Во время установки MySQL учетная запись *root* создается автоматически, однако пароль для нее не устанавливается. Сразу после установки система доступна для использования (в том числе злонамеренного) любым пользователем, имеющим MySQL-клиент и сетевое подключение к данному серверу.

Чтобы обезопасить систему, необходимо из командной строки настроить учетную запись *root*, выполнив описанные ниже действия.

1. Вызвать окно командной строки и перейти в каталог, содержащий программные файлы MySQL (обычно `C:\mysql\bin\` в Windows или `/usr/bin/` в Linux).
2. Ввести следующую команду, подставив желаемый пароль вместо слова *elephant*:

```
> mysqladmin -uroot password elephant
```

После этого можно попытаться вводить в командной строке некоторые другие команды. Например, команда `mysqlshow` выводит перечень баз данных, доступных на сервере в текущий момент времени:

```
> mysqlshow
+-----+
| Databases |
+-----+
| mysql    |
| test     |
+-----+
```

Сразу после установки MySQL эта команда показывает, что сервер уже создал две собственных базы данных. Первая *mysql* — база данных, в которой хранится вся информация, необходимая для аутентификации пользователей. Вторая — пустая тестовая база данных. Команда `mysqlshow` также может показать содержимое той или иной базы данных при условии ввода правильного пароля:

```
> mysqlshow -uroot -p mysql
Enter password: *****
Database: mysql
+-----+
| Tables |
+-----+
| columns_priv |
| db           |
| func         |
| host         |
| tables_priv  |
| user         |
+-----+
```

В этом случае команда показывает содержимое базы данных `mysql` — все системные данные организованы в виде шести именованных таблиц. В примере наглядно демонстрируется модель хранения данных, которая используется в `MySQL` для организации данных. Прежде чем двигаться далее, вкратце рассмотрим некоторые теоретические вопросы и попробуем разобраться, как такие группы таблиц позволяют эффективно хранить данные.

Реляционные базы данных

Рассмотрим еще одну ключевую особенность баз данных — способ организации информации, с которой работают пользователи. Используя базу данных для получения модификации или добавления новых данных, необходимо знать, как запрашивать данные. Более того, когда приходится самостоятельно разрабатывать базу данных, необходимо сообщить системе, какие данные и где будут в ней находиться, а также как они будут логически связаны друг с другом.

Большинство баз данных используют так называемую реляционную модель данных, поэтому они называются *реляционными базами данных* (или системами управления реляционными базами данных — СУРБД). В таких базах данные организованы в таблицах, каждая из которых разделяется на строки и столбцы.

В терминах баз данных каждая строка в таблице (кроме заголовка) представляет *запись* данных: набор внутренне связанных блоков данных. Аналогично, каждый столбец представляет *поле*: определенный тип данных, который имеет одну и ту же значимость для каждой записи в таблице.

С практической точки зрения понятие “строка” является синонимом записи, а “столбец” — синонимом поля. Это необходимо учитывать при визуализации таблиц.

Предположим, что менеджер команды по регби (вид спорта выбран случайно и никак не влияет на понимание работы баз данных) создает базу данных, чтобы отслеживать матчи, в которых участвовала его команда, и просит каждого игрока вводить в базу данных после каждого матча свои данные. После второго круга таблица менеджера выглядит так:

<i>Player_Number</i>	<i>Name</i>	<i>Phone_number</i>	<i>Date_Played</i>	<i>Nickname</i>
42	David	555-1234	03/03/04	Dodge
6	Nic	555-3456	03/03/04	Obi-d
2	David	555-6543	03/03/04	Witblitz
14	Mark	555-1213	03/03/04	Greeny
2	David	555-6543	02/25/04	Witblitz
25	Pads	555-9101	02/25/04	Pads
6	Nic	555-3456	02/25/04	Obi-d
7	Nic	555-5678	02/25/04	Nicrot

Вскоре становится ясно, что данная таблица будет сильно разрастаться после каждого сыгранного сезона. Очевидно, что такая структура таблицы неэффективна, по-

тому что данные каждого игрока — номер, имя, телефонный номер и т.д. — вводятся после каждого матча с его участием.

Подобная избыточность в базах данных нежелательна. Например, предположим, что игрок под номером 6 постоянно пропускает голы, и члены его команды решили дать ему новое прозвище (которое здесь не упоминается). Чтобы обновить таблицу, необходимо модифицировать каждую из записей игрока (т.е. поменять в каждой записи значение поля Nickname).

В дополнение к этому, вводимые игроком после каждого матча данные потребляют ценное пространство на жестком диске. Избыточность крайне неэффективна, впустую расходуется время и пространство на жестком диске.

В начале 70-х годов прошлого века математик Э.Ф. Кодд (E.F. Codd) предложил уникальный и эффективный способ решения этой проблемы. Он зафиксировал набор правил, применение которых к данным гарантирует создание правильной структуры базы данных. На самом деле он предложил множество таких правил, здесь представлены некоторые из них. Правила (или требования) можно сгруппировать в так называемые *нормальные формы* (*normal forms*). Данные, соответствующие этим нормальным формам, в значительной степени обуславливают хорошую конструкцию реляционной базы данных. Рассмотрим теперь суть процесса нормализации.

Нормализация

Нормализация определяется как “процесс разбиения данных на несколько таблиц с целью минимизации количества повторения одних и тех же данных”. Нормальные формы представляют собой степени нормализации и регулируются изящным набором правил, которые описаны далее.

Первая нормальная форма (1NF)

- ☐ Создавать новую таблицу для каждого нового набора связанных данных.
- ☐ Исключить повторяющуюся информацию в каждой отдельной таблице.
- ☐ Уникально идентифицировать каждую запись с помощью первичного ключа.

Например, предыдущую таблицу можно представить в форме 1NF в виде двух таблиц. Первая таблица — информация об игроках:

<i>Player_Id</i>	<i>Name</i>	<i>Phone_number</i>	<i>Nickname</i>
42	David	555-1234	Dodge
6	Nic	555-3456	Obi-d
14	Mark	555-1213	Greeny
2	David	555-6543	Witblitz
25	Pads	555-9101	Pads
6	Nic	555-3456	Obi-d
7	Nic	555-5678	Nicrot

Вторая таблица — протокол матчей:

<i>Player_Id</i>	<i>Date_Played</i>
42	03/03/04
6	03/03/04
2	03/03/04
14	03/03/04
2	25/02/04
25	25/02/04
6	25/02/04
7	25/02/04

Обратите внимание, насколько естественно исходная таблица разделяется на две новые таблицы. Это связано с тем, что в исходной таблице хранились данные о двух различных *элементах*: игроках и матчах. Каждая новая таблица содержит данные только об одном элементе. Разделение элементов по таблицам представляет собой важную часть процесса нормализации.

Теперь можно перейти к следующему этапу приведения базы данных к первой нормальной форме: исключить повторения информации в каждой отдельной таблице. В таблице игроков имеется некоторая избыточная информация, которую можно устранить. В этом случае таблица примет следующий вид:

<i>Player_Id</i>	<i>Name</i>	<i>Phone_number</i>	<i>Nickname</i>
42	David	555-1234	Dodge
6	Nic	555-3456	Obi-d
2	David	555-6543	Witblitz
14	Mark	555-1213	Greeny
25	Pads	555-9101	Pads
7	Nic	555-5678	Nicrot

Теперь в таблице нет дублирующейся информации, таблица представляет только связанную информацию (информацию об игроках). Что можно сказать по поводу уникального идентификатора? Каждая запись должна иметь (по крайней мере, одно) уникальное поле, иначе говоря, поле, в котором нет повторяющихся значений. Ввиду исключительной природы данного поля каждое значение в нем уникально идентифицирует каждую запись таблицы. Поле, которое используется для идентификации записей, называется *первичным ключом* (*primary key*). В каждой таблице допускается только один первичный ключ.

Перемещение информации об игроках в одну таблицу, а протокола матчей в другую упрощает процесс модификации персональных данных игроков — необходимо изменить только одну запись в таблице игроков.

Можно было бы добавить уникальный ключ, который присваивал бы каждой записи уникальный номер, но для начала стоит внимательно изучить информацию в таблице игроков. В каждом виде спорта номер игрока должен быть уникальным, таким образом, известно, что все значения в поле `Player_Id` уникальны. Кроме этого значения в поле `Nickname` также уникальны, поскольку в определенный момент времени у каждого игрока есть только одно прозвище. Какое из двух этих полей выбрать? Поскольку прозвища могут изменяться в зависимости от “эффективности” игрока, поле `Player_Id` является наиболее подходящим вариантом для первичного ключа.

Что касается таблицы протокола матчей, то в ней поле `Player_Id` уже не является уникальным, потому что для каждого игрока в таблице имеется несколько записей с разными датами игры. Очевидно, что и поле даты тоже не может быть уникальным. В такой ситуации можно самостоятельно создать уникальное поле (как это сделать, будет показано далее). Теперь обе таблицы соответствуют первой нормальной форме. Добавление ID-поля имеет еще одно применение, особенно если требуется объединить информацию двух полей таблицы протокола матчей: его можно использовать, чтобы создать уникальный ключ, который можно было бы связать с какими-либо другими данными (например, с количеством добытых для команды очков по каждому игроку в каждом матче).

Вторая нормальная форма имеет две цели, которые применимы только к базам данных соответствующих 1NF.

Вторая нормальная форма (2NF)

- ☐ Создавать новые таблицы для каждой группы значений, относящихся к нескольким записям.
- ☐ Связывать новые таблицы с существующими по *внешнему ключу* (*foreign key*) (ключ, идентифицирующий записи в различных таблицах).

Обе таблицы содержат общее поле (`Player_Id`), значения в котором совпадают в двух таблицах, поэтому можно *связать* эти таблицы вместе. Фактически для этого внешний ключ *не требуется*, однако использование внешних ключей определенно имеет свои преимущества (более глубокое их обсуждение представлено на странице www.mysql.com/doc/en/ANSI_diff_Foreign_Keys.html). Это означает, что записи в одной таблице имеют четко определенные связи с записями в других таблицах. В настоящий момент база данных имеет всю информацию, необходимую для выдачи результатов по сложным запросам, охватывающим обе таблицы. Эту информацию невозможно было бы получить, изолированно просматривая таблицы. Базы данных с подобными таблицами можно по праву называть *реляционными*. Описанные здесь специфические отношения также называются отношениями “один ко многим”. Для каждой записи в таблице игроков может существовать множество записей в таблице протокола матчей. Способность создавать выразительные отношения “один ко многим” хорошо подтверждает то, что база данных соответствует требованиям второй нормальной формы.

Третья нормальная форма (3NF)

Цель третьей нормальной формы — устранить поля, которые не зависят от первичного ключа.

Это тот случай, когда реальная практика и изящество теории могут противоречить друг другу. В некоторой степени это противоречит утверждению о связи с главной таблицей — в любой момент времени в записи могут присутствовать повторяющиеся данные, например, почтовые индексы в списке адресов. Такие данные должны хра-

ниться в отдельной таблице и связываться с отдельными использующими их записями по новому ключу. Здесь важно не ошибиться; пуристы будут жестко критиковать вас за то, что это правило не соблюдается, но истина заключается в том, что проектирование базы данных часто является серией компромиссов; иногда приходится сравнивать идеализм конструкции и новую, но полезную концепцию — простоту. Требования третьей нормальной формы могут не удовлетворять потребностям приложения; и в каждой определенной ситуации следует принимать отдельное решение о том, следует ли эти требования выполнять.

Другие нормальные формы

Существует еще немало нормальных форм. Четвертая нормальная форма требует изолировать независимые множественные отношения. Это правило главным образом применимо к проблемам отношений “многие ко многим”, где несколько записей в одной таблице могут быть перекрестно связаны с записями в другой таблице, которые в свою очередь могут быть обратно связаны с несколькими записями первой таблицы. Такая задача может показаться весьма сложной, но это как раз тот случай, когда придерживаться стандарта необязательно. Здесь эта концепция упоминается потому, что подобная ситуация вполне вероятна. Поэтому полезно знать, что такие проблемы существуют.

Обращение к базам данных с помощью SQL

Итак, MySQL — реляционная база данных. По сути, теперь можно начинать применять только что описанные принципы к самой системе баз данных. Однако сначала необходимо более подробно рассмотреть язык, который используется для взаимодействия с реляционными базами данных. Этот язык называется SQL.

SQL (или Structured Query Language — язык структурированных запросов) — стандартный набор команд, который используется для обмена данными с системой управления реляционными базами данных на любой платформе. Любая задача, такая как создание баз данных или таблиц, а также сохранение, получение, удаление и обновление данных в базах, решается посредством SQL-операторов. Реализация SQL-функций отличается в СУРБД разных поставщиков, но поскольку базовые идеи идентичны, применение на одной платформе навыков, полученных на другой платформе, не намного сложнее, чем перенос компьютерной программы с одной платформы на другую с использованием одного языка (в частности PHP). В этом разделе рассматриваются некоторые базовые особенности SQL: типы данных, индексы, ключи и запросы.

Реализация некоторых SQL-функций в MySQL отличается от стандарта ANSI SQL.

Перечень этих отличий можно получить на странице

www.mysql.com/doc/en/Differences_from_ANSI.html.

Типы данных в SQL

При создании таблицы базы данных необходимо определить тип и размер каждого поля. Поля аналогичны PHP-переменным за исключением того, что в каждом поле могут храниться только данные определенного типа и размера. Следовательно, в отличие от PHP-переменных записать символьные данные в целочисленное поле невозможно. В MySQL поддерживаются три распространенных группы типов данных: численные, дата/время и символьные. Все эти типы данных рассматриваются в следующих таблицах.

<i>Численные типы данных</i>	<i>Описание</i>	<i>Диапазон/формат</i>
INT	Целые числа обычной величины	(от -231 до 231 -1) или (от 0 до 232 -1), если UNSIGNED
TINYINT	Очень малые целые числа	(от -27 до 27 -1) или (0 до 28 -1), если UNSIGNED
SMALLINT	Малые целые числа	(от -215 до 215 -1) или (от 0 до 28 -1), если UNSIGNED
MEDIUMINT	Средние целые числа	(от -223 до 223 -1) или (от 0 до 224 -1), если UNSIGNED
BIGINT	Большие целые числа	(от -263 до 263 -1) или (от 0 до 264 -1), если UNSIGNED
FLOAT	Числа с плавающей запятой с обычной точностью	Минимальное ненулевое $\pm 1.176 \times 10^{-38}$; максимальное ненулевое $\pm 3.403 \times 10^{38}$
DOUBLE/REAL	Числа с плавающей запятой с двойной точностью	Минимальное ненулевое $\pm 2.225 \times 10^{-308}$; максимальное ненулевое $\pm 1.798 \times 10^{308}$
DECIMAL	Числа с плавающей запятой, сохраняемые как строки	Максимальный диапазон такой же, как у DOUBLE

<i>Дата/время</i>	<i>Описание</i>	<i>Диапазон/формат</i>
DATE	Дата	Формат YYYY-MM-DD. Диапазон от 1000-01-01 до 9999-12-31
DATETIME	Дата и время	Формат YYYY-MM-DD hh:mm:ss. Диапазон от 1000-01-01 00:00:00 до 9999-12-31 23:59:59
TIMESTAMP	Временная метка	Формат YYYYMMDDhhmmss. Диапазон от 19700101000000 до 2037 года
TIME	Время	Формат hh:mm:ss. Диапазон от -838:59:59 до 838:59:59
YEAR	Год	Формат YYYY. Диапазон от 1900 до 2155

<i>Символьные типы данных</i>	<i>Описание</i>	<i>Диапазон/формат</i>
CHAR	Строки фиксированной длины	0-255 символов
VARCHAR	Строки переменной длины	0-255 символов
BLOB	Большой двоичный объект (BLOB)	Двоичные данные 0-65535 байтов в длину
TINYBLOB	Небольшое BLOB-значение	Двоичные данные 0-255 байтов в длину
MEDIUMBLOB	Среднее BLOB-значение	Двоичные данные 0-16777215 байтов в длину
LONGBLOB	Большое BLOB-значение	Двоичные данные 0-4294967295 байтов в длину
TEXT	Текстовое поле обычного размера	0-65535 байтов
TINYTEXT	Малое текстовое поле	0-255 байтов

<i>Символьные типы данных</i>	<i>Описание</i>	<i>Диапазон/формат</i>
MEDIUMTEXT	Текстовое поле среднего размера	0–16777215 байтов
LONGTEXT	Большое текстовое поле	0–4294967295 байтов
ENUM	Перечисление	Присваивается одно значение из списка
SET	Установленные значения	Присваивается нуль или несколько значений из списка

Различие между типами полей CHAR и VARCHAR заключается в том, что в первом случае в поле хранится значение фиксированной длины независимо от фактической длины значения, а во втором случае в поле содержится в точности столько байтов, сколько необходимо для хранения заданного значения. Предположим, например, что в поля, определенные как `char_field CHAR(10)` и `varchar_field VARCHAR(10)`, записана строка `dodge`. Значения, сохраненные в этих полях, будут немного отличаться друг от друга:

```
char_field: 'dodge'      '// для заполнения поля справа от заданной
                        // строки добавляется пять пробелов
varchar_field: 'dodge'   '// пробелов нет
```

Из этого следует, что объявление поля с типом VARCHAR позволяет сэкономить некоторое дисковое пространство. Однако не следует использовать VARCHAR для любых строк, потому что для этого типа поля характерен ряд недостатков. Например, MySQL-сервер обрабатывает поля типа CHAR гораздо быстрее, потому что их длина предопределена. Если требуется сохранять в базе данных строки, которые незначительно отличаются друг от друга по длине, то рекомендуется использовать тип CHAR. Более того, когда все строки имеют одинаковую длину, тип VARCHAR занимает больше дискового пространства, так как кроме самой строки в дополнительном байте этого поля хранится длина этой строки.

В заключение стоит отметить, что если в таблице имеется хотя бы одно поле VARCHAR, то все символьные поля преобразовываются к типу VARCHAR, даже если они определены иначе.

Индексы и ключи

Неопытные разработчики баз данных часто жалуются на медлительность используемых СУБД. Проблема часто объясняется отсутствием *индекса*. Индекс представляет собой отдельный отсортированный список значений определенного поля (или полей) в таблице. Прежде чем изучать причину, по которой индексирование таблиц радикально влияет на производительность базы данных, рассмотрим таблицу, не имеющую индексов. Такая таблица, по сути, представляет собой простой текстовый файл, потому что СУБД осуществляет поиск данных в этой таблице последовательно. Записи в реляционной базе данных не вставляются в определенном порядке — сервер вставляет их произвольно. Чтобы гарантировать, что будут найдены все соответствующие определенному критерию записи, СУБД полностью сканирует таблицу, а это очень медленная и неэффективная операция, особенно если в таблице имеется только несколько совпадений с критерием отбора.

Теперь рассмотрим индексированную таблицу. Вместо продвижения по всей таблице в поисках записей, соответствующих требованиям пользователя, СУБД может про-

сканировать индекс. Поскольку индекс — ни что иное как упорядоченный список, такое сканирование выполняется очень быстро. Индекс направляет сервер к релевантным записям в таблице базы данных, и необходимость в полном сканировании таблицы отпадает.

Почему не сортируются сами таблицы? Сортировка таблиц имела бы практический смысл, если бы было заранее известно, что поиск данных в таблице будет всегда осуществляться только по одному полю. Однако такое бывает редко. Так как сортировать таблицу по нескольким полям невозможно, лучше всего использовать индекс, который обособлен от таблицы.

Рассмотрим случай, когда приходится выполнять поиск одновременно в нескольких таблицах. Это тот случай, когда можно получить *реальный* выигрыш от использования индекса. Поиск возможных совпадений в объединенных таблицах без индексов — очень плохая идея. СУБД должна будет проверить все возможные комбинации строк в одной таблице со строками в другой. Обработка двух таблиц по 500 записей в каждой потребовала бы $500 \times 500 = 250\,000$ комбинаций. Индексирование значительно ускоряет поиск. СУБД проверяет индекс первой таблицы в поисках позиции совпадений с критериями первой части запроса, а затем, используя данный индекс для второй таблицы, ищет в ней совпадения со второй частью запроса. Иными словами, релевантные записи выбираются непосредственно.

Первичный ключ (primary key) — специальный индекс, который, как было показано в начале главы, используется для идентификации записей в таблице и для связи этой таблицы с другими таблицами, обеспечивая таким образом реляционную модель базы данных. Каждая связанная таблица должна иметь один (и только один) первичный ключ.

Индексы и первичные ключи можно создавать на основе комбинаций полей. Чтобы сформировать таким способом ключ, комбинация значений всех полей ключа должна быть уникальной.

Поскольку индекс обеспечивает значительное повышение производительности, можно ли создавать как можно больше индексов для достижения максимальной производительности? Не всегда. Использование индексов является надежным способом повышения скорости поиска и получения данных из базы, однако оно сокращает производительность при сохранении или обновлении записей, а также увеличивает размер таблиц. Почему? При вставке записи в индексированную таблицу СУБД должна записать позицию этой записи в соответствующей индексной таблице, а это требует определенных затрат времени и дискового пространства.

Кроме того, если в таблице имеется несколько индексов, то требуется также множество операций записи в индексной таблице. Поэтому рекомендуется создавать столько индексов, сколько действительно необходимо. Рекомендуется индексировать только те столбцы, по которым будет часто выполняться поиск или сортировка данных. Если потребуется, то дополнительные индексы для таблицы можно создавать по мере увеличения производительности.

Запросы

Для создания запросов используются *SQL-операторы* или *команды*. Посредством запроса приложение обращается к СУБД, которая затем возвращает приложению записи, удовлетворяющие заданным в запросе критериям. Запросы возвращают массив записей, удовлетворяющих заданным условиям и содержащих информацию из выбранных полей. РНР и другие языки программирования, которые поддерживают подключение к базам данных, могут обрабатывать возвращаемый массив, как любой другой массив переменных. (Эта тема подробнее рассматривается далее.) Возвращаемый

массив записей, который называется *результатирующим множеством* (*result set*) — ответ СУБД на данный запрос. Например, если ввести запрос на выборку записей, содержащих в поле имени строку John, то база данных вернет все соответствующие этому запросу записи. Если таких записей нет, то будет возвращено значение NULL.

Некоторые SQL-операторы буквально являются командами, т.е. они не запрашивают данные, а указывают СУБД выполнить какие-либо действия, например: команда вроде “удалить записи, содержащие в поле имени значение John” не возвращает результирующего множества.

NULL

Рассмотрим следующую ситуацию: класс пишет диктант, а учитель должен оценить работы учеников и внести результаты в таблицу базы данных. Вносить значение по умолчанию в столбец результатов до того, как работа оценена, было бы нечестно, потому что в данном случае значение по умолчанию просто неуместно. Что в таком случае можно было бы вставить в столбец результата, чтобы обозначить, что оценка еще не поставлена?

Предположим, что требуется создать таблицу базы данных, содержащую информацию об исчезающих видах птиц. Одно из полей должно содержать значение максимальной зарегистрированной скорости полета для каждой птицы. Вводя данные о пингвинах, разработчик вспоминает, что пингвины не умеют летать. Какое в этом случае значение следует внести в столбец скорости, чтобы обозначить, что данная характеристика к пингвинам неприменима?

В обоих случаях данные пропускаются. Единственное различие заключается в том, что в первом случае ситуация временная, так как учитель вскоре добавит в таблицу недостающие результаты, а во втором случае значение для скорости полета пингвина невозможно получить в принципе. Поэтому необходим способ представления пропущенных данных в полях.

В MySQL-таблицах NULL представляет пропущенное значение. NULL не принадлежит к какому-либо определенному типу данных, но может заменить любое значение. Поскольку NULL не является ни типом данных, ни значением, но записывается в поле, концепция NULL обычно трудна для понимания. Часто программисты неправильно понимают смысл использования NULL. Например, распространено ошибочное мнение о том, что NULL — это замена нуля. Это неверно, потому что ноль (0) — значение, в отличие от NULL. Строки, состоящие из одного или нескольких пробелов, и строки нулевой длины также часто путают с NULL, однако строки — тип данных, а NULL — нет. NULL — ничто, ни тип данных, ни значение.

Что происходит, когда результирующее множество одного из запросов содержит NULL, и это множество затем используется в программе для последующих вычислений? Для математических вычислений рекомендуется придерживаться практического правила “распространять” NULL. Любые арифметические операции с NULL возвращают NULL. Это имеет смысл, потому что как иначе можно представить результат, когда все необходимые для вычислений данные отсутствуют? Это правило также применимо к ситуации, когда NULL делится на ноль, в результате чего возвращается NULL.

Команды запросов

MySQL-запросы, используемые для манипуляции данными в таблицах, могут состоять из следующих основных команд:

- ❑ SELECT: получает данные из базы;
- ❑ DELETE: удаляет данные из базы;

- ❑ INSERT: вставляет данные в базу;
- ❑ REPLACE: заменяет данные в базе; если же такая запись в таблице существует, то в нее будут внесены новые данные;
- ❑ UPDATE: обновляет данные в таблице.

Остальные команды предназначены скорее для создания или модификации структуры базы данных, нежели для манипуляции данными:

- ❑ CREATE: создает базу данных, таблицу или индекс;
- ❑ ALTER: модифицирует структуру таблицы;
- ❑ DROP: уничтожает базу данных или таблицу.

Более подробно эти команды описываются в двух последующих главах. А теперь для того чтобы понять, как происходит использование запросов, рассмотрим обычную форму MySQL-запроса SELECT, который выбирает определенные записи из таблицы:

```
mysql> SELECT поле1, поле2, ... , полеN FROM имя_таблицы WHERE условие;
```

Прежде всего, необходимо отметить, что все запросы завершаются точкой с запятой, как и PHP-операторы. Выражение запроса может занимать несколько строк. Следующий, несколько более специфический запрос, по сути, повторяет общий случай:

```
mysql> SELECT last_name, first_name  
-> FROM user  
-> WHERE first_name = 'John'
```

Рассмотрим подробнее предложения FROM, WHERE и ORDER BY данного запроса. Запрос возвращает все записи *из* таблицы user, *в которых* значением поля first_name является строка John. Если такая таблица и записи, удовлетворяющие данному запросу, существуют, то результатом запроса будет следующее:

```
Simpleton John  
Smith John  
Thomas John
```

Рассмотрим теперь практические примеры работы с MySQL.

Практическое применение MySQL

Для начала изучим работу с MySQL-сервером посредством клиентской программы mysql и несколько простых запросов к базе данных. Этот раздел познакомит читателя с настройками баз данных и созданием нескольких новых пользователей (не имеющих прав пользователя root), а практические примеры продемонстрируют работу запросов.

Запуск клиентской программы mysql

Запустите клиент mysql, введя в командной строке следующую команду:

```
> mysql -uUSER -pPASSWORD -hHOST
```

Аргументы USER, PASSWORD и HOST необходимо заменить соответствующими значениями, отражающими персональные настройки. Например, если в используемой системе имя пользователя и пароль равны phpuser и phppass соответственно, а сервер баз данных имеет адрес db.whatever.com, то команда будет следующей:

```
> mysql -uphpuser -phppass -hdb.whatever.com
```


Все аргументы являются необязательными. Для пропущенных аргументов используются следующие значения по умолчанию:

Аргумент	Значение
-u	Имя пользователя в учетной записи shell
-p	Нет пароля
-h	localhost

Получив такую команду, mysql-клиент подключается к серверу баз данных, работающему на заданном узле, используя заданную комбинацию идентификатора пользователя и пароля. Кроме того, можно указать используемую базу данных, введя ее имя в конце команды:

```
> mysql -uUSER -pPASSWORD test
```

В ответ на такую команду сервер должен выдать следующее сообщение:

```
Welcome to the MySQL monitor.  Commands end with ; or \ g.
Your MySQL connection id is 4 to server version: 4.0.18-nt
```

```
Type 'help;' or '\ h' for help. Type '\ c' to clear the buffer.
```

```
mysql>
```

```
MySQL-монитор. Команды должны завершаться символом ; или \ .
Идентификатор данного соединения с MySQL 4; версия сервера: 4.0.18-nt
```

```
Введите 'help;' или '\ h' для получения справки. Введите '\ c' для очистки буфера.
mysql>
```

Если вместо этого mysql сообщит, что не может подключиться к указанному серверу, то следует проверить правильность аргументов команды.

Выбор используемой базы данных

Чтобы получить перечень доступных баз данных, используется команда SHOW DATABASES:

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| mysql    |
| test     |
+-----+
2 rows in set (0.00 sec)
```

Это почти та же самая информация, которую ранее выдавала программа mysqlshow: до сих пор в системе были доступны две базы данных — mysql и test. Кроме того, в данном случае в выводе имеется информация о количестве записей (2) и о времени, которое потребовалось серверу на выполнение запроса (0,00 секунд с точностью до двух десятичных знаков).

Фактически, зная правильный синтаксис, клиентскую программу mysql можно заставить выполнять почти все те же действия, которые могут выполнять другие более специализированные программы. Рассмотрим эти возможности подробнее.

Чтобы выбрать определенную базу, можно использовать команду вида USE имябазыданных:

```
mysql> USE mysql;
Database changed
```

Теперь MySQL-сервер готов к работе с базой данных mysql. Предупреждение: таблицу user в этой базе данных следует использовать с особой осторожностью — она содержит важную информацию, удалять которую нежелательно.

С целью читабельности кода рекомендуется писать ключевые слова SQL в верхнем регистре, а определенные пользователем имена, такие как имена таблиц и полей, — в нижнем. Не стоит забывать, что в Linux/Unix-платформах аргументы чувствительны к регистру символов: Mysql, MYSQL и mysql — имена абсолютно разных баз данных.

Просмотр таблиц в базе данных

Для получения списка существующих в текущей базе данных таблиц используется команда SHOW TABLES:

```
mysql> SHOW TABLES;
+-----+
| Tables in mysql |
+-----+
| columns_priv   |
| db              |
| func            |
| host            |
| tables_priv     |
| user            |
+-----+
6 rows in set (0.00 sec)
```

В данном случае также выводится больший объем информации, чем при использовании mysqlshow. Теперь можно подробнее изучить таблицы. Например, чтобы просмотреть структуру таблицы user, можно использовать команду DESCRIBE (или DESC):

```
mysql> DESC user;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Host  | varchar(60) binary | | PRI | | |
| User  | varchar(16) binary | | PRI | | |
| password | varchar(16) | | | | |
| Select_priv | enum('N','Y') | | | N |
| Insert_priv | enum('N','Y') | | | N |
| Update_priv | enum('N','Y') | | | N |
| Delete_priv | enum('N','Y') | | | N |
| Create_priv | enum('N','Y') | | | N |
| Drop_priv | enum('N','Y') | | | N |
| Reload_priv | enum('N','Y') | | | N |
| Shutdown_priv | enum('N','Y') | | | N |
| Process_priv | enum('N','Y') | | | N |
| ...   | ... | ... | ... | ... |
| ...   | ... | ... | ... | ... |
| File_priv | enum('N','Y') | | | N |
| Grant_priv | enum('N','Y') | | | N |
| References_priv | enum('N','Y') | | | N |
| max_connections | int(11) unsigned | | | 0 |
+-----+-----+-----+-----+-----+-----+
31 rows in set (0.00 sec)
```

Вывод команды описывает структуру таблицы `user` — какие поля определены и как они сконфигурированы. Например, из распечатки ясно, что в поле `Host` могут храниться значения длиной не более 60 символов и что это поле определено в качестве первичного ключа. Поле `User` также определено как первичный ключ. Это не означает, что оба эти поля являются первичными ключами (это не допускается). Просто комбинация полей `Host` и `User` работает в качестве первичного ключа в формате `Host-User`.

Например, если пользователю `john` на узле `localhost` даны права доступа к серверу, то для его записи значением первичного ключа становится строка `localhost-john`. Записать в данную таблицу еще одного пользователя с именем `john` на узле `localhost` невозможно, но можно записать пользователя `john` на узле `whatever.com`.

Все поля в таблице `user`, кроме `Host`, `User` и `Password`, объявлены как `ENUM('N', 'Y')`. Это означает, что во всех этих полях может использоваться только одно значение из перечисленных в скобках (в данном случае `N` или `Y`). Необходимо отметить, что значением по умолчанию (когда значение не задано) будет буква `N`.

Использование SQL для просмотра данных

Все рассмотренные выше запросы к `mysql`-клиенту специфичны для СУБД `MySQL` и необязательно будут работать на других системах, поддерживающих `SQL`. Теперь, подойдя на шаг ближе к фактическим данным, можно использовать “настоящие” `SQL`-операторы.

Предположим, что требуется выяснить, зарегистрированы ли в настоящий момент пользователи, имеющие привилегии для доступа с локальной машины `localhost`. Это можно сделать с помощью следующего `SELECT`-запроса:

```
mysql> SELECT User, Host FROM user;
+-----+-----+
| User      | Host      |
+-----+-----+
| root      | localhost |
| dodge     | doggiesr  |
| james     | digistrawb|
+-----+-----+
3 rows in set (0.00 sec)
```

В запросе содержится лишь три записи, поэтому нетрудно заметить, что в текущий момент зарегистрирован только один локальный пользователь `root`. Конечно, выяснить это будет сложнее при наличии нескольких сотен зарегистрированных пользователей, но на этот случай есть другой способ получения необходимого результата:

```
mysql> SELECT User FROM user WHERE Host='localhost';
+-----+
| User      |
+-----+
| root      |
+-----+
1 row in set (0.00 sec)
```

Использование предложения `WHERE` сужает диапазон выбираемых записей. Это предложение работает подобно оператору `if` в `PHP`, кроме того, что в случае `WHERE` для проверки равенства используется один знак равенства (`=`), а не два (`==`), как в `if`.

Если требуется выбрать все доступные поля, а не несколько указанных, то можно использовать специальный символ `*`. Например:

```
mysql> SELECT * FROM tablename;
```

Данная команда выбирает все поля всех записей в таблице `tablename`.

Манипуляции данными

Создадим нового пользователя, *вставив* в таблицу `user` новую запись с помощью команды `INSERT`:

```
mysql> INSERT INTO user VALUES (
-> 'localhost',
-> 'phpuser',
-> Password('phppass'),
-> 'N', 'N', 'N', 'N', 'N', 'N', 'N',
-> 'N', 'N', 'N', 'N', 'N', 'N', 'N',
-> 'N', 'N', 'N', 'N', 'N', 'N', 'N',
-> 'N', 'N', 'N', 'N', 'N', 'N', 'N');
Query OK, 1 row affected (0.00 sec)
```

Данный `INSERT`-запрос создает в таблице `user` запись для пользователя `phpuser`, использующего пароль `phppass`, не предоставляя никаких прав доступа. Каждое строковое значение помещается между двумя одинарными кавычками (подробно этот момент обсуждается далее). Очевидно, что клиент `mysql` сообщает о том, что запрос был успешно выполнен и что была вставлена одна запись, как и ожидалось.

MySQL сохраняет пользовательские пароли, предварительно зашифровав их. Для шифрования паролей в MySQL используется собственная схема (которая отличается от схемы Linux) с применением встроенной MySQL-функции `password()`.

Предположим теперь, что пользователю `phpuser` (который в данный момент не имеет прав доступа вообще) необходимо назначить административные права `Reload_priv` и `Shutdown_priv`. Обновить таблицу `user` можно с помощью следующего `UPDATE`-запроса:

```
mysql> UPDATE user SET Reload_priv='Y', Shutdown_priv='Y'
-> WHERE User='phpuser';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Клиент `mysql` сообщает, что условию соответствовала только одна запись и что она была успешно обновлена. Теперь пользователь `phpuser` обладает некоторыми привилегиями администратора сервера: ему разрешено только перезагружать и останавливать сервер.

Предложение `WHERE` является необязательным и если оно опущено, то `UPDATE`-запрос изменяет все записи в заданной таблице, используя при этом новые значения. Предупреждение: необходимо исключить случайное изменение записей. Ночной кошмар может стать реальностью, если невнимательно созданный `UPDATE`-запрос, например, обновит 10 000 пользовательских записей, назначив им один и тот же пароль. Еще хуже, если все записи в таблице будут случайно удалены в результате выполнения следующего `DELETE`-запроса:

```
mysql> DELETE FROM test;
Query OK, 0 rows affected (0.00 sec)
```

В данном случае были удалены все записи в таблице `test`. Отчет `mysql` может ввести в заблуждение, поскольку сообщает, что запрос не затронул ни одной записи. На самом деле это не так. После выполнения запроса сервер не может определить количество затронутых (удаленных) строк, так как в результате запроса они удаляются безвозвратно. Следует отметить, что в этом примере команда `DELETE` не содержала никаких предложений. С помощью предложения `WHERE` можно точно указать удаляемые записи:

```
DELETE FROM имятаблицы WHERE условие (я);
```

Предположим теперь, что пользователю `phpuser` необходимо предоставить все привилегии администратора сервера. Что для этого требуется? Можно создать другой UPDATE-запрос или с помощью REPLACE-запроса заменить целиком запись данного пользователя. В последнем случае синтаксис весьма прост:

```
mysql> REPLACE INTO user VALUES(
-> 'localhost', 'phpuser', Password('phppass'),
-> 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
-> 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
-> 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
-> 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y');
Query OK, 1 row affected (0.00 sec)
```

REPLACE-запрос заменяет старую запись новой. Обратите внимание, что и старая и новая записи должны иметь одинаковое значение в поле, которое определено как ключ (или в полях `Host` и `User` (в данном случае) для комбинированного ключа). Различие между командами UPDATE и REPLACE заключается в том, что UPDATE заменяет только выбранный набор полей в записи, а REPLACE заменяет всю запись новыми значениями.

Наконец, чтобы активизировать заново созданную учетную запись `phpuser`, необходимо обновить привилегии, перезагрузив информацию о них из таблицы. (Сервер обычно считывает информацию о привилегиях доступа только однажды при загрузке.) Для этого сначала следует выйти из mysql-клиента, введя в командной строке команду `quit` (в противном случае программа выдаст сообщение о некорректном SQL-синтаксисе). Затем можно ввести команду:

```
> mysqladmin -uUSER -pPASSWORD -hHOST flush-privileges
```

или после входа в mysql можно ввести команду `FLUSH PRIVILEGES`, чтобы изменения отразились в клиентской программе:

```
mysql> FLUSH PRIVILEGES;
```

Использование команд GRANT и REVOKE

Команды GRANT и REVOKE позволяют распределять и отменять привилегии на использование баз данных. Указанные команды применимы на разных уровнях структуры привилегий в СУБД: на глобальном уровне баз данных, на уровне таблиц, а также на уровне столбцов — в зависимости от того, насколько точный контроль необходим администратору. Естественно, любые права, предоставленные пользователю, можно отменить с помощью соответствующей команды REVOKE.

В последующих разделах показано, как использовать команды для управления правами доступа к таблицам. Например, в предыдущем разделе новый пользователь создавался путем непосредственного внесения изменений в таблицу `user`. Той же цели можно достичь более простым способом, используя команду GRANT. Рассмотрим эту команду подробнее.

GRANT

Ниже представлена простейшая форма команды GRANT:

```
mysql> GRANT ALL PRIVILEGES ON *.* TO
-> phpuser@localhost IDENTIFIED BY 'phppass';
```

Данная команда предоставляет пользователю `phpuser`, использующему пароль `phppass`, на узле `localhost` все права доступа к каждой базе данных в системе. Обратите внимание, что аргументы `user` и `host` указываются *без кавычек*.

Если требуется предоставить этому пользователю доступ только к таблицам базы данных `test`, команда будет выглядеть так:

```
mysql> GRANT ALL PRIVILEGES ON test.*
-> TO phpuser@localhost IDENTIFIED BY 'phppass';
```

Здесь запись `test.*` обозначает все таблицы в базе данных `test`. Аналогично, можно предоставлять доступ только к одной таблице в базе данных, например, к таблице `sample` базы данных `test`:

```
mysql> GRANT ALL PRIVILEGES ON test.sample
-> TO phpuser@localhost IDENTIFIED BY 'phppass';
```

Заменяя ключевые слова `ALL PRIVILEGES` перечнем типов запросов, можно предоставить ограниченное множество привилегий доступа:

```
mysql> GRANT SELECT,INSERT,UPDATE ON test.*
-> TO phpuser@% IDENTIFIED BY 'phppass';
```

Данная команда разрешает пользователю `phpuser` вводить только запросы `SELECT`, `INSERT` и `UPDATE` к любой таблице в базе данных `test`. Все остальные запросы, такие как `DELETE`, не допускаются. Весьма полезны специальные символы, такие как `*`. Например, заменив `localhost` на специальный символ `%`, можно предоставить пользователю `phpuser` доступ к указанным таблицам с любого узла.

Узел также можно указать с помощью частично заданного имени домена: например, с помощью записи `phpuser@%.whatever.com` можно предоставить доступ к серверу только пользователям с идентификатором `phpuser`, подключающимся с домена `whatever.com`.

Можно еще точнее определить полномочия пользователя, указав поля, к которым пользователь может иметь доступ:

```
mysql> GRANT SELECT (User, Host) ON mysql.user
-> TO phpuser@localhost IDENTIFIED BY 'phppass';
```

В данном случае пользователь `phpuser` может вводить только `SELECT`-запросы на поля `User` и `Host` в таблице `user`.

Можно использовать предложение `WITH GRANT OPTION`. Оно позволяет предоставить определенному пользователю возможность назначать другим пользователям любой уровень привилегий (в пределах разрешенных для него самого уровней):

```
mysql> GRANT ALL PRIVILEGES ON test.*
-> TO phpuser@localhost IDENTIFIED BY 'phppass' WITH GRANT OPTION;
```

Данная команда эквивалентна созданию другого суперпользователя с именем `phpuser`. Соблюдайте осторожность, используя предложение `WITH GRANT OPTION`: два пользователя с разными привилегиями, обменявшись ими, могут легко расширить свои привилегии.

REVOKE

Команда `REVOKE` отменяет привилегии определенного пользователя. Например, если требуется отменить *все* привилегии пользователя `phpuser`, то можно ввести следующую команду:

```
mysql> REVOKE ALL PRIVILEGES ON *.* FROM phpuser;
```

Можно указать несколько пользователей, разделяя их имена запятыми:

```
mysql> REVOKE ALL PRIVILEGES ON *.*
-> FROM phpuser@localhost, phpuser2, phpuser3;
```

Команда будет работать верно только в том случае, когда все заданные пользователи существуют и указаны правильные привилегии.

Следующая команда отнимает у пользователя `phpuser` только право на ввод `SELECT`-запроса:

```
mysql> REVOKE SELECT ON *.* FROM phpuser@localhost;
```

Кроме того, можно задать необходимые имена полей:

```
mysql> REVOKE SELECT (User, Host) ON mysql.user FROM phpuser@localhost;
```

Если с помощью команд `GRANT` или `REVOKE` модифицируются таблицы привилегий, то изменения вступают в действие немедленно. Обновлять привилегии или перезагружать сервер необязательно.

Подключение к базам данных MySQL из PHP-программ

Вернемся к PHP и рассмотрим возможность использования всей мощи баз данных в динамических Web-страницах, написанных на PHP.

На данной стадии запоминать все эти сложные SQL-операторы необязательно, как и разбираться во всех мельчайших деталях. В настоящий момент достаточно общего понимания методики и способов взаимодействия с реляционными базами данных, такими как MySQL. Гораздо более полное понимание всей специфики баз данных придет потом при разработке собственных приложений.

Подобно тому как для доступа к серверу баз данных MySQL требуется клиентская программа, PHP-программам требуется клиентский код для обмена данными с MySQL.

Во времена PHP4 осуществить это было нетрудно, поскольку весь необходимый код был интегрирован в сам PHP. К сожалению, теперь это не так, и PHP5 ожидает, что клиентские библиотеки MySQL с поддержкой PHP присутствуют в системе еще до компиляции PHP. Для Linux-пользователей эта проблема решается весьма просто: стоит лишь скомпилировать PHP5 с параметром `--with-mysql`.

Для Windows-пользователей дело обстоит несколько сложнее. Необходимо выполнить следующие действия:

1. Убедиться, что библиотека `libmysql.dll` находится в корневом каталоге системы (`libmysql` для версий MySQL 4.1 и выше).
2. Раскомментировать `mysql`-расширение в разделе расширений файла `php.ini`.
3. Убедиться, что файл `php_mysql.dll` находится в таком каталоге, который PHP может найти (параметр `extension_dir` в `php.ini`).
4. Перезапустить Web-сервер после внесения изменений.

Здесь нет ничего сложного — все делается так же, как и для других расширений в PHP. Теперь рассмотрим, как заставить PHP5 и MySQL обмениваться данными друг с другом.

Связь PHP и MySQL

Для работы PHP-программ с MySQL-сервером (или любой другой СУРБД) необходимо выполнить следующие действия.

1. Открыть соединение с сервером.
2. Выполнить работу с базами данных на сервере.
3. Закрыть соединение.

Эта последовательность аналогична последовательности при работе с файлами и каталогами.

Основные функции соединения

Рассмотрим несколько базовых PHP-функций для работы с MySQL.

- ❑ `mysql_connect()`: создает соединение с MySQL-сервером. Принимает три строковых аргумента: имя узла, имя пользователя базы данных и пароль пользователя. Функция в случае успешного подключения к заданному MySQL-серверу возвращает идентификатор соединения (или NULL в случае ошибки):

```
$link_id = mysql_connect("localhost", "phpuser", "phppass");
```

Идентификатор соединения работает во многом так же, как и дескриптор файла или каталога. Он используется позднее для ввода запросов. Все эти аргументы являются необязательными, и если все они пропущены, то по умолчанию используются значения "localhost", пользовательское имя владельца Web-сервера и пустой пароль.

- ❑ `mysql_close()`: соединение с MySQL-сервером закрывается после завершения работы сценария. Чтобы закрыть соединение раньше, необходимо использовать следующую функцию, передав ей в качестве аргумента идентификатор соединения:

```
mysql_close($link_id);
```

Функция возвращает true при успешном выполнении и false в случае ошибки. Если идентификатор соединения опущен, то используется открытое ранее соединение (если иное не оговорено особо, то все `mysql_*`-функции, принимающие обязательный аргумент идентификатора соединения, в случае отсутствия этого аргумента используют ранее открытое соединение).

- ❑ `mysql_list_dbs()`: PHP-эквивалент MySQL-команды SHOW DATABASES. Единственным аргументом (необязательным) является идентификатор соединения. Функция возвращает указатель (\$result) на массив, содержащий имена доступных баз данных:

```
$result = mysql_list_dbs($link_id);
```

- ❑ `mysql_select_db()`: используется для выбора базы данных и возвращает true при успешном выполнении и false, если возникла ошибка. В качестве аргумента данная функция принимает имя базы данных, хотя аргумент идентификатора соединения является необязательным. Например:

```
mysql_select_db("mysql", $link_id);
```

Если соединение не было установлено заранее, то перед выбором заданной базы данных функция пытается установить соединение. Рассмотрим пример, который иллюстрирует подключение к локальному MySQL-серверу из PHP-программы с использованием учетной записи, созданной ранее:

```
$link_id = mysql_connect("localhost", "phpuser", "phppass");  
if(mysql_select_db("mysql", $link_id)) echo "Подключение к узлу localhost выполнено."  
else die ("Подключение не состоялось.");
```


Код устанавливает соединение с локальным MySQL-сервером и выбирает базу данных привилегий mysql. Если соединение установить не удалось, то отображается соответствующее сообщение об ошибке.

- `mysql_list_tables()`: эквивалент MySQL-команды `SHOW TABLES`. В качестве аргументов принимается имя базы данных и необязательный идентификатор соединения. Функция возвращает указатель на массив, содержащий имена доступных таблиц, связанных с этой базой данных:

```
$result = mysql_list_tables("mysql", $link_id);
```

- `mysql_num_rows()`: используется для определения количества строк в результирующем множестве, возвращаемом заданным запросом. В качестве аргумента функция принимает указатель на результирующее множество:

```
$num_rows = mysql_num_rows($result);
```

Эту функцию следует применять к результирующим множествам, возвращаемым SELECT-запросами и другими функциями, выбирающими записи баз данных.

- `mysql_affected_rows()`: используется для получения количества записей, затронутых запросами `INSERT`, `UPDATE` или `DELETE`, вместо функции `mysql_num_rows()`. Аргументом данной функции является необязательный идентификатор соединения:

```
$num_rows = mysql_affected_rows($link_id);
```

DELETE-запрос без предложения WHERE удаляет все записи в заданной таблице и заставляет функцию `mysql_affected_rows()` возвращать нуль.

Возвращаемое функцией `mysql_num_rows()` или `mysql_affected_rows()` значение фактически представляет собой количество выбранных/измененных записей. Возвращаемым значением функций `mysql_list_dbs()` и `mysql_list_tables()` является указатель на результирующее множество.

- `mysql_fetch_row()`: используется для выборки строк из возвращаемого сервером результирующего множества. Функция принимает указатель на результирующее множество, возвращенный предыдущим запросом, и возвращает массив, соответствующий выбранной строке (или `false`, если строк больше не осталось):

```
$fetched_row = mysql_fetch_row($result_set);
```

PHP создает внутренний указатель на строку, возвращенную предыдущим запросом. Каждый последующий вызов данной функции передвигает этот указатель на следующую доступную строку. Когда указатель выходит за пределы результирующего множества, функция `mysql_fetch_row()` возвращает `False`.

Полный перечень MySQL-функций представлен в приложении В.

Практика Подключение к MySQL-серверу из PHP-программы

В следующем примере сценария, `db_connect.php`, используются упомянутые выше функции для перечисления доступных MySQL-пользователю `phpuser` баз данных и таблиц:

```
<?php
$link_id = mysql_connect("localhost", "phpuser", "phppass");
$result = mysql_list_dbs($link_id);
```

```

while($db_data = mysql_fetch_row($result)) {
    echo $db_data[0] . "<BR>";
    $result2 = mysql_list_tables($db_data[0]);
    $num_rows = mysql_num_rows($result2);
    while($table_data = mysql_fetch_row($result2)) {
        echo "--" . $table_data[0] . "<BR>";
    }
    echo "==> $num_rows таблиц в " . $db_data[0] . "<P>";
}
?>

```

Сценарий отображает все доступные базы данных и все таблицы в каждой из них; пример вывода:

```

mysql
--columns_priv
--db
--func
--host
--tables_priv
--user
==> 6 table(s) in mysql
test_db
--sample_table1
--sample_table2
==> 2 table(s) in test_db

```

При использовании этих SQL-команд возможно появление сообщений об ошибках. Если это так, тогда следует еще раз прочесть разделы GRANT и REVOKE и предоставить пользователю `phpuser` достаточные привилегии для просмотра таблиц в базах данных, а затем запустить сценарий снова. Полезные рекомендации также содержатся в разделе по обработке ошибок.

Как это работает

Сначала сценарий устанавливает соединение с сервером, используя заданный набор значений: `localhost`, `phpuser` и `phppass`. В случае использования других параметров доступа данные значения следует заменить.

Идентификатор соединения, возвращаемый функцией `mysql_connect()`, используется функцией `mysql_list_dbs()`. Фактически передавать идентификатор соединения необязательно, поскольку `mysql_list_dbs()` в случае отсутствия аргумента будет использовать последнее открытое соединение:

```
$result = mysql_list_dbs();
```

Весьма распространенной ошибкой является перезапись указателя результата при вводе другого запроса во время прохождения по результирующему множеству, возвращенному предыдущим запросом. Подобно тому как можно открыть множество файлов, используя различные дескрипторы файлов, следует создавать несколько результирующих множеств, назначая каждому из них уникальный указатель.

В данном случае для обработки результирующего множества, возвращенного функцией `mysql_list_dbs()`, в сценарии используется цикл `while`. Еще один цикл `while` просматривает каждый элемент в результирующем множестве, возвращенном функцией `mysql_list_tables()`. Оба цикла завершаются, когда в результирующих множествах не остается строк, потому что переменные `$db_data` и `$table_data` равны `False`, когда функция `mysql_fetch_row()` заканчивает выборку возвращенных строк.

```

while($db_data = mysql_fetch_row($result)) {
    echo $db_data[0] . "<BR>";
    $result2 = mysql_list_tables($db_data[0]);
    $num_rows = mysql_num_rows($result2);
    while($table_data = mysql_fetch_row($result2)) {
        echo "--" . $table_data[0] . "<BR>";
    }
    echo "==> $num_rows таблиц в " . $db_data[0] . "<P>";
}

```

Каждый из массивов, возвращенных функциями `mysql_list_dbs()` и `mysql_list_tables()`, содержит только один элемент; имя базы данных `$db_data[0]` и имя таблицы `$table_data[0]` соответственно. Если нужно получить несколько полей из таблицы, то можно обратиться к каждому из них, используя соответствующий индекс:

```

$result_set = mysql_fetch_row($result);
$first_column = $result_set[0];
$second_column = $result_set[1];
$third_column = $result_set[2];

```

Здесь не использовалась функция `mysql_close()`, она редко используется в реальных приложениях, если нет необходимости по какой-либо причине закрывать соединение до завершения работы приложения. PHP автоматически закрывает открытое соединение перед завершением работы сценария.

В последующих главах работа тандема MySQL-PHP рассматривается более подробно.

Обработка серверных ошибок

Теперь попытаемся заменить имя пользователя базы данных в последнем сценарии на `no_such_user`. Если в файле `php.ini` не настроено подавление ошибок, то сценарий выдаст следующие сообщения:

```

Warning: MySQL Connection Failed: Access denied for user: 'no_such_user@local
host' (Using password: YES) in /home/apache/htdocs/db_connect.php on line 3
Предупреждение: ошибка подключения к MySQL: пользователю 'no_such_user@local
host' отказано в доступе (использование пароля: да) файл
/home/apache/htdocs/db_connect.php строка 3

```

```

Warning: Supplied argument is not a valid MySQL-Link resource in /home/apache/
htdocs/db_connect.php on line 5
Предупреждение: неверная ссылка на MySQL-соединение файл /home/apache/
htdocs/db_connect.php строка 5

```

```

Warning: Supplied argument is not a valid MySQL result resource in /home/
apache/htdocs/db_connect.php on line 6
Предупреждение: неверная ссылка на MySQL-результат файл /home/apache/
htdocs/db_connect.php строка 6

```

```

Warning: Supplied argument is not a valid MySQL result resource in /home/
apache/htdocs/db_connect.php on line 8
Предупреждение: неверная ссылка на MySQL-результат файл /home/apache/
htdocs/db_connect.php строка 8

```

Почему появились подобные неинформативные сообщения? Все просто — только что созданный сценарий не способен определить, какие последствия повлекла за собой ошибка; сценарий “знает” лишь то, что он попытался получить доступ к MySQL-ресурсам, но потерпел неудачу. Сообщения генерируются PHP-интерпретатором, однако ошибка возникла на сервере баз данных.

PHP представляет две функции, которые перехватывают серверные ошибки и определяют, что произошло, используя значения от собственного обработчика ошибок

сервера. Если MySQL-сервер сталкивается с ошибкой при выполнении определенной задачи, то он возвращает текст и номер сообщения об ошибке. Эти значения можно получить с помощью PHP-функций `mysql_errno()` и `mysql_error()`.

Функция `mysql_errno()` возвращает номер ошибки, а `mysql_error()` — текст ошибки от предыдущей MySQL-операции. Обе эти функции принимают в качестве аргумента необязательный идентификатор соединения (по умолчанию — идентификатор ранее открытого соединения):

```
$MYSQL_ERRNO = mysql_errno($link_id);
$MYSQL_ERROR = mysql_error($link_id);
```

Упомянутые функции предполагают, что соединение с сервером уже установлено, поэтому если вызвать их, для того чтобы узнать, почему соединение не устанавливается, никакой информации об ошибках они не дадут, так как нет доступного открытого соединения. Однако, как уже было показано, можно использовать функцию `die()` для прерывания сценария и выдачи сообщения о неудачной попытке соединения:

```
if(!mysql_connect("localhost", "no_such_user", "phppass"))
    die("Подключение не установлено!");
```

Единственная проблема, связанная с данным подходом, заключается в том, что PHP при неудачной попытке подключения все равно выдает множество однотипных сообщений об ошибках. Чтобы предотвратить появление предупреждений или/и сообщений об ошибках, можно либо предварить вызов функции оператором `@` (подавляющим сообщения об ошибках), либо использовать функцию `error_reporting()`.

Функцию `error_reporting()` можно использовать для того, чтобы задать уровень отображения ошибок, который PHP применит к последующему коду. Вызов `error_reporting(0)` гарантирует, что никаких PHP-сообщений об ошибках или предупреждений не появится. Примеры использования этих функций рассматриваются далее.

Устанавливать высокий уровень отображения ошибок в реальной среде нежелательно, потому что предупреждения и ошибки могут отпугнуть пользователей. Высокий уровень рекомендуется использовать только для отладки приложения.

Практика Обработка серверных ошибок

Рассмотрим несколько функций, которые окажутся очень полезными в последующих разделах. Создайте файл с именем `common_db.inc` и введите в него следующий код:

```
<?php
$dbhost = 'localhost';
$dbusername = 'phpuser';
$dbuserpassword = 'phppass';
$default_dbname = 'mysql';

$MYSQL_ERRNO = '';
$MYSQL_ERROR = '';

function db_connect($dbname='') {
    global $dbhost, $dbusername, $dbuserpassword, $default_dbname;
    global $MYSQL_ERRNO, $MYSQL_ERROR;

    $link_id = mysql_connect($dbhost, $dbusername, $dbuserpassword);
    if(!$link_id) {
        $MYSQL_ERRNO = 0;
```

```

    $MYSQL_ERROR = "Не удалось подключиться к узлу $dbhost.";
    return 0;
}
else if(empty($dbname) && !mysql_select_db($default_dbname)) {
    $MYSQL_ERRNO = mysql_errno();
    $MYSQL_ERROR = mysql_error();
    return 0;
}
else return $link_id;
}

function sql_error() {
    global $MYSQL_ERRNO, $MYSQL_ERROR;

    if(empty($MYSQL_ERROR)) {
        $MYSQL_ERRNO = mysql_errno();
        $MYSQL_ERROR = mysql_error();
    }
    return "$MYSQL_ERRNO: $MYSQL_ERROR";
}
?>

```

Созданный файл можно включать в последующие примеры, а также расширить его при необходимости. Обратите внимание на функцию `db_connect()`, предназначенную для установки соединений с MySQL-сервером. Она неоднократно будет использоваться в дальнейших примерах.

Необходимо помнить о том, что этот файл содержит важную информацию, которую нежелательно передавать пользователям. Вообще рекомендуется в целях безопасности хранить файлы, содержащие пароли, в виде простого текста, вне дерева каталогов Web-сервера. PHP сможет получить к ним доступ, и вместе с тем это позволит предотвратить передачу таких файлов пользователям непредусмотренным способом, например, путем “обмана” сервера и вывода файлов в виде простого текста. Файл должен выполняться в PHP-интерпретаторе. Это предотвратит отображение паролей.

Теперь проверим сценарий `db_connect.php` с некорректной комбинацией имени пользователя и пароля или с указанием несуществующей базы данных:

```

<?php
include "common_db.inc";
error_reporting(0);

$link_id = db_connect();
if(!$link_id) die(sql_error());
else echo "Подключение к узлу $dbhost успешно установлено.<BR>";
?>

```

Попытаемся запустить данный сценарий с неверным именем пользователя или паролем; будет выведено следующее сообщение об ошибке:

```
0: Не удалось подключиться к узлу localhost.
```

Если переменная `$default_dbname` пуста, то получим следующее сообщение:

```
1046: No Database Selected (не выбрана база данных)
```

Если в сценарии задана несуществующая база данных (например, `no_such_db`), то будет выведено сообщение:

```
1049: Unknown database (неизвестная база данных) 'no_such_db'
```

Как это работает

После подключения переменных и определений функций задается уровень отображения ошибок (`error_reporting(0)`), который предотвращает появление PHP-сообщений об ошибках или предупреждений.

Также можно подавить ошибки, используя префикс @ в вызовах функций `mysql_connect()` и `mysql_select_db()`.

Если соединение с базой данных создается успешно, то функция `db_connect()` возвращает идентификатор этого соединения. В противном случае устанавливаются глобальные переменные `$MYSQL_ERRNO` и `$MYSQL_ERROR`, и функция возвращает нуль, указывая таким образом на возникновение ошибки. После этого сценарий завершает свою работу:

```
if(!$link_id) die(sql_error());
```

Вместо непосредственного определения сообщения об ошибке вызывается функция `sql_error()`, создающая соответствующее сообщение:

```
function sql_error() {
    global $MYSQL_ERRNO, $MYSQL_ERROR;

    if(empty($MYSQL_ERROR)) {
        $MYSQL_ERRNO = mysql_errno();
        $MYSQL_ERROR = mysql_error();
    }
    return "$MYSQL_ERRNO: $MYSQL_ERROR";
}
```

По существу, эта функция только возвращает символьную строку, содержащую код ошибки и сообщение об ошибке, разделенные двоеточием. Вместе с тем эта функция будет весьма полезной в ситуациях, когда возвращаемые переменные не были определены. Например, можно вызывать функцию `sql_error()` сразу за ошибкой базы данных; проблема заключается в том, что переменные `$MYSQL_ERRNO` и `$MYSQL_ERROR` в таком случае не будут определены. Чтобы эта функция была полезной в такой ситуации (а также в случае предугадываемого сбоя соединения), необходимо проверять значение глобальной переменной `$MYSQL_ERROR`. Если переменная пуста, то следует вызвать функции `mysql_errno()` и `mysql_error()` и назначить значения обеим переменным.

После успешного подключения функция `db_connect()` пытается выбрать базу данных по умолчанию, имя которой задано в глобальной переменной `$default_dbname`. Можно усовершенствовать функцию `db_connect()`, заставив ее принимать необязательный аргумент `$dbname`, так чтобы она выбирала указанную в аргументе базу данных:

```
function db_connect($dbname='') {

    global $dbhost, $dbusername, $dbuserpassword, $default_dbname;
    global $MYSQL_ERRNO, $MYSQL_ERROR;

    $link_id = mysql_connect($dbhost, $dbusername, $dbuserpassword);
    if(!$link_id) {
        $MYSQL_ERRNO = 0;
        $MYSQL_ERROR = "Не удалось подключиться к узлу $dbhost.";
        return 0;
    }
    else if(empty($dbname) && !mysql_select_db($default_dbname)) {
        $MYSQL_ERRNO = mysql_errno();
```

```

        $MYSQL_ERROR = mysql_error();
        return 0;
    }

    else if(!empty($dbname) && !mysql_select_db($dbname)) {
        $MYSQL_ERRNO = mysql_errno();
        $MYSQL_ERROR = mysql_error();
        return 0;
    }

    else return $link_id;
}

```

Теперь функцию можно вызвать так:

```
$link_id = db_connect("sample_db");
```

Идентификатор соединения, записываемый в переменную `$link_id`, автоматически будет связываться с базой данных `sample_db`. Если аргумент не задан, то функция `db_connect()` по умолчанию будет использовать базу данных, указанную в переменной `$default_dbname`.

Создание баз данных и таблиц с помощью MySQL-клиента

Вернемся к созданию баз данных в командной строке из `mysql`-клиента. Для того чтобы создать базу данных `sample_db`, необходимо ввести следующую команду:

```
mysql> CREATE DATABASE sample_db;
Query OK, 1 row affected (0.05sec)
mysql>
```

Удалить базу данных также несложно:

```
mysql> DROP DATABASE sample_db;
Query OK, 0 rows affected (0.00sec)
mysql>
```

В количество затронутых строк `mysql` не включает удаленные строки. Можно удалить тысячи записей, а согласно отчету `mysql` затронутых строк будет все равно 0. Поэтому эту небольшую и на вид безвредную команду следует использовать с особой осторожностью — она полностью удаляет заданную базу данных и все таблицы в ней без каких-либо подсказок пользователю, не говоря уже о подтверждении.

Предположим, что база данных `sample_db` только что была удалена. Создадим ее снова с помощью команды `CREATE` (эта команда часто используется в двух последующих главах). Представленные далее примеры поэтапно показывают, как создавать структуру и записи в базе данных.

Создание таблицы базы — более сложный процесс. Предположим, что требуется создать таблицу с именем `user`. Если начать с очевидного на первый взгляд синтаксиса, то появится следующее сообщение:

```
mysql> CREATE TABLE user;
ERROR 1113: A table must have at least 1 column
(ОШИБКА 1113: в таблице должен быть хотя бы один столбец)
mysql>
```

Хотя база данных может нормально существовать вообще без таблиц, сами таблицы более требовательны. Необходимо задать некоторую структуру, введя в скобках

после имени таблицы несколько описаний полей. Описание поля задается в следующей форме:

```
fieldname TYPE(length)
```

Например, создадим очень простую таблицу с двумя полями:

```
mysql> CREATE TABLE test_table (name CHAR(40), number INT);
Query OK, 0 rows affected (0.06sec)
mysql>
```

Выше уже рассматривались различные типы переменных, поддерживаемые MySQL. Кроме того, существует возможность задать атрибуты для каждого поля — чтобы это сделать, необходимо добавить их к дескриптору. Описанные ниже атрибуты можно использовать в любой комбинации.

Атрибуты	Описание
BINARY	Делает значение поля чувствительным к регистру символов. Работает только с полями типа CHAR и VARCHAR
NULL или NOT NULL	Поля NOT NULL не допускают NULL-значений. Поля, объявленные как NULL, сохраняют значение, заданное по умолчанию, в случае если в поле записывается NULL. Если значение по умолчанию не указано, то в поле просто сохраняется NULL. Если атрибут не задан, то предполагается NULL
DEFAULT <i>default_value</i>	Определяет значение по умолчанию, если система пытается записать в поле значение NULL
AUTO_INCREMENT	Если программа пытается записать в поле AUTO_INCREMENT значение NULL, то фактически в поле записывается значение, на единицу превышающее текущее максимальное значение в этом поле таблицы. AUTO_INCREMENT работает только с уникальными целочисленными полями. В таблице может присутствовать только одно поле AUTO_INCREMENT

После того как заданы типы полей, можно определить конкретные поля как индексы, первичные ключи и уникальные значения. Ниже представлено краткое описание данных элементов.

- ❑ KEY/INDEX: ключевые слова-синонимы, указывающие на то, что данное поле используется как индекс. Значения заданных полей копируются в отдельную индексную таблицу, где они перечисляются вместе с указателями на соответствующие записи в исходной таблице. Затем они сортируются, составляя индекс, во многом напоминающий предметный указатель в конце обычной книги. Если указать в качестве индекса несколько полей ("field1", "field2", ...), то сначала индекс будет отсортирован по полю "field1", а затем все повторяющиеся значения в поле "field1" будут сортироваться по полю "field2" и т.д.
- ❑ PRIMARY KEY: в качестве первичного ключа можно указать только одно поле в таблице. Впоследствии оно будет использоваться как индекс, состоящий из уникальных значений. В результате каждая запись может затем использоваться в других таблицах как уникальная ссылка на запись, которой она принадлежит, — по сути, это связующий материал, на котором основываются реляционные базы данных. Чтобы использовать поле как первичный ключ, его необходимо

объявить как NOT NULL. Стоит еще раз подчеркнуть, что если при определении первичного ключа указать более одного имени поля, то первичный ключ будет создан на основе объединенных имен полей.

- UNIQUE: значение каждой записи в UNIQUE-поле должно быть уникальным в пределах этого поля. Иными словами, если одна запись в поле содержит значение 10, то система гарантирует, что ни одна другая запись в данном поле не имеет такого же значения. Как уже отмечалось, уникальность — обязательное требование для AUTO_INCREMENT-полей.

В двух последующих главах приводится множество примеров, основанных на небольшой реляционной базе данных, состоящей из двух таблиц. Первая таблица называется `user` и служит как ведомость всех пользователей, зарегистрированных на вообразаемом Web-сервере. Вторая таблица, `access_log`, содержит относительные пути к Web-страницам, которые посещал каждый пользователь, а также счетчик посещений каждой страницы. Обе таблицы содержатся в базе данных `sample_db`.

Рассмотрим структуру первой таблицы `user`:

Поле	Тип	Null	Атрибуты
<code>usernumber</code>	MEDIUMINT(10)	нет	AUTO_INCREMENT
<code>userid</code>	VARCHAR(8)	нет	BINARY
<code>userpassword</code>	VARCHAR(20)	нет	BINARY
<code>username</code>	VARCHAR(30)	нет	не задано
<code>userposition</code>	VARCHAR(40)	нет	не задано
<code>useremail</code>	VARCHAR(50)	нет	не задано
<code>userprofile</code>	VARCHAR (250)	нет	не задано

Поле `usernumber` имеет атрибут AUTO_INCREMENT, поэтому каждый раз при добавлении новой записи с NULL-значением в первом поле значение этого поля будет увеличиваться на единицу. Например, когда регистрируется новый пользователь и его запись добавляется в таблицу `user`, номер, назначаемый пользователю, будет на единицу больше, чем максимальное значение, имеющееся в данном поле этой таблицы. Кроме того, это поле объявляется как UNIQUE, а это гарантирует, что ни какие две записи не могут иметь одно и то же значение данного поля.

Значениями полей `userid` и `userpassword` могут быть строки различной длины (длина указывается как 8 и 20 символов соответственно). Эти поля объявляются как BINARY и, следовательно, они чувствительны к регистру символов. По умолчанию символьные поля нечувствительны к регистру.

Поле `userid` определяется как первичный ключ и используется для создания связи с таблицей `access_log`, в которой также имеется поле `userid`.

Поле `userposition` описывает позицию игрока в команде — в примере имеются такие значения, как Mid и Link, но их можно легко изменить на футбольные позиции, такие как защитник или нападающий, если база данных используется для хранения информации о команде по американскому футболу.

Вторая таблица `access_log` определяется следующим образом:

Поле	Тип	Null	Атрибуты
page	VARCHAR(250)	нет	нет
userid	VARCHAR(8)	нет	BINARY
visitcount	MEDIUMINT(5)	нет	нет
accessdate	TIMESTAMP(14)	да	нет

Поле типа `TIMESTAMP` всегда может принимать `NULL`-значение, потому что если дата и время в данном поле не заданы явно, при добавлении новой записи или при обновлении имеющейся сохраняется текущее системное время в формате `YYMMDDhhmmss`.

В таблице `access_log` имеется такое же поле `userid`, как в таблице `user`. Оно служит в качестве ключа для определения связи между двумя таблицами.

Теперь, когда структуры таблиц определены, следует создать сами таблицы и базу данных, в которую они входят (`sample_db`).

Читатели, которые опасаются вводить следующий 12-строчный SQL-оператор (при вводе длинного оператора возможны опечатки), могут воспользоваться приведенными ниже рекомендациями.

Если PHP работает на Unix-машине, то можно использовать в командной строке `mysql` команду `edit`, которая вызывает используемый по умолчанию текстовый редактор (в Windows эта команда не работает). Имя текстового редактора хранится в системной переменной `$EDITOR`. Как правило, таким редактором в Unix-системах является `vi`: `mysql> edit`. Если в процессе ввода SQL-оператора была допущена опечатка, то следует ввести команду `edit` еще раз. Редактор хранит предыдущий ввод.

Предыдущая команда сохраняется клиентской программой `mysql`. Нажав клавишу “стрелка вверх”, можно вернуться на одну строку.

Можно сохранить SQL-оператор в текстовом файле, а затем передать этот файл программе `mysql`: `mysql uhpuser pphppass hlocalhost < query.sql`

Команда `CREATE TABLE` создает таблицу `user` в базе данных `sample_db`:

```
mysql> CREATE TABLE user (
->     usernumber MEDIUMINT(10) DEFAULT '0' NOT NULL AUTO_INCREMENT,
->     userid VARCHAR(8) BINARY NOT NULL,
->     userpassword VARCHAR(20) BINARY NOT NULL,
->     username VARCHAR(30) NOT NULL,
->     userposition VARCHAR(50) NOT NULL,
->     useremail VARCHAR(50) NOT NULL,
->     userprofile TEXT NOT NULL,
->     PRIMARY KEY (userid),
->     UNIQUE usernumber (usernumber)
-> );
Query OK, 0 rows affected (0.69 sec)
mysql>
```

Команда `CREATE TABLE` создает таблицу `access_log`:

```
mysql> CREATE TABLE access_log (
->     page VARCHAR(250) NOT NULL,
->     userid VARCHAR(8) BINARY NOT NULL,
```

```

-> visitcount MEDIUMINT(5) DEFAULT '0' NOT NULL,
-> accessdate TIMESTAMP(14),
-> PRIMARY KEY (userid, page));
Query OK, 0 rows affected (0.00 sec)
mysql>

```

В таблице `access_log` поле `userid` не объявляется как первичный ключ или как уникальное значение, потому что пользователь может посещать множество Web-страниц, и каждый раз при посещении страницы в этой таблице остается соответствующая запись. Вместо этого в качестве первичного ключа определена комбинация полей `userid` и `page`.

Создание демонстрационной базы данных и таблиц с помощью PHP

Предположим, что все прошло по плану, и теперь в системе имеется база данных и две таблицы. И поскольку таблицы так же просто можно генерировать из PHP-сценариев, попробуем создать базу данных и таблицы, используя PHP вместо mysql-клиента.

Для начала нужен способ, позволяющий вводить из PHP запросы к MySQL-серверу. Для этого предназначена функция `mysql_query()`. В качестве первого аргумента она принимает строку SQL-запроса и применяет данный запрос к выбранной в текущий момент базе данных, используя при этом идентификатор соединения, заданный вторым аргументом. Как обычно, идентификатор можно опустить, тогда будет использоваться последнее открытое соединение. Если открытых соединений нет, то функция сначала попытается самостоятельно создать соединение.

Использовать точку с запятой для завершения строки SQL-запроса, когда она передается в качестве аргумента в PHP-функцию, не следует, потому что это может привести к появлению ошибок в сценарии.

Если заданный запрос выполняется успешно, то функция `mysql_query()` возвращает ненулевое значение, которое указывает на возвращенное результирующее множество, или `False` в случае ошибки. Следующий вызов эквивалентен выполнению запроса `SHOW DATABASES` в mysql-клиенте:

```

$link_id = db_connect();
$result = mysql_query("SHOW DATABASES", $link_id);
while($query_data = mysql_fetch_row($result)){
    echo $query_data[0], "<P>";
}

```

Этот код передает ожидаемый список баз данных в таблицу, возвращая указатель на эту таблицу. Затем выбирается и выводится каждая строка, например:

```

mysql
sample_db
test

```

Параметр, представленный переменной `$link_id`, можно опустить, так как функция `mysql_query()` по умолчанию использует последнее открытое соединение.

Чтобы задать имя базы данных, к которой будет применен запрос, можно использовать PHP-функцию `mysql_db_query()`, например:

```

$result = mysql_db_query("sample_db", "SHOW TABLES");

```

Данная функция возвращает список доступных таблиц в базе данных `sample_db`, даже если эта база данных не была выбрана явно с помощью функции `mysql_select_db()`.

Кроме того, функцию `mysql_query()` можно использовать для создания и удаления баз данных:

```
$result = mysql_query("CREATE DATABASE dummy_db");
$result = mysql_query("DROP DATABASE dummy_db");
```

В PHP имеется также альтернативный способ создания и удаления баз данных — пара специально предназначенных для этого функций: `mysql_create_db()` и `mysql_drop_db()`. Обе функции в качестве первого аргумента принимают имя базы данных, которую необходимо создать или удалить, а также необязательный идентификатор соединения и возвращают `True`, если база данных была успешно создана/удалена, и `False` в случае ошибки. Например:

```
$link_id = db_connect();
if(!mysql_create_db("dummy_db", $link_id)) die(sql_error());
echo "База данных dummy_db была успешно создана.";
if(!mysql_drop_db("dummy_db", $link_id)) die(sql_error());
echo "База данных dummy_db была успешно удалена.";
```

Практика Создание базы данных и таблиц

Следующий сценарий представляет собой PHP-аналог описанных выше действий в командной строке. Сначала сценарий создает базу данных `sample_db`, а затем таблицы `user` и `access_log` согласно приведенным ранее определениям.

```
<?php
include "../common_db.inc";

$dbname = "sample_db";
$user_tablename = 'user';
$user_table_def = "usernumber MEDIUMINT(10) NOT NULL AUTO_INCREMENT,";
$user_table_def .= "userid VARCHAR(8) BINARY NOT NULL,";
$user_table_def .= "userpassword VARCHAR(20) BINARY NOT NULL,";
$user_table_def .= "username VARCHAR(30) NOT NULL,";
$user_table_def .= "userposition VARCHAR(50) NOT NULL,";

$user_table_def .= "useremail VARCHAR(50) NOT NULL,";
$user_table_def .= "userprofile TEXT NOT NULL,";
$user_table_def .= "PRIMARY KEY (userid),";
$user_table_def .= "UNIQUE usernumber (usernumber)";

$access_log_tablename = "access_log";
$access_log_table_def = "page VARCHAR(250) NOT NULL,";
$access_log_table_def .= "userid VARCHAR(8) BINARY NOT NULL,";
$access_log_table_def .= "visitcount MEDIUMINT(5) DEFAULT '0' NOT NULL,";
$access_log_table_def .= "accessdate TIMESTAMP(14), KEY page (page),"
$access_log_table_def .= "PRIMARY KEY (userid, page)";

$link_id = db_connect();
if(!$link_id) die(sql_error());

if(!mysql_query("CREATE DATABASE $dbname")) die(sql_error());

echo "База данных $dbname была успешно создана.<BR>";

if(!mysql_select_db($dbname)) die(sql_error());

if(!mysql_query("CREATE TABLE $user_tablename ($user_table_def)"))
    die(sql_error());
```

```
if(!mysql_query("CREATE TABLE $access_log_tablename ($access_log_table_def))
    die(sql_error());

echo "Таблицы $user_tablename и $access_log_tablename были успешно созданы.";

?>
```

Если перед этим был выполнен пример с использованием командной строки, то этот сценарий сгенерирует следующее вполне понятное сообщение об ошибке:

```
1007: Can't create database 'sample_db'. Database exists
1007: Невозможно создать базу данных 'sample_db'. База данных существует
```

Чтобы избежать этого, можно вернуться к командной строке и, используя клиент `mysql`, удалить базу данных `sample_db`, а затем запустить данный сценарий снова. В результате должен появиться следующий вывод:

```
База данных sample_db была успешно создана.
Таблицы user и access_log были успешно созданы.
```

Как это работает

Данный сценарий весьма прост. Он начинается с подключения файла `common_db.inc` при условии, что используются функции `db_connect()` и `sql_error()`, которые были определены ранее:

```
<?php
include "../common_db.inc";
```

Затем определяются переменные, описывающие две таблицы. Здесь выполняется основная работа — необходимо задать те же дескрипторы полей, которые задавались в примере с использованием командной строки:

```
$dbname = "sample_db";
$user_tablename = 'user';
$user_table_def = "usernumber MEDIUMINT(10) NOT NULL AUTO_INCREMENT,";
$user_table_def .= "userid VARCHAR(8) BINARY NOT NULL,";
$user_table_def .= "userpassword VARCHAR(20) BINARY NOT NULL,";
$user_table_def .= "username VARCHAR(30) NOT NULL,";
$user_table_def .= "userposition VARCHAR(50) NOT NULL,";

$user_table_def .= "useremail VARCHAR(50) NOT NULL,";
$user_table_def .= "userprofile TEXT NOT NULL,";
$user_table_def .= "PRIMARY KEY (userid),";
$user_table_def .= "UNIQUE usernumber (usernumber)";

$access_log_tablename = "access_log";
$access_log_table_def = "page VARCHAR(250) NOT NULL,";
$access_log_table_def .= "userid VARCHAR(8) BINARY NOT NULL,";
$access_log_table_def .= "visitcount MEDIUMINT(5) DEFAULT '0' NOT NULL,";
$access_log_table_def .= "accessdate TIMESTAMP(14),KEY page (page),";
$access_log_table_def .= "PRIMARY KEY (userid, page)";
```

Подключение к серверу выполняется с помощью следующего кода:

```
$link_id = db_connect();
if(!$link_id) die(sql_error());
```

Создается база данных `sample_db`:

```
if(!mysql_query("CREATE DATABASE $dbname")) die(sql_error());
echo "База данных $dbname была успешно создана.<BR>";
```

После чего эта база данных выбирается с помощью функции `mysql_select_db()`:

```
if(!mysql_select_db($dbname)) die(sql_error());
```

Последующие вызовы функции `mysql_query()` создают таблицы `user` и `access_log`. Если во время выполнения запроса возникает ошибка, то в браузер пользователя выводится номер ошибки и ее текстовое описание (для этого используется функция `sql_error()`).

```
if(!mysql_query("CREATE TABLE $user_tablename ($user_table_def)"))
    die(sql_error());

if(!mysql_query("CREATE TABLE $access_log_tablename ($access_log_table_def)"))
    die(sql_error());
```

Если сценарий выполняется до конца, то его успешное выполнение подтверждается с помощью следующего кода:

```
echo "Таблицы $user_tablename и $access_log_tablename были успешно созданы.";
?>
```

Напомним, что функция `sql_error()` вызывает функции `mysql_errno()` и `mysql_error()`, если переменная `$MYSQL_ERRNO` не содержит значения:

```
if(empty($MYSQL_ERRNO)) {
    $MYSQL_ERRNO = mysql_errno();
    $MYSQL_ERROR = mysql_error();
}
```

Это гарантирует, что функция возвращает номер ошибки и ее описание, даже если подключение к базе данных было успешным, но последующий запрос оказался неудачным.

Изменение структуры таблицы

Итак, база данных создана и в ней имеется две таблицы. Эта база данных уже использовалась несколько раз, и очевидно, что она неидеальна, но вполне справляется с поставленными перед ней задачами. Предположим, что в будущем неожиданно выяснится, что база данных работает не так, как нужно, и что таблицы спроектированы неудачно. Кроме того, обнаруживаются новые данные, которые не вписываются в ожидаемый формат. Очень многие факторы могут подтолкнуть разработчика к модификации таблиц в базе данных.

Для модификации таблиц MySQL предоставляет команду `ALTER TABLE`. С ее помощью можно выполнять следующие операции:

- ☐ добавлять или удалять поля или индексы (ключевые слова `ADD` и `DROP`);
- ☐ изменять определения существующих полей (ключевые слова `ALTER`, `CHANGE` и `MODIFY`);
- ☐ переименовывать поля (`CHANGE`) или даже саму таблицу (`RENAME AS`).

Например, для переименования таблицы `test` в `tested` можно ввести следующую команду:

```
mysql> ALTER TABLE test RENAME AS tested;
Query OK, 0 rows affected (0.02 sec)
```

Ключевое слово `AS` использовать необязательно, поэтому следующая команда работает так же:

```
mysql> ALTER TABLE test RENAME tested;
Query OK, 0 rows affected (0.02 sec)
```

Добавим в таблицу `user` новое поле типа `ENUM` с именем `sex`. Данное поле можно будет использовать для записи пола пользователя:

```
mysql> ALTER TABLE user ADD sex ENUM('M', 'F') DEFAULT 'M';
Query OK, 0 rows affected (0.24 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> DESC user;
```

Field	Type	Null	Key	Default	Extra
...					
sex	enum('M', 'F')	YES		M	

10 rows in set (0.00 sec)

Новое поле добавляется в таблицу как последнее поле. Чтобы вставить новое поле между другими полями, используется ключевое слово `AFTER`. Теперь удалим только что созданное поле:

```
mysql> ALTER TABLE user DROP sex;
Query OK, 0 rows affected (0.08 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Чтобы вставить новое поле `sex` сразу после поля `username`, можно ввести следующую команду:

```
mysql> ALTER TABLE user ADD sex ENUM('M', 'F') DEFAULT 'M' AFTER username;
Query OK, 0 rows affected (0.09 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> desc user;
```

Field	Type	Null	Key	Default	Extra
...					
sex	enum('M', 'F')	YES		M	

10 rows in set (0.00 sec)

Чтобы поместить новое поле в начало списка полей, вместо ключевого слова `AFTER` необходимо использовать ключевое слово `FIRST`, так как предшествующих полей в этом случае нет.

Если сайт адресован женщинам, вероятно, значение по умолчанию для поля `sex` следует изменить с `M` на `F`:

```
mysql> ALTER TABLE user ALTER sex SET DEFAULT 'F';
Query OK, 0 rows affected (0.01 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Для полного изменения определения поля используется ключевое слово `MODIFY`:

```
mysql> ALTER TABLE user MODIFY userprofile VARCHAR(250) NOT NULL
-> DEFAULT 'No Comment';
Query OK, 0 rows affected (0.01 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Чтобы изменить имя и определение поля, можно использовать ключевое слово `CHANGE`:

```
mysql> ALTER TABLE user CHANGE userposition playerposition
VARCHAR(50) NOT NULL;
Query OK, 0 rows affected (0.01 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Аналогично можно добавлять или удалять индексы или первичные ключи:

```
mysql> ALTER TABLE user ADD INDEX (username);
Query OK, 0 rows affected (0.08 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> ALTER TABLE user DROP INDEX username;
Query OK, 0 rows affected (0.09 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Чтобы приобрести опыт работы с командой ALTER TABLE, рекомендуется поэкспериментировать с ней, в частности для начала можно восстановить структуру таблицы user.

Изменять структуру таблиц в PHP также просто, как вводить любые другие запросы с помощью функции `mysql_query()`. Предположим, что создано подключение к базе данных `sample_db`. Чтобы удалить индекс `registerdate`, необходимо использовать следующий код:

```
mysql_query("ALTER TABLE user DROP INDEX registerdate");
```

Вставка данных в таблицу

Попытаемся вставить в созданные таблицы некоторые данные. (Вставка данных с помощью PHP более подробно рассматривается в следующей главе.)

Чтобы вставить в таблицу новую запись (или несколько записей), используется SQL-команда INSERT:

```
mysql> INSERT INTO access_log VALUES ('/score.html', 'Pads', 2, NULL);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM access_log WHERE userid = 'Pads';
+-----+-----+-----+-----+
| page          | userid | visitcount | accessdate          |
+-----+-----+-----+-----+
| /score.html   | Pads   | 2          | 20040804153559      |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Чтобы вставить строковое значение, его необходимо заключить в кавычки.

Чтобы использовать этот синтаксис, нужно указать соответствующее значение для каждого поля в таблице. Для поля `accessdate` в качестве значения указывается NULL. Данное поле имеет тип `TIMESTAMP` и в нем сохраняется текущее системное время в формате `YYYYMMDDhhmmss`. Для поля типа `integer` с атрибутом `AUTO_INCREMENT` задается значение NULL или 0.

Чтобы вставить значения только в определенное подмножество полей таблицы, используется другой синтаксис:

```
mysql> INSERT INTO access_log (page, userid, visitcount)
-> VALUES ('/stats.html', 'Pads', 1);
Query OK, 1 row affected (0.00 sec)
```

В случае если данный запрос выполняется успешно, указанные поля получают перечисленные значения, тогда как остальным полям присваиваются значения по умолчанию. Например, `accessdate` — поле имеет тип `timestamp`, поэтому его значение по умолчанию равно текущему системному времени, что и требовалось:

```
mysql> SELECT * FROM access_log WHERE userid = 'Pads';
+-----+-----+-----+-----+
| page          | userid | visitcount | accessdate          |
+-----+-----+-----+-----+
```



```

|/score.html| Pads | 2 | 20040804153559 |
|/stats.html| Pads | 1 | 20040804152382 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

Экранирование кавычек

При вставке строкового значения в поле символьного или текстового типа необходимо убедиться, что все записываемые в поле кавычки экранированы (с помощью обратной косой черты). В противном случае возникнет ошибка:

```

mysql> INSERT INTO user (userprofile)
-> VALUES('I'm a rugby player.');
```

ERROR 1064: You have an error in your SQL syntax near 'm a rugby player.';
 at line 1
 (ОШИБКА 1064: ошибка SQL-синтаксиса около 'm a rugby player.';
 в строке 1)

Несмотря на то что в конце запроса стоит точка с запятой, клиент mysql выводит сообщение об ошибке, так как MySQL-сервер ожидает, что для последней одинарной кавычки будет добавлена парная одинарная кавычка. Именно поэтому в начале сообщения об ошибке стоит кавычка. Если вставить еще одну одинарную кавычку, чтобы удовлетворить ожидания сервера, то будет сгенерирована ошибка. Чтобы решить эту проблему, можно либо экранировать внутреннюю одинарную кавычку:

```
mysql> INSERT INTO user (userprofile) VALUES('I\'m a rugby player.');
```

либо заключить все строковое значение в двойные кавычки:

```
mysql> INSERT INTO user (userprofile) VALUES("I'm a rugby player.");
```

Необходимо помнить о том, что обратная косая черта (которая используется для обозначения каталога на DOS/Windows-платформе) также должна быть экранирована. В такой ситуации используется двойная обратная косая черта:

```
mysql> INSERT INTO user (userprofile) VALUES("C:\\Program Files\\PHP");
```

Если попытаться вставить новую запись, содержащую такое же значение для первичного или уникального ключа, как и у существующей записи, то возникнет ошибка:

```
mysql> INSERT INTO user (userid, userprofile)
-> VALUES('Pads', 'I\'m a rugby player.');
```

ERROR 1062: Duplicate entry 'Pads' for key 1
 (ОШИБКА 1062: Дублирующееся значение 'Pads' для ключа 1)

MySQL-сервер сообщает, что пользователь пытается вставить дублирующуюся запись, содержащую уже существующее значение для первичного ключа userid. Пользователь Pads в этой таблице уже существует. И поскольку поле userid определено как первичный ключ, дублирование записей не допускается.

Однако можно вставить новую запись, переписав существующую. Например, предположим, что были вставлены неверные значения для записи в таблице user, и вместо того чтобы удалять эту запись и вставлять ее снова, неправильную запись можно просто заменить правильной. Для этого используется команда REPLACE. Единственное отличие REPLACE от INSERT заключается в том, что если первичный ключ для новой записи дублирует существующее значение, то REPLACE переписывает существующую запись, тогда как INSERT генерирует ошибку.

```
mysql> REPLACE INTO user (userid, userprofile)
-> VALUES('Pads', 'I\'m a rugby player.');
```

Query OK, 1 row affected (0.00 sec)

Заполнение таблиц данными

Теперь можно попытаться вставлять в таблицы какие-либо записи, например, используя команду INSERT, вставить запись для пользователя Pads:

```
mysql> INSERT INTO user VALUES (
-> NULL,
-> 'Pads',
-> password('12345'),
-> 'Brian Reid',
-> 'Winger',
-> 'Stickypads@doggiesrugby.co.za',
-> 'Высококласный нападающий.');
```

Чтобы зашифровать пароль 12345, можно использовать функцию password(). Это одна из функций MySQL-сервера, которые подробнее обсуждаются в следующей главе.

В данном случае в поле usernumber записывается значение NULL. Поскольку это поле имеет атрибут AUTO_INCREMENT, номер, назначаемый каждому новому пользователю, автоматически увеличивается на единицу.

Ниже показаны данные для таблицы user. Следует учесть, что здесь не приводятся значения поля password, так как значения, сохраняемые MySQL, все равно будут зашифрованы. В этом поле для каждого пользователя можно вставить любое значение.

<i>Usernumber</i>	<i>Userid</i>	<i>Username</i>	<i>Userposition</i>	<i>Useremail</i>	<i>Userprofile</i>
1	Pads	Brian Reid	Winger	Stickypads@doggiesrugby.co.za	Высококласный нападающий
2	Nicrot	Nic Malan	Mid	therot@mywebsiteaddress.com	Неудачник
3	Spargy	Andrew Sparg	Mid	Spargy@whatyoumaycallit.com	Никогда не получит мяч от Dodge
4	Dodge	Dave Mercer	Link	davidm@contechst.com	Адмирал!
5	Mac	Murray McCallum	Winger	murray@doggiesrugby.co.za	Это не мой номер
6	Greeny	Mark Greenfield	Utility back	greeny@greenyweb.co.za	Никогда не выбрасывает мяч за линию

Следующая команда вставляет в таблицу access_log запись для пользователя Greeny:

```
mysql> INSERT INTO access_log VALUES (
-> '/penalties/index.html',
-> 'Greeny',
-> 9,
-> '20040321123155');
```

Поле accessdate в обычных обстоятельствах получает значение NULL. Таблица access_log должна содержать только несколько фиктивных записей, необходимых для последующих примеров. Разрабатываемое приложение, которое будет использовать таблицу для регистрации числа посещений фиктивного Web-сайта, будет автоматически записывать в эту таблицу данные. Это касается всех примеров вплоть до конца 11 главы.

После ввода всей необходимой информации должна получиться готовая к использованию база данных.

Резюме

В данной главе рассматривались модели хранения информации, базы данных и наиболее жизнеспособные варианты хранения больших массивов данных, а также был дан краткий обзор преимуществ систем управления реляционными базами данных (СУРБД). Кроме того, в главе обсуждались такие проблемы, как производительность, работоспособность и безопасность. В главе объясняется, почему свободно доступная СУРБД MySQL является серьезным кандидатом для применения в большинстве приложений, использующих базы данных.

Еще раз кратко опишем преимущества реляционных баз данных с клиент/серверной архитектурой (которую поддерживают не все СУРБД):

- **производительность:** нормализация баз данных значительно увеличивает производительность;
- **многопользовательская среда:** единственным ограничением, налагаемым на количество одновременно подключающихся пользователей, является тип приобретенной лицензии и/или производительность имеющегося аппаратного обеспечения;
- **доступность:** сетевая СУРБД может обрабатывать запросы “в реальном времени”, в любое время делая доступными актуальные данные; иначе говоря, при необходимости к такой базе данных можно получить доступ отовсюду и в любой момент;
- **безопасность:** жизненно важным является управление доступом к данным посредством хорошо организованной схемы обеспечения безопасности; например, система MySQL не предоставляет непосредственный доступ к каким-либо сохраненным в ней данным, обеспечивая таким образом безопасность информации на более высоком по сравнению с операционной системой уровне; пользователь, не имеющий необходимых привилегий, не может выполнять несанкционированные операции с данными;
- **надежность:** когда речь идет о программном обеспечении, степень надежности, как правило, измеряется частотой таких отказов продукта в течение срока службы, которые требуют перезапуска; например, MySQL порождает дочерние процессы для обработки одновременных запросов; когда дочерний процесс работает неверно и в конечном итоге останавливается, он редко тянет за собой весь сервер баз данных, таким образом, сводя к минимуму свое влияние на целостность данных в базе.

Подведем итог: планируя разработку целевых многопользовательских Web-приложений, в качестве сервера для хранения данных стоит рассматривать сетевую СУРБД.

В этой главе также рассматривалась установка MySQL-сервера и основы SQL — языка структурированных запросов, на котором основана система MySQL. В главе описывались некоторые основные SQL-запросы — SELECT, INSERT, REPLACE, DELETE и UPDATE, а также права доступа в MySQL.

Любой сценарий, в котором для подключения к MySQL-серверу используются PHP-функции, позволяет выбрать определенную информацию в базах данных сервера. В главе описывалось несколько способов для обработки ошибок, возникающих в MySQL-сервере; рассматривалось создание и модификация таблиц с помощью SQL-операторов, а также заполнение таблиц в базе данных информацией, которую можно будет использовать в последующих главах.

10

Получение данных от MySQL с помощью PHP

До сих пор основное внимание в книге уделялось проверке соединения с MySQL либо через клиентскую программу, либо посредством MySQL-функций PHP, а также созданию таблиц и заполнению их данными. Одним из первых SQL-операторов, которые описывались в главе 9, был базовый оператор `SELECT`. Чтобы точно определить, какие данные должна возвращать база, можно использовать множество других операторов. В этой главе описываются способы получения доступа к данным, хранящимся в MySQL-базе данных, из PHP-сценариев.

В начале следует рассмотреть PHP-функции, предназначенные для получения данных, а затем изучить методику формирования SQL-операторов `SELECT`, позволяющих получить доступ к необходимым данным, упорядоченным так, как нужно для работы приложения.

Затем будет рассмотрена возможность ограничения количества возвращаемых результатов, а также их упорядочение и группировка. Кроме того, в учебных примерах данной главы создается сценарий для просмотра пользовательских записей, позволяющий просматривать таблицы созданной в предыдущей главе базы данных.

Получение данных с помощью PHP

При использовании `SELECT`-операторов с функцией `mysql_query()` результирующее множество (которое также можно отобразить в командной строке `mysql`) передается в память, и возвращается идентификатор результата. Обычно этот идентификатор сохраняется в переменной, например, `$result`, и используется для указания результирующего множества в последующих вызовах функций.

Ранее уже было показано, как функция `mysql_fetch_row()` возвращает одну строку из результирующего множества. В простом примере из предыдущей главы используется цикл `while` для поочередного вывода каждой строки результирующего

множества, созданного оператором SHOW DATABASES. Ниже приводится усовершенствованная версия сценария show_db.php, которая распечатывает информацию из таблицы user базы данных sample_db:

```
<?php include "./common_db.inc";

$link_id = db_connect('sample_db');
$result = mysql_query("SELECT * FROM user", $link_id);

while($query_data = mysql_fetch_row($result)) {
    echo "'",$query_data[1]," - ",$query_data[4], "<br>";
}
?>
```

Сценарий возвращает следующий вывод:

```
'Nicrot' - Mid
'Spargy' - Mid
'Pads' - Winger
'Dodge' - Link
'Mac' - Winger
'Greeny' - Utility back
```

Сначала сценарий подключается к серверу с помощью функции db_connect (которая определена в подключаемом файле common_db.inc), указывая необходимую базу данных sample_db. Затем выполняется SQL-оператор "SELECT * FROM user", результирующее множество которого представляет собой все содержимое таблицы user.

В переменной \$result хранится возвращаемый идентификатор результата, с помощью которого в вызове функции mysql_fetch_row() указывается результирующее множество. Данная функция в свою очередь выбирает первую строку результата, из которой затем выводятся второе и четвертое поля (userid и userposition, соответственно). После этого внутренний указатель перемещается к следующей строке результата, а цикл while продолжается до тех пор, пока не будут выбраны все содержащиеся в таблице строки.

Достичь того же результата можно более эффективным способом:

```
$result = mysql_query("SELECT userid, userposition FROM user", $link_id);

while($query_data = mysql_fetch_row($result)) {
    echo "'",$query_data[0]," - ",$query_data[1], "<br>";
}
```

Извлекая из таблицы только те поля, которые нужны, можно сэкономить время и память.

В PHP имеется еще две функции, которые можно использовать для выборки данных: mysql_fetch_array и mysql_fetch_object(). Обе они работают в основном аналогично функции mysql_fetch_row(), единственное отличие заключается в типе возвращаемых данных. Лучше всего это можно проиллюстрировать на примере кода для выборки значений из результирующего множества.

Функция mysql_fetch_array() () возвращает из результирующего множества один ассоциативный массив, сохраняя каждое значение в элементе, имя которого соответствует имени поля:

```
while($query_data = mysql_fetch_array($result)) {
    echo "'",$query_data["userid"]," - ",
        $query_data["userposition"], "<br>";
}
```

Функция `mysql_fetch_object()` возвращает один объект из результирующего множества, сохраняя каждое значение как свойство данного объекта. Свойства именуются согласно именам полей:

```
while($query_data = mysql_fetch_object($result)) {
    echo "", $query_data->userid, "' - ",
        $query_data->userposition, "<br>";
}
```

Две эти функции особенно полезны в ситуациях, когда необходимо изменить структуру таблиц базы данных. Если не изменять имена полей, то можно не беспокоиться о порядке, в котором они появляются в таблице, потому что можно извлечь значение поля, указав его имя, а не индекс. Хотя обе функции работают медленнее, чем `mysql_fetch_row()`, они обладают одним дополнительным преимуществом — эти функции делают сценарии более читабельными (например, впоследствии в коде большого приложения можно перепутать поля, если указывать только их индексы).

Существует еще одна функция, заслуживающая внимания: `mysql_result()`, которая возвращает значение определенного поля в определенной записи. В качестве аргументов данная функция принимает, как обычно, идентификатор результата, а также номер строки и имя поля. Можно переписать приведенный выше пример так:

```
$result = mysql_query("SELECT * FROM user", $link_id);
```

```
for($i = 0; $i < mysql_num_rows($result); $i++){
    echo "", mysql_result($result, $i, "userid"), "' - ",
        mysql_result($result, $i, "userposition"), "<br>";
}
```

Можно также выводить значения в обратном порядке:

```
$result = mysql_query("SELECT * FROM user", $link_id);
```

```
for($i = mysql_num_rows($result)-1; $i >= 0; $i--){
    echo "", mysql_result($result, $i, "userid"), "' - ",
        mysql_result($result, $i, "userposition"), "<br>";
}
```

В этом случае результирующее множество будет таким:

```
'Greeny' - Utility back
'Mac' - Winger
'Dodge' - Link
'Pads' - Winger
'Spargy' - Mid
'Nicrot' - Mid
```

Имя поля также можно указать в виде целого числа, представляющего смещение поля:

```
echo "", mysql_result($result, $i, 1), "' - ",
    mysql_result($result, $i, 3), "<br>";
```

Другой способ перемещения к заданной строке данных заключается в использовании функции `mysql_data_seek()`. Она работает аналогично функции `fseek()`, которая использовалась в примерах главы 7 для навигации в файловом потоке. Эта функция в качестве аргументов принимает дескриптор результата и целое число, представляющее позицию в результирующем множестве, к которой необходимо перейти. Как можно предположить, данная функция возвращает `True` в случае успешного перемещения и `False` при переходе за границу массива. Сохраните следующий код в файле `show_seek.php` и откройте его в браузере:

```

<?php
include "./common_db.inc";

$link_id = db_connect('sample_db');
$result = mysql_query("SELECT * FROM user", $link_id);

for($i = mysql_num_rows($result)-1; $i >=0; $i--){

    mysql_data_seek($result, $i);

    $query_data = mysql_fetch_array($result);
    echo "|", $query_data["userid"], "' - ",
    $query_data["username"], "<P>";
}
?>

```

В этом разделе были рассмотрены различные способы доступа к результирующему множеству из PHP-сценариев. Эти методики помогают найти решение в различных ситуациях и вывести необходимые данные, представленные в том порядке, который требуется. Однако это еще далеко не все.

Известно, что повысить эффективность сценариев можно путем сохранения небольших размеров результирующего множества. В приведенных примерах для этого указывались необходимые поля в исходном SELECT-операторе, например:

```
$result = mysql_query("SELECT userid, username FROM user", $link_id);
```

Однако при отображении результатов этого запроса в другом порядке могут возникнуть некоторые проблемы. Допустим, что данные нужно выводить в обратном или в алфавитном порядке. Хлопот можно избежать, если упорядочить данные, поступающие в результирующее множество. На самом деле существует множество вариантов обработки данных, эффективность которых, как по времени выполнения, так и по сложности, почти наверняка будет выше, если использовать хорошо продуманные SQL-запросы, а не специальные PHP-функции. По этой причине следует вернуться к mysql-клиенту и рассмотреть возможности, предоставляемые командой SELECT.

SQL-операторы для выборки данных

Рассмотрим команду SELECT подробнее. Обратите внимание, что все описываемые здесь команды полностью применимы к предыдущим примерам с использованием PHP — таблица, отображаемая в командной строке, представляет собой просто экранированное представление результирующего множества, с которым в конечном итоге должен работать PHP-сценарий.

Серверные функции

Возьмемся к основам и попытаемся вводить SELECT-запросы, которые вообще не требуют наличия таблицы базы данных. MySQL обладает множеством весьма полезных встроенных серверных функций. Например, чтобы узнать текущее время, можно ввести следующую команду:

```

mysql> SELECT now();
+-----+
| now() |
+-----+
| 2004-08-03 16:56:03 |
+-----+
1 row in set (0.00 sec)

```

Функция `now()` возвращает текущее время для системы, на которой работает MySQL-сервер. Чтобы получить текущую дату и время по отдельности, используются функции `curdate()` и `curtime()` соответственно:

```
mysql> SELECT curdate(), curtime();
+-----+-----+
| curdate() | curtime() |
+-----+-----+
| 2004-08-03 | 16:57:19 |
+-----+-----+
1 row in set (0.00 sec)
```

Многие из встроенных функций MySQL-сервера очень похожи на PHP-функции как в отношении синтаксиса, так и в отношении решаемых задач. Тем не менее, существуют определенные тонкие различия. Например, функция MySQL-сервера `substring()` работает почти так же, как и PHP-функция `substr()`, но первая возвращает подстроку, начиная отсчет с единицы, тогда как вторая начинает отсчет с нуля:

```
mysql> SELECT substring('test',1,1);
+-----+
| substring('test',1,1) |
+-----+
| t                      |
+-----+
1 row in set (0.54 sec)
```

Эквивалентный вызов PHP-функции `substr()` будет выглядеть так:

```
$substring = substr('test', 0, 1);
```

Выбираемые поля

Ранее уже было показано, как получать информацию из таблицы базы данных. Чтобы получить содержимое поля `userid` в таблице `user`, достаточно ввести команду:

```
mysql> SELECT userid FROM user;
+-----+
| userid |
+-----+
| Dodge  |
| Greeny |
| Mac    |
| Nicrot |
| Pads   |
| Spargy |
+-----+
6 rows in set (0.05 sec)
```

Очевидно, что в таблице `user` имеется шесть записей. (Обратите внимание на то, что ни имена полей, ни имена таблиц не содержат пробелов; имена должны вводиться как одно слово, так же, как имена переменных в PHP.)

Можно одновременно извлечь данные из нескольких полей, разделяя имена полей запятыми:

```
mysql> SELECT userid, username FROM user;
+-----+-----+
| userid | username |
+-----+-----+
| Nicrot | Nic Malan |
| Spargy | Andrew Sparg |
| Pads   | Brian Reid |
+-----+-----+
```



```

+-----+-----+
| Dodge  | Dave Mercer          |
| Mac    | Murray McCallum     |
| Greeny | Mark Greenfield      |
+-----+-----+
6 rows in set (0.00 sec)

```

Все поля таблицы можно просмотреть, используя символ *:

```
mysql> SELECT * FROM user;
```

Это эквивалентно указанию каждого поля таблицы. Однако при написании приложений рекомендуется избегать такой формы записи, даже когда действительно требуется извлечь значения всех полей таблицы. Если позднее структура таблицы будет изменена, и добавятся новые поля или изменится их порядок, то может оказаться, что приложение получает совсем не то результирующее множество, которое ожидает, а это может привести к непредсказуемым результатам. Ниже описываются различные способы применения SQL для более четкого определения результирующего множества.

Ограничение количества возвращаемых результатов

Сузить массив извлекаемых данных можно с помощью SQL-предложений LIMIT и WHERE. Предположим, что необходимо заставить MySQL-сервер извлечь только несколько записей из таблицы. Для этого используется предложение LIMIT. Например, чтобы выбрать первые две строки соответствующих данных, начиная с записи 0, можно ввести следующую команду:

```

mysql> SELECT userid, username FROM user LIMIT 0, 2;
+-----+-----+
| userid | username          |
+-----+-----+
| Nicrot | Nic Malan         |
| Spargy | Andrew Sparg      |
+-----+-----+
2 rows in set (0.00 sec)

```

Если начальная запись не указана, то по умолчанию используется 0, поэтому следующий запрос достигает той же цели:

```
mysql> SELECT userid, username FROM user LIMIT 2;
```

Предложение LIMIT всегда записывается в конце запроса.

Предложение WHERE используется для выборочного получения строк данных, соответствующих определенным условиям. Например, чтобы получить данные об игроках, занимающих позицию Mid, можно ввести следующую команду:

```

mysql> SELECT userid, username FROM user
-> WHERE userposition = 'Mid';
+-----+-----+
| userid | username          |
+-----+-----+
| Nicrot | Nic Malan         |
| Spargy | Andrew Sparg      |
+-----+-----+
2 rows in set (0.00 sec)

```

Знак равенства используется для указания условия (в данном примере Mid). Не считая оператора равенства, который эквивалентен PHP-оператору ==, операторы сравнения в предложениях WHERE выглядят почти так же, как операторы сравнения в PHP. В приведенной ниже таблице показаны операторы сравнения в MySQL.

Оператор сравнения	Описание
=	равно
!= or <>	не равно
<	меньше
>	больше
<=	меньше или равно
>=	больше или равно

Существует также нуль-безопасный оператор сравнения (`<=>`), который позволяет использовать значения равные или не равные NULL. Например:

```
mysql> Select 1 <=> NULL;
+-----+
| 1 = NULL |
+-----+
|        NULL        |
+-----+
```

С помощью логических операторов AND и OR можно указать множество условий:

```
mysql> SELECT usernumber, userid, userposition FROM user
-> WHERE userposition = 'Utility Back'
-> OR userposition = 'Mid'
-> AND usernumber < 5;
```

```
+-----+-----+-----+
| usernumber | userid | userposition |
+-----+-----+-----+
|          2 | Nicrot | Mid          |
|          3 | Spargy | Mid          |
|          6 | Greeny | Utility back |
+-----+-----+-----+
3 rows in set (0.01 sec)
```

Однако полученные данные не совсем верные. Требовалось перечислить только строки, в которых значения поля `usernumber` не превышают 5, но последняя строка в результирующем множестве имеет значение `usernumber` равное 6. В чем причина? Все очень просто — оператор AND в данном случае применяется только к условию `userposition='Mid'`. Когда сервер выбирает записи об игроках `Utility back`, он не учитывает последнее условие, поэтому оператор AND не имеет смысла. Чтобы сгруппировать условия и определить приоритет операций сравнения, необходимо использовать круглые скобки, так же как и в PHP:

```
mysql> SELECT usernumber, userid, userposition FROM user
-> WHERE (userposition = 'Utility Back'
-> OR userposition = 'Mid')
-> AND usernumber < 5;
```

```
+-----+-----+-----+
| usernumber | userid | userposition |
+-----+-----+-----+
|          2 | Nicrot | Mid          |
|          3 | Spargy | Mid          |
+-----+-----+-----+
2 rows in set (0.47 sec)
```

Более подробная информация о MySQL-операторах приведена на Web-странице www.mysql.com/doc/en/Functions.html.

Упорядочение результатов

Чтобы отсортировать полученные строки данных по определенному полю, можно использовать предложение ORDER BY:

```
mysql> SELECT usernumber, userid, userposition FROM user
->     WHERE usernumber < 5 ORDER BY userid;
```

usernumber	userid	userposition
4	Dodge	Link
2	Nicrot	Mid
1	Pads	Winger
3	Spargy	Mid

4 rows in set (0.05 sec)

При сортировке символьных полей MySQL упорядочивает их согласно ASCII-значениям, поэтому все прописные буквы помещаются перед строчными.

Предложение ORDER BY по умолчанию сортирует полученные значения в порядке по возрастанию, поэтому в предыдущем примере строки сортируются в алфавитном порядке по полю userid.

Чтобы отсортировать значения в обратном порядке, необходимо использовать ключевое слово DESC:

```
mysql> SELECT userid FROM user ORDER BY userid DESC;
```

userid
Spargy
Pads
Nicrot
Mac
Greeny
Dodge

6 rows in set (0.03 sec)

Можно сортировать извлекаемые строки по нескольким полям, разделяя имена полей запятыми. В следующем примере данные сортируются по полю userposition (по возрастанию) и по полю userid (по убыванию):

```
mysql> SELECT userid, username, userposition FROM user
->     WHERE userposition = 'Winger'
->     OR userposition = 'Mid'
->     ORDER BY userposition, userid DESC;
```

userid	username	userposition
Spargy	Andrew Sparg	Mid
Nicrot	Nic Malan	Mid
Pads	Brian Reid	Winger
Mac	Murray McCallum	Winger

4 rows in set (0.00 sec)

Не забывайте о том, что ORDER BY лучше всего работает с индексированными полями, поскольку такие поля уже отсортированы.

Сравнение с образцом

С помощью операторов LIKE и NOT LIKE можно создавать образцы для выборки данных. Для этого используются следующие специальные символы:

- ❑ % соответствует любой возможной строке, включая пустые строки (как * в командах оболочки Unix или DOS);
- ❑ _ соответствует любому одному символу (как ? в командах оболочки Unix или DOS).

Чтобы выбрать все строки, в которых значение поля useremail заканчивается подстрокой doggies_rugby.com, можно ввести следующий запрос:

```
mysql> SELECT userid, username, userposition, useremail FROM user
-> WHERE useremail LIKE '%doggiesrugby.co.za'
-> ORDER BY username;
```

userid	username	userposition	useremail
Pads	Brian Reid	Winger	tickyPads@doggiesrugby.co.za
Mac	Murray McCallum	Winger	murray@doggiesrugby.co.za

2 rows in set (0.49 sec)

Значения в полях типа TEXT нечувствительны к регистру символов. Если при сравнении необходимо учитывать регистр символов, то следует использовать поле типа BLOB.

Образец __c соответствует значению Mac в поле userid, тогда как образец __ соответствует любой строке, состоящей в точности из двух символов. Вне зависимости от того, сколько символов _ имеется в образце, mysql пытается найти записи, имеющие точное количество символов:

```
mysql> SELECT userid FROM user WHERE userid LIKE '__c';
+-----+
| userid |
+-----+
| Mac    |
+-----+
1 row in set (0.00 sec)
```

Работа оператора NOT LIKE диаметрально противоположна. Ниже выбираются все записи, в которых значение поля userid не заканчивается на букву y:

```
mysql> SELECT userid FROM user
-> WHERE userid NOT LIKE '%y';
+-----+
| userid |
+-----+
| Dodge  |
| Mac    |
| Nicrot |
| Pads   |
+-----+
4 rows in set (0.00 sec)
```

Кроме того, в MySQL можно использовать регулярные выражения. Однако эта тема выходит за рамки данной главы, более подробно регулярные выражения в MySQL описаны в документации на Web-странице www.mysql.com/doc/en/Pattern_matching.html.

Получение итоговых данных

Каким же образом MySQL-сервер выводит сводную информацию по данным таблицы? В MySQL имеется несколько функций группировки, которые выводят не все строки запроса, а только итоговые результаты:

- ☐ `sum()` выводит сумму значений заданного поля;
- ☐ `max()` выводит максимальное число в заданном поле;
- ☐ `min()` выводит минимальное число в заданном поле;
- ☐ `avg()` выводит среднее значение по заданному полю;
- ☐ `count()` выводит количество возвращаемых строк.

Например, используя функции `min()` и `max()`, можно получить минимальные и максимальные значения целочисленных полей:

```
mysql> SELECT min(usernumber), max(usernumber) FROM user;
+-----+-----+
| min(usernumber) | max(usernumber) |
+-----+-----+
| 1               | 6               |
+-----+-----+
1 row in set (0.00 sec)
```

Количество возвращаемых строк можно получить с помощью функции `count()`. Ее можно использовать двумя способами:

- ☐ `count(имя_поля)` подсчитывает только строки, в которых значение заданного поля не равно NULL;
- ☐ `count(*)` подсчитывает все строки результирующего множества.

В следующем запросе с помощью функции `count(*)` определяется количество игроков, имеющих номера (значения поля `usernumber`) меньше 3:

```
mysql> SELECT count(*) FROM user WHERE usernuber < 3;
+-----+
| count(*) |
+-----+
| 2        |
+-----+
1 row in set (0.00 sec)
```

Более сложные выборки

Существует множество ключевых слов и запросов, которые можно использовать в MySQL для получения данных. Предположим, например, что необходимо получить список игроков, у которых имена страниц в таблице `access_log` заканчиваются на `.html`. Такую информацию может дать простой перечень посещенных страниц, однако в нем, скорее всего, будут содержаться дублирующиеся записи — особенно в более крупной базе данных. На самом деле нужен список *уникальных* значений — иначе говоря, без дублирования записей. Не удивительно, что в SQL имеется специальное ключевое слово `DISTINCT`, которое позволяет формировать подобные списки. Данное ключевое слово используется следующим образом:

```
mysql> SELECT DISTINCT userid FROM access_log WHERE page LIKE '%.html'
                                ORDER BY userid;
+-----+
| userid |
+-----+
| Brian  |
| Greeny |
+-----+
```

```

| Nicrot |
| Pads   |
+-----+
4 rows in set (0.00 sec)

```

Можно применять функции группировки (`min()`, `max()`, `count()` и др.) для определенных групп записей, а не для всей базы данных. Например, для того чтобы подсчитать количество человек, которые играют в определенной позиции, можно сгруппировать возвращаемые строки с помощью предложения `GROUP BY`:

```

mysql> SELECT userposition, count(*) FROM user GROUP BY userposition
-> ORDER BY userposition DESC;
+-----+-----+
| userposition | count(*) |
+-----+-----+
| Winger       | 2        |
| Utility back | 1        |
| Mid          | 2        |
| Link         | 1        |
+-----+-----+
4 rows in set (0.00 sec)

```

Если использовать в одном запросе имена полей и функции группировки и не включить в запрос предложение `GROUP BY`, то будет сгенерирована ошибка:

```

mysql> SELECT userposition, count(*) FROM user ORDER BY userposition;
ERROR 1140: Mixing of GROUP columns (MIN(),MAX(),COUNT()...) with no GROUP
columns is illegal if there is no GROUP BY clause
(ОШИБКА 1140: Смешивание GROUP-столбцов (MIN(),MAX(),COUNT() и т.д.)
с не GROUP-столбцами в случае отсутствия предложения GROUP BY не допускается)

```

Необходимо помнить о том, что предложение `ORDER BY` должно записываться в запросе последним (если запрос не ограничивается предложением `LIMIT`, которое должно находиться в конце запроса), а предложение `GROUP BY` должно ему предшествовать.

Предположим, что последнюю таблицу требуется отсортировать по значениям во втором столбце, т.е. отсортировать ее строки по количеству игроков в каждой позиции. Использовать `count(*)` в предложении `ORDER BY` нельзя, как же быть? Для таких ситуаций в MySQL предусмотрено решение: псевдонимы. *Псевдоним* (*alias*) фактически представляет собой новое имя для выражения, поля или даже таблицы. При указании в `SELECT`-операторе поля (или выражения) перед ним можно задать присваиваемое ему обозначение: `AS имя_псевдонима`.

После этого результат вызова `count(*)` показывается в результирующем множестве как поле с именем `num_users`. Более того, псевдоним можно интерпретировать как обычное поле и использовать его в предложении `ORDER BY`:

```

mysql> SELECT userposition, count(*) AS num_users FROM user
-> GROUP BY userposition ORDER BY num_users;
+-----+-----+
| userposition | num_users |
+-----+-----+
| Link         | 1         |
| Utility back | 1         |
| Mid          | 2         |
| Winger       | 2         |
+-----+-----+
4 rows in set (0.00 sec)

```

Например, чтобы получить список позиций, в которых играет не менее двух игроков, можно было бы использовать следующую команду:

```

mysql> SELECT userposition, count(*) AS num_users FROM user
-> WHERE num_users > 1 GROUP BY userposition;
ERROR 1054: Unknown column 'num_users' in 'where clause'
(ОШИБКА 1054: Неизвестное имя поля 'num_users' в предложении 'where')

```

Однако возникает проблема — имя `num_users` не может быть определено до тех пор, пока не будет указано, как группировать записи. В то же время необходимо ввести предложение `WHERE` до предложения `GROUP BY`, иначе возникнет синтаксическая ошибка. Решение в такой ситуации заключается в использовании предложения `HAVING`, которое позволяет указывать условие для столбца или псевдонима в конце оператора `SELECT`. Предложение `HAVING` используется так же, как и `WHERE` за исключением того, что оно помещается *после* предложения `GROUP BY`. Сервер выполняет данный запрос как и раньше, но прежде чем вернуть результирующее множество, он отбрасывает результаты, не соответствующие заданному условию:

```
mysql> SELECT userposition, count(*) AS num_users FROM user
-> GROUP BY userposition HAVING num_users > 1;
+-----+-----+
| userposition | num_users |
+-----+-----+
| Mid         | 2         |
| Winger      | 2         |
+-----+-----+
2 rows in set (0.00 sec)
```

Как извлечь данные из двух таблиц? Необходимо просто указать имена обеих таблиц в предложении `FROM`:

```
mysql> SELECT username, page FROM user, access_log
+-----+-----+
| username | page |
+-----+-----+
| Nic Malan | /score.html |
| Andrew Sparg | /score.html |
| Brian Reid | /score.html |
| Dave Mercer | /score.html |
| Murray McCallum | /score.html |
| Mark Greenfield | /score.html |
| Nic Malan | /stats.html |
| Andrew Sparg | /stats.html |
| Brian Reid | /stats.html |
| Dave Mercer | /stats.html |
...
...
| Mark Greenfield | /log.html |
| Nic Malan | /penalties.html |
| Andrew Sparg | /penalties.html |
| Brian Reid | /penalties.html |
| Dave Mercer | /penalties.html |
| Murray McCallum | /penalties.html |
| Mark Greenfield | /penalties.html |
+-----+-----+
60 rows in set (0.00 sec)
```

Отношения между двумя таблицами не были определены, поэтому данный запрос возвращает все возможные комбинации значений. Однако MySQL-сервер верно определяет, какой таблице принадлежит каждое поле. Что произойдет, если использовать имена полей, которые имеются в обеих таблицах?

```
mysql> SELECT userid, page from user, access_log;
ERROR 1052: Column: 'userid' in field list is ambiguous
(ОШИБКА 1052: Неоднозначное имя 'userid' в перечне полей)
```

Сообщение об ошибке вполне понятно. В данном случае использование имени поля `userid` является неоднозначным, потому что оно соответствует любому из двух полей. Человеку ясно, что оба поля ссылаются на одни и те же данные, но сервер считает эти

поля полностью независимыми друг от друга. Устранить неоднозначность можно путем явного указания таблицы, из которой требуется выбрать значения данного поля. Для этого используется следующий синтаксис:

```
mysql> SELECT user.userid, page FROM user, access_log;
```

userid	page
Dodge	/score.html
Greeny	/score.html
Mac	/score.html
Nicrot	/score.html
Pads	/score.html
Spargy	/score.html
...	
Spargy	/log.html
Dodge	/penalties.html
Greeny	/penalties.html
Mac	/penalties.html
Nicrot	/penalties.html
Pads	/penalties.html
Spargy	/penalties.html

60 rows in set (0.01 sec)

Каждая запись в таблице `access_log` должна соответствовать одной записи в таблице `user` — база данных должна работать именно так. Это действительно имеет смысл: посещения страниц записываются только для зарегистрированных пользователей, поэтому любой пользователь, записанный в таблице `access_log`, должен быть зарегистрированным пользователем. В данном случае поле `userid` является общим для обеих таблиц. Предположим, что требуется узнать, какие страницы просматриваются и кем именно. Теперь, когда известно, как различить два поля с именем `userid`, можно приравнять значения этих полей в предложении `WHERE` и таким образом сузить результирующее множество:

```
mysql> SELECT user.userid, page FROM user, access_log
-> WHERE user.userid = access_log.userid;
```

userid	page
Dodge	/score.html
Dodge	/stats.html
Greeny	/log.html
Greeny	/penalties.html
Mac	/score.html
Nicrot	/log.html
Nicrot	/refs.html
Pads	/score.html

8 rows in set (0.00 sec)

В таких ситуациях отчетливо видна эффективность реляционных баз данных. Созданная связь между записями данных в двух таблицах называется *объединением* (*join*) — результирующее множество создается на основе объединения записей из одной таблицы с записями из другой таблицы. В последнем запросе поле `userid` послужило ключом для объединения таблиц `user` и `access_log`; т.е. поле `userid` использовалось для создания явной взаимосвязи между записями в двух таблицах. Ключи требуются для операций объединения, так как они связывают таблицы и устраняют необходимость повторения данных в каждой таблице.

По договору между издательством **"Вильямс"** и Интернет-Магазином "Books.Ru - Книги России" единственный легальный способ получения данного файла с книгой **“PHP 5 для начинающих ” (ISBN 5-8459-1039-0)** – покупка в Интернет-магазине "Books.Ru - Книги России".

Если вы получили данный файл каким-либо другим образом, вы нарушили законодательство об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству **"Вильямс"** где именно вы получили данный файл.

Существует множество других более сложных операций объединения, но та, которая рассматривалась выше, в большинстве ситуаций является более чем достаточной. На сайте MySQL можно найти справочные сведения по более сложным запросам и операциям объединения. Однако чтобы развить навыки использования SQL даже на самом начальном уровне, необходимо придерживаться правила: практика, практика и еще раз практика. Используйте mysql-клиент для экспериментирования с запросами с различными типами полей — в конечном счете это единственный способ оценить возможности и сложности SQL-синтаксиса.

Практический пример сценария

Используя полученные навыки, можно создать сценарий для просмотра регистрационных записей (назовем его `userviewer.php`), который использует таблицы базы данных `sample_db` и выполняет следующие функции:

- ❑ подключается к базе данных `sample_db`, в которой находятся таблицы `user` и `access_log`;
- ❑ отображает перечень зарегистрированных пользователей и навигационные ссылки, дающие администратору возможность перемещаться вперед и назад;
- ❑ отображает ссылки с именами полей, которые можно использовать для сортировки списка пользователей по определенному полю в порядке по возрастанию или по убыванию;
- ❑ для каждого пользователя отображает ссылку на новое окно, в котором представляется более подробная информация по данному пользователю и записи его журнала доступа.

На рис. 10.1 показан внешний вид интерфейса для этого сценария.

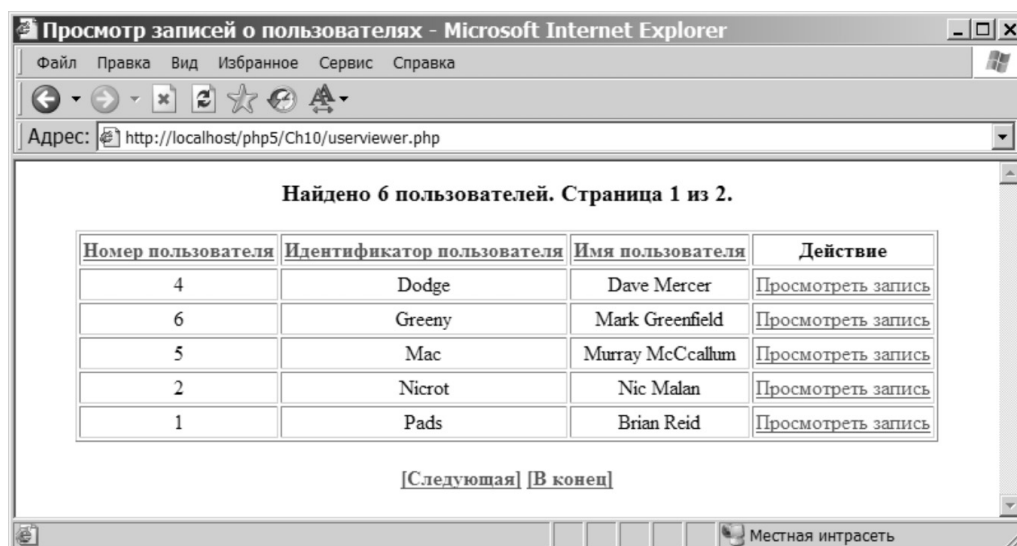


Рис. 10.1.

Для создания сценария просмотра базы данных потребуется два отдельных файла. Глобальные переменные и часто используемые функции помещаются в подключаемый файл `common_db.inc`, а бизнес-логика реализована в файле `userviewer.php`. Предполагается, что существует база данных `sample_db`, которая содержит таблицы `user` и `access_log`, причем в обеих таблицах имеются соответствующие данные.

Содержимое файла `common_db.inc`

В последующих разделах рассматривается каждый блок кода в файле `common_db.inc`.

Для краткости код функций `db_connect()` и `sql_error()` здесь не описывается. Они ничем не отличаются от одноименных функций, описанных в предыдущей главе.

Глобальные переменные

В данном сценарии придется использовать несколько переменных. Их объявления можно поместить в файл `common_db.inc` и обращаться к ним из сценария `userviewer` как к глобальным переменным (некоторые переменные, возможно, следует изменить, чтобы отразить параметры конкретной системы):

```
<?php
$dbhost = 'localhost';
$dbusername = 'phpuser';
$dbuserpassword = 'phppass';
$default_dbname = 'sample_db';
```

Определим количество записей отображаемых на странице по умолчанию (5):

```
$records_per_page = 5;
```

Укажем имена таблиц (`user` и `access_log`):

```
$user_tablename = 'user';
$access_log_tablename = 'access_log';
```

Определим переменные для хранения номера и описания ошибки:

```
$MYSQL_ERRNO = "";
$MYSQL_ERROR = "";
```

Зададим размер нового окна браузера:

```
$new_win_width = 600;
$new_win_height = 400;
```

Функция `html_header()`

Код функции `html_header()` представляет собой начало HTML-страницы и определение JavaScript-функции `open_window()`, которую можно вызывать для открытия нового окна при отображении подробной информации о пользователе:

```
function html_header() {
    global $new_win_width, $new_win_height;
    ?>
    <HTML>
    <HEAD>
    <SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
    <!--
```

```

function open_window(url)    {
    var NEW_WIN = null;
    NEW_WIN = window.open ("", "RecordViewer",
        "toolbar=no,width="+
        "<?php echo $new_win_width ?>"+
        ",height="+<?php echo $new_win_height?>"+
        ",directories=no,status=no"+
        ",scrollbars=yes,resize=no,menubar=no");
    NEW_WIN.location.href = url;
}
//-->
</SCRIPT>
<TITLE>Сценарий просмотра базы данных</TITLE>
</HEAD>
<BODY>
<?php
}

```

Функция `html_footer()`

Функция `html_footer()` завершает создание HTML-страницы (работа функций `db_connect()` и `sql_error()` уже рассматривалась):

```

function html_footer() {
    ?>
    </BODY>
    </HTML>
    <?php
}

function db_connect($dbname="") {
    ...
}

function sql_error() {
    ...
}

```

Функция `error_message()`

Функция `error_message()` сообщает пользователю о возникновении ошибки, используя для этого JavaScript-метод `alert()`:

```

function error_message($msg) {
    html_header();
    echo "<SCRIPT>alert(\"Error: $msg\ ");history.go(-1)</SCRIPT>";
    html_footer();
    exit;
}
?>

```

Теперь рассмотрим рабочую часть приложения.

Содержимое файла `userviewer.php`

Сценарий `userviewer.php` использует подключаемый файл `common_db.inc`. В данном случае этот файл подключается из текущего каталога:

```

<?php
include "../common_db.inc";

```

Следует отметить, что при разработке реальных приложений не рекомендуется включать имя базы данных и пароль к ней в текст Web-сценария. Эти данные желательно помещать в отдельный файл и хранить этот файл за пределами корневого каталога Web-сервера.

Большая часть кода, используемого для реализации данного приложения, сосредоточена в двух функциях, которые собирают и отображают информацию базы данных. Рассмотрим эти функции подробнее.

Функция `list_records()`

Функция `list_records()` отображает список зарегистрированных пользователей, а также навигационные гиперссылки и ссылку для просмотра записи базы данных, которая вызывает функцию `view_record()`. Функция начинает работу с подключения к базе данных, заданной по умолчанию, и получает общее число зарегистрированных пользователей. Если зарегистрированных пользователей нет, то отображается соответствующее сообщение об ошибке.

```
function list_records()
{
    global $default_dbname, $user_tablename;
    global $records_per_page;
    $PHP_SELF = "userviewer.php";
    $link_id = db_connect($default_dbname);
    if(!$link_id) error_message(sql_error());
    $query = "SELECT count(*) FROM $user_tablename";
    $result = mysql_query($query);
    if(!$result) error_message(sql_error());
    $query_data = mysql_fetch_row($result);
    $total_num_user = $query_data[0];
    if(!$total_num_user) error_message('Нет зарегистрированных пользователей!');
```

Глобальная переменная `$cur_page`, номер текущей страницы, содержит ключ для навигации по списку. Если данные списка занимают больше одной страницы, то они разделяются на несколько страниц. Размер страницы определяется глобальной переменной `$records_per_page`. Общее количество страниц вычисляется как результат деления общего числа пользователей (`$total_num_user`) на длину страницы (`$records_per_page`) и округляется до ближайшего большего целого значения с помощью функции `ceil()`.

Номер страницы перед его выводом в браузер необходимо увеличить на единицу, так как в глобальной переменной `$cur_page` первым номером будет 0, а нумерация страниц в браузере должна начинаться с 1. Переменная `$record` определена как верхняя граница для предложения `LIMIT`, которое используется в SQL-запросе. Переменная `$cur_page` инициализируется со значением, которое отправляется, как только пользователь щелкнет на навигационной ссылке. Если значения нет, то переменной присваивается 0:

```
if(empty($_GET['next_page'])) {
    $_GET['next_page'] = 0;
}
$cur_page = $_GET['next_page'];
$page_num = $cur_page + 1;
$record = $cur_page * $records_per_page + 5;
$total_num_page = $last_page_num = ceil($total_num_user/$records_per_page);

html_header();
```

```
echo "<CENTER><H3>В базе найдено $total_num_user пользователей. Страница
$page_num из $last_page_num.</H3></CENTER>\n";
```

Теперь следует создать SQL-запрос, который будет извлекать данные из базы и организовывать их так, как это необходимо. Способ формирования запроса на самом деле зависит от ссылки, по которой щелкает пользователь. Возникает вопрос: как сформировать SQL-запрос динамически?

Для этого следует воспользоваться специальным PHP-массивом `$_GET`. Если информация для данного запроса отправляется в URL, то ее можно получить и использовать. Необходимо помнить, что при первом посещении страницы никаких значений в массиве `$_GET` не будет, поэтому первое, что необходимо сделать, — это присвоить всем переменным значения по умолчанию. Значение по умолчанию для сортировки (предложения `ORDER BY` в запросе) следует установить равным `userid`, по умолчанию порядок сортировки по возрастанию, а номер страницы по умолчанию равным 1 (напомним, что это влечет за собой присвоение переменной `$cur_page` значения 0).

Посетитель страницы может выбрать порядок сортировки, щелкнув на имени поля в заголовке таблицы с перечнем записей. Эта функция основана на изменении значения порядка сортировки (переменная `$sort_order`) до вывода этого значения в HTML-ссылку. Обратимся еще раз к рис. 10.1. Ниже таблицы располагаются ссылки Следующая и В конец (а также В начало и Предыдущая). Эти ссылки не изменяют порядок сортировки списка, они просто позволяют перемещаться по таблице, поэтому для них следует создать отдельную переменную порядка сортировки (`$org_sort_order`):

```
if (empty($_GET['order_by'])) {
    $_GET['order_by'] = 'userid';
} $order_by = $_GET['order_by'];

if (empty($_GET['sort_order'])) {
    $_GET['sort_order'] = 'ASC';
    $sort_order = 'ASC';
}

if ($_GET['sort_order'] == 'ASC') {
    $sort_order = 'DESC';
    $org_sort_order = 'ASC';
} else {
    $sort_order = 'ASC';
    $org_sort_order = 'DESC';
}
```

Затем необходимо создать и сохранить в переменной `$limit_str` предложение `LIMIT`, используя переменные `$cur_page`, `$records_per_page` и `$record`.

```
$limit_str = "LIMIT ". $cur_page * $records_per_page . ", $record";
```

Окончательный запрос формируется путем передачи значений, полученных из `$_GET`-массива, в только что созданную переменную `$limit_str`:

```
$query = "SELECT usernumber, userid, username FROM $user_tablename ORDER BY
        $_GET[order_by] $_GET[sort_order]
        $limit_str ";

$result = mysql_query($query);    if (!$result) {
    error_message(sql_error());
}

?>
```

Теперь следует создать HTML-таблицу для вывода списка:

```
<DIV ALIGN="CENTER">
<TABLE BORDER="1" WIDTH="90%" CELLPADDING="2">

  <TR>
    <TH WIDTH="25%" NOWRAP>
      <A HREF="<?php echo "$PHP_SELF?action=list_records&
                                sort_order=$sort_order&
                                order_by=username"; ?>">
        Номер пользователя
      </A>
    </TH>
    <TH WIDTH="25%" NOWRAP>
      <A HREF="<?php echo "$PHP_SELF?action=list_records&
                                sort_order=$sort_order&
                                order_by=userid"; ?>">
        Идентификатор пользователя
      </A>
    </TH>
    <TH WIDTH="25%" NOWRAP>
      <A HREF="<?php echo "$PHP_SELF?action=list_records&
                                sort_order=$sort_order&
                                order_by=username"; ?>">
        Имя пользователя
      </A>
    </TH>
    <TH WIDTH="25%" NOWRAP>Действие</TH>
  </TR>
<?php
```

Ссылки в заголовках таблицы являются неотъемлемой частью данной программы. Каждая ссылка указывает на страницу `userviewer` (`$PHP_SELF`) и передает значения, которые записываются в массив `$_GET`, который в свою очередь будет использоваться в сценарии при следующем запуске, т.е. когда пользователь щелкнет на ссылке. Обратите внимание, что значением параметра `action` в URL является строка `list_records`. Это важно, потому что в конце программы используется оператор `switch`, позволяющий определить, какая функция из файла `userviewer` будет использоваться.

Затем результирующее множество обрабатывается с помощью цикла `while`, в результате чего формируются строки HTML-таблицы. Стоит отметить, что последняя строка содержит вызов JavaScript-функции `open_window`, определенной в файле `common_db.inc`. Кроме того, параметру `action` присваивается значение `view_record`. Это означает, что вместо функции `list_records()` будет вызвана функция `view_record()`, которая описывается в следующем разделе. Пример кода:

```
while($query_data = mysql_fetch_array($result)) {
  $username = $query_data["username"];
  $userid = $query_data["userid"];
  $usernumber = $query_data["usernumber"];
  echo "<TR>\n";
  echo "<TD WIDTH=\"25%\" ALIGN=\"CENTER\">$username</TD>\n";
  echo "<TD WIDTH=\"25%\" ALIGN=\"CENTER\">$userid</TD>\n";
  echo "<TD WIDTH=\"25%\" ALIGN=\"CENTER\">$username</TD>\n";
  echo "<TD WIDTH=\"25%\" ALIGN=\"CENTER\">
    <A HREF=\"javascript:open_window('$PHP_SELF?action=view_record&
    userid=$userid');\">Просмотреть запись</A></TD>\n";
  echo "</TR>\n";
}
```

```

    }
?>
</TABLE>
</DIV>
<?php
    echo "<BR>\n";
    echo "<STRONG><CENTER>";

```

Наконец, нужно создать навигационные ссылки на основе номера текущей страницы и общего количества страниц. Поскольку значения переменной `$cur_page` начинаются с нуля, для хранения номера текущей страницы используется другая переменная — `$page_num`. Если номер текущей страницы больше 1, то необходимо вывести ссылки В начало и Предыдущая, которые позволят пользователю перейти к первой и предыдущей странице соответственно:

```

if ($page_num > 1) {
    $prev_page = $cur_page - 1;

    echo "<A HREF=\"\$PHP_SELF?action=list_records&
        ort_order=$org_sort_order&order_by=$order_by&next_page=0\">[В начало]
        </A>";

    echo "<A HREF=\"\$PHP_SELF?action=list_records&sort_order=$org_sort_order
        &order_by=$order_by&next_page=$prev_page\">[Предыдущая] </A> ";
}

```

Аналогично, ссылки Следующая и В конец отображаются, если номер текущей страницы меньше, чем общее количество страниц:

```

if ($page_num < $total_num_page) {
    $next_page = $cur_page + 1;
    $last_page = $total_num_page - 1;

    echo "<A HREF=\"\$PHP_SELF?action=list_records&sort_order=$org_sort_order
        &order_by=$order_by&next_page=$next_page\">[Следующая] </A> ";

    echo "<A HREF=\"\$PHP_SELF?action=list_records&sort_order=$org_sort_order&
        order_by=$order_by&next_page=$last_page\">[В конец] </A> ";
}

```

```
echo "</STRONG></CENTER>";
```

Определение функции `list_records()` завершается вызовом функции `html_footer()`, которая определена в файле `common_db.inc`:

```

html_footer();
}

```

Теперь сценарий можно запустить, не используя ссылок Просмотреть запись с правой стороны таблицы. Чтобы эти ссылки работали, необходимо создать функцию `view_record()`.

Функция `view_record()`

Функция `view_record()` отображает подробную информацию об определенном пользователе и данные из его журнала посещений.

Когда посетитель нажимает на ссылку Просмотреть запись, информация о выбранном пользователе отображается в новом окне Web-браузера, которое открывается с помощью JavaScript-функции `open_window()`. Эта функция в свою очередь вызывает

сценарий со значением переменной `$action`, приводящим к выполнению функции `view_record()`.

Сначала из таблицы `user` выбирается вся информация об указанном пользователе, а также все записи из журнала посещений (из таблицы `access_log`) этого пользователя. Если пользователь не посещал протоколируемых Web-страниц, то отображается соответствующее предупреждение и сценарий завершает работу. В данном случае нет необходимости проверять массив `$_GET`, так как функция `view_record()` вызывается только со страницы `userviewer`; иными словами, просмотреть страницу записей, не щелкнув предварительно на ссылке, которая устанавливает `$_GET`-переменные, невозможно:

```
function view_record() {
    global $default_dbname, $user_tablename, $access_log_tablename;
    $PHP_SELF = $_SERVER['PHP_SELF'];
    $userid = $_GET['userid'];

    if(empty($userid)){
        error_message('Выберите пользователя!');
    }

    $link_id = db_connect($default_dbname);

    if(!$link_id){
        error_message(sql_error());
    }

    $query = "SELECT * FROM $user_tablename WHERE userid = '$userid'";
    $result = mysql_query($query);

    if(!$result){
        error_message(sql_error());
    }

    $query_data = mysql_fetch_array($result);
    $usernumber = $query_data["usernumber"];
    $userid = $query_data["userid"];
    $username = $query_data["username"];
    $userposition = $query_data["userposition"];
    $useremail = $query_data["useremail"];
    $userprofile = $query_data["userprofile"];
```

Затем отображается информация, относящаяся к выбранному пользователю:

```
html_header();
echo "<CENTER><H3>
    Запись для пользователя №.$usernumber - $userid($username)
    </H3></CENTER>";

?>
<DIV ALIGN="CENTER">
<TABLE BORDER="1" WIDTH="90%" CELLPADDING="2">
    <TR>
        <TH WIDTH="40%">position</TH>
        <TD WIDTH="60%"><?php echo $userposition ?></TD>
    </TR>
    <TR>
        <TH WIDTH="40%">Email</TH>
        <TD WIDTH="60%"><?php echo "<A
    HREF=\"mailto:$useremail\">$useremail</A>\"
        ; ?></TD>
    </TR>
</TR>
```

```

        <TH WIDTH="40%">Профиль</TH>
        <TD WIDTH="60%"><?php echo $userprofile ?></TD>
    </TR>

</TABLE>
</DIV>
<?php
    echo "<HR SIZE=\"2\" WIDTH=\"90%\">\n";

```

После чего выводится вторая таблица, в которой перечислены записи журнала посещений для данного пользователя:

```

$query = "SELECT page, visitcount, accessdate FROM $access_log_tablename
        WHERE userid = '$userid'";

$result = mysql_query($query);
if(!$result){

    error_message(sql_error());

}

```

Функция `mysql_num_rows()` возвращает 0, если в таблице `access_log` нет записей, содержащих информации о страницах, посещенных этим пользователем:

```

if(!mysql_num_rows($result)){
    echo "<CENTER>Для пользователя $userid ($username) нет записей в журнале
        посещений.</CENTER>";
} else {
    echo "<CENTER>Записи журнала посещений для пользователя $userid
        ($username).</CENTER>";
?>
<DIV ALIGN="CENTER">
<TABLE BORDER="1" WIDTH="90%" CELLPADDING="2">
    <TR>
        <TH WIDTH="40%" NOWRAP>Web-страница</TH>
        <TH WIDTH="20%" NOWRAP>Количество посещений</TH>
        <TH WIDTH="40%" NOWRAP>Время последнего посещения</TH>
    </TR>
    <?php
        while($query_data = mysql_fetch_array($result)) {
            $page = $query_data["page"];
            $visitcount = $query_data["visitcount"];

```

Для получения форматированной строки с датой используется PHP-функция `substr()` (функция MySQL-сервера, позволяющая достичь той же цели более простым способом, рассматривается в следующей главе):

```

        $accessdate = substr($query_data["accessdate"], 0, 4) . '-' .
            substr($query_data["accessdate"], 4, 2) . '-' .
            substr($query_data["accessdate"], 6, 2) . ' ' .
            substr($query_data["accessdate"], 8, 2) . ':' .
            substr($query_data["accessdate"], 10, 2) . ':' .
            substr($query_data["accessdate"], 12, 2);
        echo "<TR>\n";
        echo "<TD WIDTH=\"40%\">$page</TD>\n";
        echo "<TD WIDTH=\"20%\" ALIGN=\"CENTER\">$visitcount</TD>\n";
        echo "<TD WIDTH=\"40%\" ALIGN=\"CENTER\">$accessdate</TD>\n";
        echo "</TR>\n";
    }
?>
</TR>
</TABLE>

```

```
</DIV>
<?php
    }
    html_footer();
}
```

Выбор действия

В конце файла `userview.php` расположен блок кода, который выполняется первоначально при запуске сценария. В зависимости от значения переменной `$action` программа вызывает одну из главных функций, которые были определены выше. Таким образом, в одном сценарии создается двойная функциональность. Конечно, необходимо настроить поведение по умолчанию на случай первого посещения данной страницы. Для этого проверяется элемент `action` массива `$_GET`, и если он не установлен, то ему присваивается значение по умолчанию — `list_records`:

```
if (empty($_GET['action'])) {
    $_GET['action'] = 'list_records';
}
switch($_GET['action']) {
    case "view_record":
        view_record();
        break;
    default:
        list_records();
        break;
}
```

Это все. Запустите Web-браузер и вызовите в нем страницу `userviewer.php`.

Использование сценария

Рассмотрим работу сценария на примере. Предположим, что требуется отсортировать записи по имени пользователя и вывести подробные сведения о пользователе Brian Reid. Результаты зависят от имеющейся в базе данных `sample_db` информации, но в целом страницы будут иметь вид, аналогичный рис. 10.2.

Следует отметить URL в обоих окнах — в верхнем окне URL указывает на то, что данные отсортированы по имени пользователя, а в нижнем окне показаны подробные сведения о пользователе Brian Reid, идентификатор которого (значение поля `userid`) равен Pads.

Резюме

В главе рассматривались способы получения информации из базы данных, а также описывалась основная MySQL-команда `SELECT`, предназначенная для извлечения данных из таблицы.

Часто для формирования запросов используется несколько SQL-предложений, позволяющих сузить диапазон возвращаемых данных, упорядочить эти данные или определенным образом сгруппировать их. В главе описывались следующие SQL-предложения:

- `LIMIT` — предложение для ограничения количества возвращаемых результатов;
- `WHERE` — предложение, определяющее критерии отбора данных;

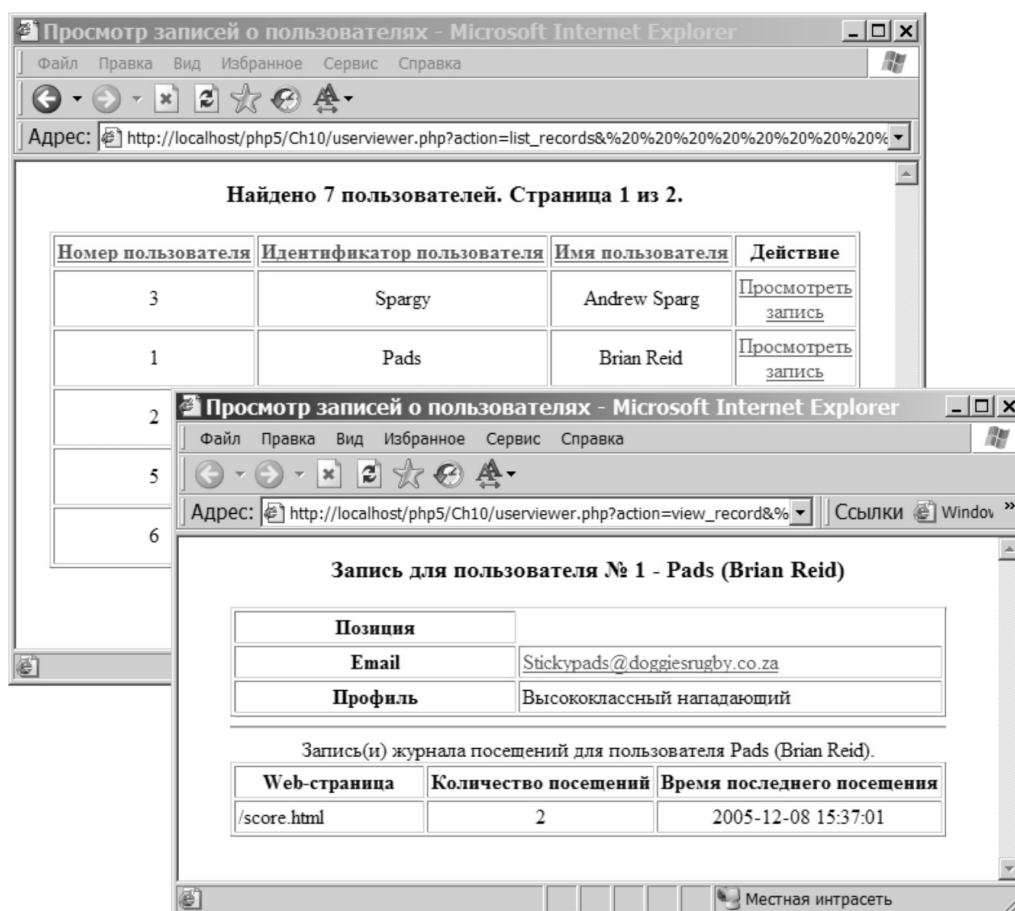


Рис. 10.2.

- ☐ ORDER BY — предложение для упорядочения возвращаемых результатов;
- ☐ GROUP BY — предложение для группировки возвращаемых результатов.

Кроме того, в главе было введено понятие псевдонимов и операций объединения для получения данных из нескольких таблиц. Рассматривались встроенные функции MySQL-сервера и связанные с MySQL PHP-функции для работы с информацией в базе данных.

В дополнение к этому в данной главе был представлен сценарий для просмотра информации о пользователях, который способен разделять список пользователей на несколько страниц, а также дает возможность навигации по этим страницам. Сценарий также позволяет просматривать персональные данные и информацию из журнала посещений по определенному пользователю.

Обсуждение аспектов управления базами данных представлено в следующей главе, в которой рассматривается вставка, удаление и обновление записей в базах данных с помощью PHP.

11

Использование PHP для управления информацией в базах данных MySQL

В этой главе показано, как управлять записями в базах данных MySQL из PHP-сценариев. В частности, здесь рассматривается вставка новых записей в таблицу базы данных с помощью PHP, а также удаление и обновление записей, уже имеющихся в таблице базы данных.

Чтобы дать читателю возможность разобраться в этой функциональности, авторы подробно рассматривают разработку двух сценариев: регистрации пользователей (User Manager) и протоколирования посещения страниц (Access Logger). Кроме того, в этой главе показано, как усовершенствовать сценарий для просмотра пользовательских записей, созданного в главе 10, добавив в него возможность управления записями пользователей. Эта глава завершает серию глав, посвященных MySQL и взаимодействию PHP и MySQL.

Вставка записей с помощью PHP

В предыдущей главе было показано, как вставлять данные в таблицу, используя клиентскую программу `mysql`. Например, следующий запрос вставляет запись для пользователя Pads:

```
mysql> INSERT INTO user VALUES (  
-> NULL,
```

```

-> 'Pads',
-> Password(12345),
-> 'Brian Reid',
-> 'Winger',
-> 'Stickypads@doggies_rugby.com',
-> 'Высококласный перехватчик мячей.');

```

Той же цели можно достичь с помощью PHP:

```
$result = mysql_query("INSERT INTO user VALUES( ... )");
```

Обычно для вставки значений в таблицу вместо непосредственного указания значений в запросе используются PHP-переменные:

```

$query = "INSERT INTO user VALUES(NULL, '$userid',
                                     password('$userpassword'),
                                     '$username', '$userposition', '$useremail',
                                     '$userprofile')";
$result = mysql_query($query);

```

Напомним, что поле `usernumber` имеет тип `AUTO_INCREMENT` — запрос передает в это поле `NULL`, чтобы его значение в каждой новой добавляемой записи было на единицу больше, чем у предыдущей записи.

Иногда возможность получать результирующее значение бывает очень удобной; например, можно сделать так, чтобы сценарий при регистрации нового пользователя показывал ему его регистрационный номер. Автоматически увеличенное число, сгенерированное последним `INSERT`-запросом, можно получить с помощью функции `mysql_insert_id()`. Например:

```

$link_id = db_connect('sample_db');
$result = mysql_query("INSERT INTO user (userid)
                     VALUES('sphinx')");
$usernumber = mysql_insert_id($link_id);
echo "Спасибо. Ваш номер - $usernumber.";

```

Специальные символы

Как уже отмечалось в главе 9, прежде чем вставлять строковое значение в таблицу базы данных, необходимо экранировать каждую одинарную кавычку в этом значении, в противном случае возникает ошибка. В предыдущей главе приводился пример с вставкой строки "I'm a rugby player" в поле `userprofile` типа `TEXT`. Ввиду того, что эта строка ограничена двойными кавычками, она хорошо вставляется в базу данных как с помощью клиентской программы `mysql`, так и с помощью PHP. Чтобы заключить эту строку в одинарные кавычки, понадобилось бы экранировать одинарную кавычку, содержащуюся в строке. Рассмотрим несколько примеров:

```

"I'm a PHP developer."  Работает.

'I\m a PHP developer.'  Тоже работает.

'I'm a PHP developer.'  Не работает.

```

Однако почему приходится учитывать эти различия? Несомненно, лучше всего экранировать все вхождения кавычек, как двойных, так и одинарных — в конце концов, в PHP экранированные кавычки работают как в строковых значениях, ограниченных одинарными кавычками, так и в строковых значениях, ограниченных двойными кавычками. Все зависит от программы. Если поставить обратную косую черту перед одинарной кавычкой в первом примере, то получится следующая строка: "I\'m a PHP developer".

Если впоследствии вывести эту строку с помощью PHP, то она отобразится в браузере в таком же виде, т.е. с обратной косой чертой, а это очевидно совсем не то, что нужно. Данная проблема решается в PHP с помощью пары очень удобных функций: `addslashes()` и `stripslashes()`.

Эти функции тесно связаны друг с другом. Функция `addslashes()` принимает строковый аргумент, предваряет каждый символ, который должен быть экранирован в запросах к базе данных, символом обратной косой черты, а затем возвращает модифицированную строку. Функция `stripslashes()` решает обратную задачу, удаляя из строки все символы обратной косой черты.

Следующие строки кода показывают, как можно использовать эти функции для вставки записей в таблицу, экранируя значения так, как это требуется:

```
$link_id = db_connect('sample_db');
$userprofile = addslashes($userprofile);
$query = "INSERT INTO user (userprofile) VALUES('$userprofile')";
mysql_query($query, $link_id);
$userprofile = stripslashes($userprofile);
```

Если предположить, что в переменной `$userprofile` хранится строка:

```
I'm a PHP developer.
```

то после обработки функцией `addslashes()` эта переменная получит значение:

```
I\'m a PHP developer.
```

Теперь эту переменную можно безопасно передавать в SQL-оператор. Когда понадобится прочесть это значение из базы данных, сервер вернет строку с кавычкой, но без `escape`-последовательностей. Однако если использовать переменную `$userprofile` в последующем коде, то посторонние символы обратной косой черты будут искажать правильно отформатированный текст. Чтобы привести эту строку в порядок, следует использовать функцию `stripslashes()`.

Функция `htmlspecialchars()`

Вопросы, связанные со специальными символами, не заканчиваются экранированием кавычек. Как известно, определенные символы имеют в HTML специальное значение, например, символ `<` обозначает начало HTML-тега. Если этот символ содержится во введенной пользователем строке, которая затем выводится на HTML-странице, то это неизбежно приведет к неверному отображению Web-страницы в браузере. Синтаксический анализатор HTML-кода будет связывать все, что следует после этого символа (вплоть до символа `>`), с каким-либо тегом, хотя вряд ли это будет действительно так.

Чтобы избежать этого, можно представить указанный символ в форме HTML-последовательности, которая выглядит так:

```
&lt;
```

PHP-функция `htmlspecialchars()`, которая уже рассматривалась вкратце в главе 5, решает задачу преобразования специальных HTML-символов в их текстовую форму, что значительно упрощает работу программиста. Так же как функции, добавляющие/удаляющие обратную косую черту, которые рассматривались выше, `htmlspecialchars()` принимает строковый аргумент, заменяет все специальные символы соответствующими безопасными формами и возвращает модифицированную строку:

```
$userprofile = htmlspecialchars($userprofile);
```

Преобразовываются следующие символы:

Специальный символ	HTML-последовательность
& (амперсанд)	&
" (двойная кавычка)	"
< (знак меньше)	<
> (знак больше)	>

Теперь, после того как были раскрыты некоторые тонкости, связанные со вставкой информации в базы данных с помощью PHP, следует рассмотреть обновление и удаление существующих данных.

Обновление и удаление записей в таблицах

Предположим, что один из зарегистрированных пользователей сайта намерен изменить его регистрационное имя, так как в нем обнаружилась опечатка. Вместо того чтобы заменять всю запись для данного пользователя, можно изменить только значение в поле `username` имеющейся записи. Очевидно, что для этого используется команда `UPDATE`. Рассмотрим пример соответствующего оператора:

```
UPDATE user SET username = 'Darren Ebbs'
      WHERE username = 'Daren Ebbs'
```

Указанная команда изменяет в поле `username` все вхождения строки `Daren Ebbs` на `Darren Ebbs`.

Чтобы удалить из таблицы существующую запись, необходимо использовать команду `DELETE`:

```
DELETE FROM access_log WHERE page = 'who.html'
```

Она удаляет все записи, в которых значение поля `page` равно `'who.html'`.

Ниже приводится пример такого `UPDATE`-запроса, который использовать *не следует*. Он обновляет все записи в таблице `user` так, что любой пользователь впоследствии сможет входить на сайт, используя пароль `comeonin`:

```
UPDATE user SET userpassword = password('comeonin')
```

Еще хуже последствия запроса, который очищает таблицу `user`:

```
DELETE FROM user
```

Такого рода грубые ошибки могут допускать даже очень опытные администраторы баз данных — это последствия невнимательности во время ввода запроса. Используя данные команды, следует быть очень внимательным.

Предположим, что из таблицы `access_log` требуется удалить все записи, начиная с 1999 года, так как они больше не нужны. Для этого можно использовать такой запрос:

```
mysql> DELETE FROM access_log WHERE accessdate LIKE '1999%';
```

Пользователь может захотеть время от времени изменять свой пароль. Следующий запрос изменяет пароль пользователя на `guessit`:

```
mysql> UPDATE user SET userpassword = password('guessit')
      -> WHERE userid = 'Greeny';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```


Сервер сообщает, что с заданным условием совпала одна строка, которая, следовательно, и была изменена. Если сервер не находит совпадающих записей, то обновление не выполняется:

```
mysql> UPDATE user SET userpassword = password('guessit')
-> WHERE userid = 'Ginger';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0 Changed: 0 Warnings: 0
```

Можно использовать существующие значения для обновления записи. Например, в поле `visitcount` содержится количество посещений определенным пользователем какой-либо Web-страницы. Если пользователь посещает данную страницу снова, то значение в поле `visitcount` необходимо увеличить на 1. Для этого можно получить текущее значение из таблицы `access_log`, прибавить к нему 1, а затем записать в таблицу новое значение. Однако той же цели можно достичь намного проще — используя существующее значение в UPDATE-запросе. Следующий запрос делает как раз то, что нужно — он увеличивает значение поля `visitcount` для пользователя Dodge, посещающего страницу `/score.html`:

```
mysql> UPDATE access_log SET visitcount = visitcount + 1
-> WHERE user_id = 'Dodge' AND page = '/score.html';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Поле `lastaccesstime` необходимо обновлять отдельно (этот вопрос обсуждается далее).

Строковые значения обновляются аналогично:

```
mysql> UPDATE user SET userid = concat(userid, '_1');
Query OK, 6 row affected (0.00 sec)
Rows matched: 6 Changed: 6 Warnings: 0
```

В запросе используется функция MySQL-сервера для слияния строк, которая добавляет строку `_1` к каждому значению поля `userid`. Например, в данном случае идентификатор пользователя Dodge станет равным Dodge_1. Этот метод оказывается очень удобным, когда новый пользователь запрашивает идентификатор, который уже используется другим пользователем, и нужно предложить альтернативный идентификатор.

Если бы использовался запрос SET userid = userid + '_1', то значение поля userid было бы равным 0, а не Dodge_1.

Допускаются также множественные обновления одного поля:

```
mysql> UPDATE access_log
-> SET visitcount = visitcount + 1, visitcount = visitcount * 2;
Query OK, 6 rows affected (0.00 sec)
Rows matched: 6 Changed: 6 Warnings: 0
```

Присвоения значений выполняются слева направо, поэтому во втором присвоении используется уже увеличенное на 1 значение поля `visitcount`. Другими словами, сначала ко всем текущим значениям поля `visitcount` прибавляется 1, а затем полученные числа удваиваются.

К настоящему моменту читатели уже, вероятно, привыкли видеть отчеты MySQL-сервера в командной строке. Когда используется запрос DELETE или UPDATE, сервер автоматически сообщает количество задействованных строк, время, потраченное на выполнение данного запроса, количество совпадающих и измененных строк, а также выводит любые сгенерированные предупреждения.

Первый из этих параметров можно получить в PHP-сценарии, используя функцию `mysql_affected_rows()`. Она в качестве аргумента принимает идентификатор соединения с базой данных и возвращает количество записей, затронутых предыдущим SQL-запросом — DELETE-запросом в следующем примере:

```
$link_id = db_connect("sample_db");
mysql_query("DELETE FROM access_log
            WHERE page = '/penalties.html'");
$num_rows = mysql_affected_rows($link_id);
echo "Было удалено $num_rows пользователей.";
```

Если страницу нарушений посещал только один пользователь, то сценарий вывел бы следующее сообщение:

Было удалено 1 пользователей.

Если сценарий удаляет все записи в таблице, то функция `mysql_affected_rows()` возвращает 0.

Работа с полями даты и времени

Пожалуй, из всех типов данных в MySQL самыми сложными для обработки являются дата и время. Для правильной работы с ними необходима длительная практика. В полях этих типов значения даты и времени хранятся в следующих форматах:

Тип данных	Формат
DATE	2004-01-01
TIME	12:00:00
DATETIME	2004-01-01 12:00:00
TIMESTAMP	20000101120000
YEAR	2004

Если попытаться вставить в такое поле неверно отформатированное значение даты и/или времени, то это поле в соответствующей записи будет заполнено нулями. Например, вставка строки Pads в поле TIME-типа приведет к тому, что в данном поле сохранится значение 0000000000000000:

```
mysql> select accessdate from access_log where userid = 'Pads' AND page = '';
+-----+
| accessdate |
+-----+
+ 0000000000000000 |
+-----+
1 row in set (0.00 sec)
```

MySQL-сервер предоставляет несколько встроенных функций, связанных с датой и временем, которые позволяют гарантировать вставку корректных значений. Выше уже упоминалась функция `now()`, которая возвращает текущую дату и время в формате DATETIME:

```
mysql> select now();
+-----+
| now() |
+-----+
| 2004-03-04 19:30:15 |
+-----+
1 row in set (0.00 sec)
```

а также функции `curdate()` и `curtime()`, которые возвращают по отдельности значения `DATE` и `TIME`:

```
mysql> select curdate(), curtime();
+-----+-----+
| curdate() | curtime() |
+-----+-----+
| 2004-03-04 | 19:31:33 |
+-----+-----+
1 row in set (0.00 sec)
```

Для вставки значения в поле типа `TIMESTAMP` можно указать либо дату и время в виде строки из 14 цифр, либо `NULL`, в результате чего в поле сохраняется текущая системная дата и время в таком же формате. Чтобы правильно обновлять записи в таблице `access_log`, следует использовать команду:

```
mysql> UPDATE access_log
-> SET visitcount = visitcount + 1, accessdate = NULL
-> WHERE userid = 'Dodge' AND page = '/score.html';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Серверная функция `date_format()` возвращает отформатированное значение даты и времени. Она принимает аргумент `DATE`- или `DATETIME`-типа, а также строку формата. Если поле `lastaccesstime` имеет тип `DATE` или `DATETIME`, то можно использовать следующую команду:

```
mysql> SELECT date_format('2004-03-04 22:23:00', '%M %e, %Y');
+-----+
| date_format('2004-03-04 22:23:00', '%M %e, %Y') |
+-----+
| March 4, 2004 |
+-----+
1 row in set (0.03 sec)
```

Существует более 30 спецификаторов, которые можно использовать в строке формата. Ниже перечислены наиболее распространенные из них.

Спецификатор	Описание
%s	Секунда в двузначной числовой форме (00,01...)
%i	Минута в двузначной числовой форме (00,01...)
%H	Час в двузначной 24-часовой числовой форме (00,01...)
%h	Час в двузначной 12-часовой числовой форме (00,01...)
%T	Время в 24-часовой форме (hh:mm:ss)
%r	Время в 12-часовой форме (hh:mm:ss AM PM)
%W	День недели (Monday, Tuesday, Wednesday...)
%a	Сокращенное название дня недели (Mon, Tue, Wed...)
%d	День месяца в двузначной числовой форме (01,02,03...)
%e	День месяца в числовой форме (1,2,3...)
%D	День месяца с порядковым суффиксом (1st, 2nd, 3rd...)
%M	Месяц (January, February...)
%b	Сокращенное название месяца (Jan, Feb...)
%Y	Год в четырехзначной числовой форме
%y	Год в двухзначной числовой форме

Полный перечень спецификаторов приведен в документации по MySQL www.mysql.com/doc/en/Date_and_time_functions.html#IDX1335.

В коде функции `view_record()` сценария для просмотра пользовательских записей (см. предыдущую главу) значение поля `accessdate` форматировалось с помощью следующих строк кода:

```
$accessdate = substr($query_data["accessdate"], 0, 4) . '-' .
    substr($query_data["accessdate"], 4, 2) . '-' .
    substr($query_data["accessdate"], 6, 2) . ':' .
    substr($query_data["accessdate"], 8, 2) . ':' .
    substr($query_data["accessdate"], 10, 2) . ':' .
    substr($query_data["accessdate"], 12, 2);
```

которые генерировали значение даты и времени в следующем формате:

```
2004-01-22 10:31:35
```

Той же цели можно достичь с помощью одного запроса:

```
mysql> SELECT date_format(accessdate, '%Y-%m-%d %H:%i:%s')
-> FROM access_log WHERE userid='Dodge';
+-----+
| date_format(accessdate, '%Y-%m-%d %H:%i:%s') |
+-----+
| 2004-03-11 12:22:49 |
| 2004-01-25 16:41:33 |
+-----+
2 rows in set (0.00 sec)
```

Если необходимо узнать день недели, когда состоялось последнее посещение страницы определенным пользователем, то можно применить следующий запрос, в котором используется серверная функция `dayname()`, возвращающая название дня недели для заданной даты:

```
mysql> SELECT accessdate, dayname(accessdate) FROM access_log
-> WHERE userid='Dodge';
+-----+-----+
| accessdate | dayname(accessdate) |
+-----+-----+
| 20040311122249 | Thursday |
| 20040125164133 | Sunday |
+-----+-----+
2 rows in set (0.01 sec)
```

Серверная функция `to_days()` вычисляет общее количество дней с нулевого года нашей эры:

```
mysql> SELECT to_days(accessdate) FROM access_log
-> WHERE userid='Dodge';
+-----+
| to_days(accessdate) |
+-----+
| 732016 |
| 731970 |
+-----+
2 rows in set (0.00 sec)
```

Предположим, что требуется узнать количество дней, прошедших после последнего посещения пользователем сайта:

```
mysql> SELECT userid, to_days(now()) - to_days(accessdate)
-> FROM access_log GROUP BY userid LIMIT 5;
+-----+-----+
| userid | to_days(now()) - to_days(accessdate) |
+-----+-----+
```

Brian	7
Dodge	0
Greeny	47
Mac	49
Nicrot	47

-----+
5 rows in set (0.01 sec)

Данный запрос в частности показывает, что пользователь Mac не посещал сайт 49 дней. Если это обстоятельство огорчает владельца сайта, то можно просто удалять пользователей, которые не посещали сайт более чем 48 дней. Это можно сделать с помощью следующего запроса:

```
mysql> DELETE FROM access_log WHERE to_days(now()) - to_days(accessdate) > 48;
Query OK, 1 row affected (0.01 sec)
```

Выше приведены лишь простые примеры некоторых серверных MySQL-функций для обработки даты и времени. За более подробной информацией рекомендуется обратиться к справочному руководству по MySQL на Web-сайте www.mysql.com.

Получение информации о таблицах в базе данных

Выяснение информации об используемой базе данных часто бывает настолько же важным, насколько важны сами данные, содержащиеся в базе. Например, нередко для принятия решения о следующей операции над таблицей необходимо узнать некоторые сведения *об этой таблице*. Сделать это несложно — получить информацию о таблицах баз данных с помощью PHP так же просто, как получить сами данные из таблиц.

Для просмотра структуры таблиц базы данных предназначены следующие функции.

- ❑ `mysql_list_fields()`: эту функцию следует вызывать первой для получения информации о полях таблицы. Функция принимает три аргумента (имя базы данных, имя таблицы и необязательный идентификатор подключения к базе данных) и возвращает указатель результата, который ссылается на список всех полей в заданной таблице заданной базы данных. Например, чтобы получить указатель на список всех полей в таблице `user` базы данных `sample_db`, можно использовать следующий код:

```
$result = mysql_list_fields("sample_db", "user");
```

Переменная `$result` теперь содержит указатель на список полей в таблице `user`. Пример использования данной функции будет представлен позднее.

- ❑ `mysql_num_fields()`: принимает в качестве аргумента указатель на результат (возвращаемый предыдущей функцией) и возвращает общее количество полей результирующего множества, на которое ссылается данный указатель. Чтобы получить количество полей в таблице `user`, достаточно передать в функцию `mysql_num_fields()` указатель `$result`, полученный от функции `mysql_list_fields()`:

```
$number_of_fields = mysql_num_fields($result);
```

Переменная `$number_of_fields` содержит (как и ожидалось) количество полей таблицы `user`.

- ❑ `mysql_field_name()`, `mysql_field_len()` и `mysql_field_type()`: эти функции возвращают имя поля, длину поля и тип поля соответственно. Каждая

из функций принимает в качестве аргументов указатель на результирующее множество и индекс поля, определяющие таблицу и поле (отсчет полей начинается с нуля) соответственно. Например, чтобы получить длину поля `username` в таблице `user`, сначала необходимо знать, что поле `username` — четвертое поле таблицы и, следовательно, имеет индекс 3 (индексы начинаются с нуля), а затем можно использовать следующий код:

```
$username_length = mysql_field_len($result, 3);
```

Так как при определении поля `username` его длина была задана равной 30 символам, в результате выполнения данной функции переменная `$username_length` должна содержать число 30. Две другие функции работают почти так же.

- `mysql_field_flags()`: принимает указатель на результирующее множество и индекс поля, а возвращает строку, содержащую атрибуты для заданного поля. Атрибутами поля могут быть `NULL` или `NOT NULL`, `PRIMARY KEY`, `UNIQUE`. Чтобы записать атрибуты поля `username` (таблицы `user`) в переменную `$attributes`, можно использовать следующий код:

```
$attributes = mysql_field_flags($result, 3);
```

Поле `username` имеет единственный атрибут — `NOT NULL`, поэтому переменная `$attributes` получит это значение.

Практика Получение информации о полях

Рассмотрим упомянутые выше функции в действии. Следующий сценарий, `metadata.php`, возвращает имя, длину и тип каждого поля, определенного в таблице `user`:

```
<?php
include "../common_db.inc";
$link_id = db_connect();
$result = mysql_list_fields("sample_db", "user", $link_id);

for($i=0; $i < mysql_num_fields($result); $i++ ) {
    echo mysql_field_name($result,$i );
    echo "(" . mysql_field_len($result, $i) . ")";
    echo " - " . mysql_field_type($result, $i) . "<BR>";
}
?>
```

Результат работы данного сценария представлен на рис. 11.1.

Как это работает

Как обычно, сначала сценарий подключает файл с функциями для работы с базами данных (`common_db.inc`), а затем подключается к серверу, используя функцию `db_connect()`:

```
include_once "../common_db.inc";
$link_id = db_connect();
```

После этого вызывается функция `mysql_list_fields()` с указанием базы данных, таблицы и идентификатора соединения с используемой базой данных (в рассматриваемом случае `sample_db`, `user` и `$link_id` соответственно):

```
$result = mysql_list_fields("sample_db", "user", $link_id);
```

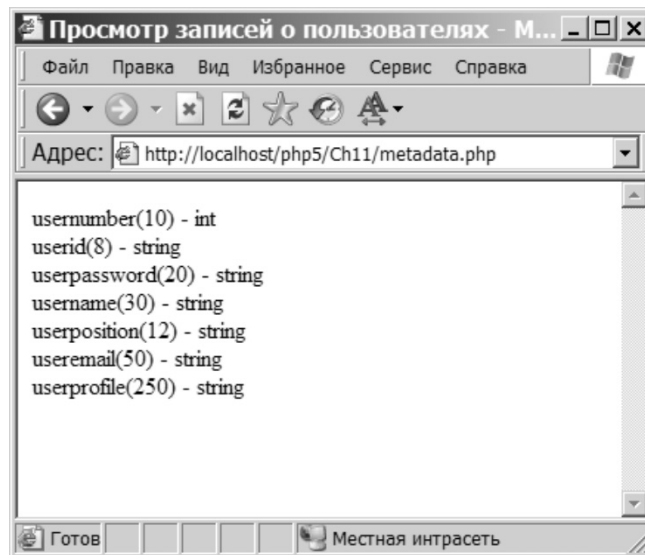


Рис. 11.1.

Данная функция возвращает указатель на результирующее множество, в котором перечислены все поля таблицы `user`. Указатель сохраняется в переменной `$result`, а затем используется в функции `mysql_num_fields()` для выяснения количества полей, содержащихся в данной таблице. Полученное значение используется как верхний предел для цикла `for...next`:

```
for($i=0; $i < mysql_num_fields($result); $i++) {
```

Затем используется индекс цикла для указания поля и вызываются функции `mysql_field_name()`, `mysql_field_len()` и `mysql_field_type()`, которые возвращают имя, длину и тип доступных полей:

```
echo mysql_field_name($result,$i );
echo "(" . mysql_field_len($result, $i) . " ";
echo " - " . mysql_field_type($result, $i) . "<BR>";
```

Некоторые из возвращаемых типов полей неоднозначны, например, `CHAR` и `VARCHAR` возвращаются как `string`, тогда как `TEXT` возвращается как `BLOB`. Чтобы получить точные типы полей (так, как они определены в `MySQL`-таблице), требуется дополнительная работа. Здесь оказывается полезной функция `mysql_field_flags()`. Код можно легко модифицировать так, чтобы сценарий выводил атрибуты полей. Ниже приведена новая версия данного сценария:

```
$link_id = db_connect();
$result = mysql_list_fields("sample_db", "user", $link_id);
```

```
for($i=0; $i < mysql_num_fields($result); $i++) {
    echo mysql_field_name($result,$i );
    echo "(" . mysql_field_len($result, $i) . " ";
```

```
    echo " - " . mysql_field_type($result, $i);
    echo " " . mysql_field_flags($result, $i) . "<BR>";
```

```
}
```

Примерный результат работы сценария показан на рис. 11.2.

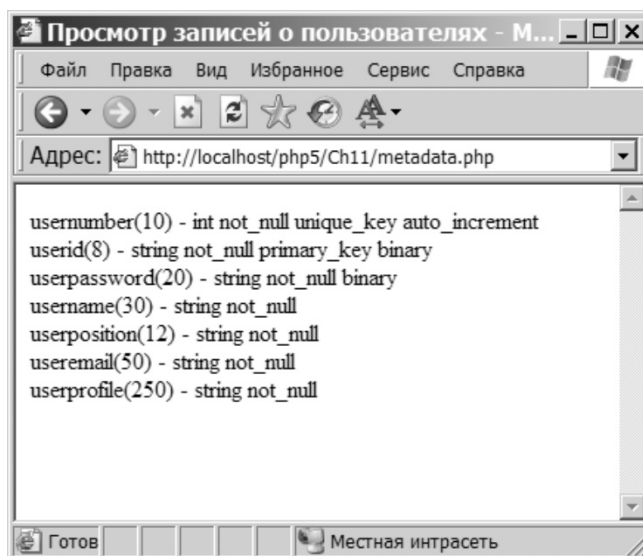


Рис. 11.2.

Наконец, функцию `mysql_fetch_field()` можно использовать для получения всей информации о полях таблицы за один вызов. Она принимает два аргумента — указатель на результирующее множество и индекс поля (отсчет также начинается с нуля) и возвращает объект, в перечень свойств которого включается имя поля и имя таблицы (которой это поле принадлежит), длина поля, атрибуты и другие сведения. Свойства возвращаемого объекта описываются в приведенной ниже таблице.

Свойство	Описание
<code>blob</code>	Равно <code>True</code> , если поле имеет тип <code>BLOB</code>
<code>max_length</code>	Максимальная длина поля
<code>multiple_key</code>	Равно <code>True</code> , если поле является ключом, но не уникально
<code>name</code>	Имя поля
<code>not_null</code>	Равно <code>True</code> , если значением поля не может быть <code>NULL</code>
<code>numeric</code>	Равно <code>True</code> , если поле является числовым
<code>primary_key</code>	Равно <code>True</code> , если поле является первичным ключом
<code>table</code>	Имя таблицы, которой принадлежит данное поле
<code>type</code>	Тип поля
<code>unique_key</code>	Равно <code>True</code> , если поле является уникальным ключом
<code>unsigned</code>	Равно <code>True</code> для беззнакового поля
<code>def</code>	Значение по умолчанию, если оно задано
<code>zerofill</code>	Равно <code>True</code> , если поле заполняется нулями. Этот атрибут используется для того, чтобы сохранять число определенной длины, заполняя поле ведущими нулями

Практика Использование функции `mysql_fetch_field()`

Указанную функцию можно использовать для получения результатов, аналогичных результатам предыдущего примера. Сохраните код в файле `fetch_field.php`:

```
<?php
include "./common_db.inc";
$link_id = db_connect();
$result = mysql_list_fields("sample_db", "user", $link_id);

for($i=0; $i < mysql_num_fields($result); $i++ ) {

    $field_info_object = mysql_fetch_field($result, $i);

    echo $field_info_object->name . "(" .
        $field_info_object->max_length . ")";

    echo " - " . $field_info_object->type;

    if($field_info_object->def) {
        echo "<br><b>Значение по умолчанию:</b>";
        $field_info_object->def</b> ";
    }
    if($field_info_object->not_null) {
        echo " not_null ";
    } else {
        echo " null ";
    }

    if($field_info_object->primary_key) {
        echo " primary_key ";
    } else if ($field_info_object->multiple_key) {
        echo " key ";
    } else if ($field_info_object->unique_key) {
        echo " unique ";
    }
    if($field_info_object->unsigned) {
        echo " unsigned ";
    }
    if($field_info_object->zerofill) {
        echo " zero-filled ";
    }
    echo "<BR>";
}
?>
```

Результат работы данного сценария показан на рис. 11.3.

Как это работает

Этот сценарий работает почти так же, как предыдущий, однако длина каждого поля равна 0, что, естественно, неверно. К сожалению, несмотря на то, что функция `mysql_fetch_field()` — весьма универсальный инструмент, ее реализация имеет небольшой дефект, который приводит к тому, что свойство `max_length` всегда равно 0 независимо от фактической длины поля. На момент написания данной книги последней версией PHP5 была версия RC1. Есть надежда, что в следующей версии эта ошибка будет устранена. А пока вернемся к функции `mysql_field_len()`:

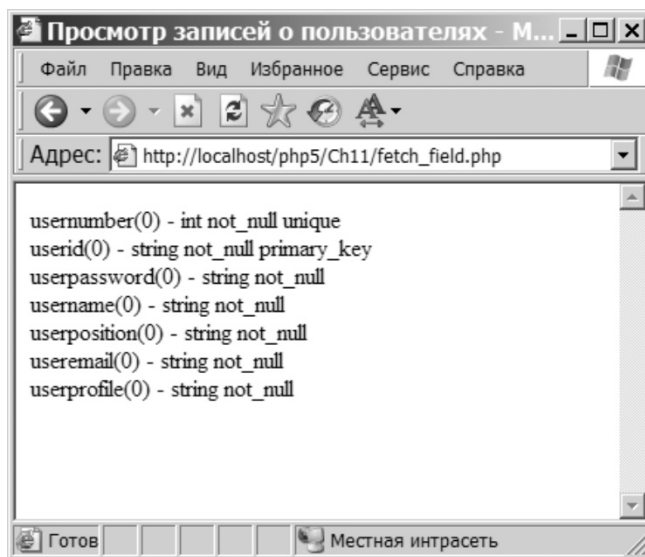


Рис. 11.3.

```
$field_info_object = mysql_fetch_field($result, $i);
```

```
echo $field_info_object->name . "(" .  
    mysql_field_len($result, $i) . ")";
```

```
echo " - " . $field_info_object->type;
```

Теперь сценарий возвращает корректный размер каждого поля (рис. 11.4).

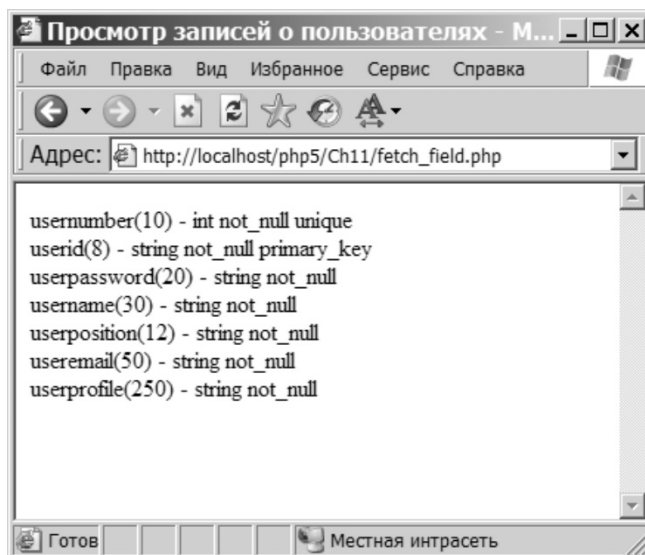


Рис. 11.4.

Возможность обращаться к метаданным MySQL-таблиц является неотъемлемой частью любого многоцелевого приложения для управления базами данных, как правило, ничего “не знающего” о структуре таблиц, с которыми оно должно работать. Чтобы всегда правильно обрабатывать таблицы, такое приложение должно в значительной степени опираться на функции, рассмотренные выше, и им подобные.

Параметры ENUM и стандартные значения полей

Тип поля ENUM уже упоминался в предыдущей главе, когда в таблицу пользователей было включено поле sex. Это поле может принимать одно из двух значений: M или F. В качестве еще одного хорошего примера рассмотрим модификацию поля userposition. Ограничим возможные значения данного поля предопределенным набором.

Первоначально для поля userposition был выбран тип VARCHAR(50) NOT NULL. Предположим, что все будущие игроки будут занимать одну из четырех позиций, которые уже введены в поле, и переопределим поле userposition с учетом этого предположения:

```
mysql> ALTER TABLE user MODIFY userposition
-> ENUM('Mid','Link','Winger','Utility Back') DEFAULT 'Utility Back';
Query OK, 6 rows affected (0.01 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Если в таблицу уже были добавлены другие записи, то эти значения можно модифицировать. Ранее введенные в это поле значения останутся нетронутыми.

Какая из РНР-функций показывает стандартное значение для поля таблицы? Ответ на этот вопрос находится в представленной выше таблице. Свойство def объекта, возвращаемого функцией mysql_fetch_field, содержит стандартное значение любого заданного поля. В последний пример можно добавить следующие строки кода:

```
if($field_info_object->def) echo "<br><b>Значение по умолчанию:
$field_info_object->def</b> ";
```

Примерный результат работы сценария показан на рис. 11.5.

Получить список доступных в поле ENUM-типа вариантов сложнее. Для этого необходимо ввести следующий запрос:

```
mysql> SHOW COLUMNS FROM user LIKE 'userposition';
```

результат которого представлен ниже:

Field	Type	Null	Key	Default	Extra
userposition	enum('Mid','Link','Winger','Utility Back')	YES		Utility Back	

1 row in set (0.02 sec)

Отчет показывает, что в информации о типе поля (Type) уже содержатся необходимые значения. Извлечь их можно с помощью несложного приема.

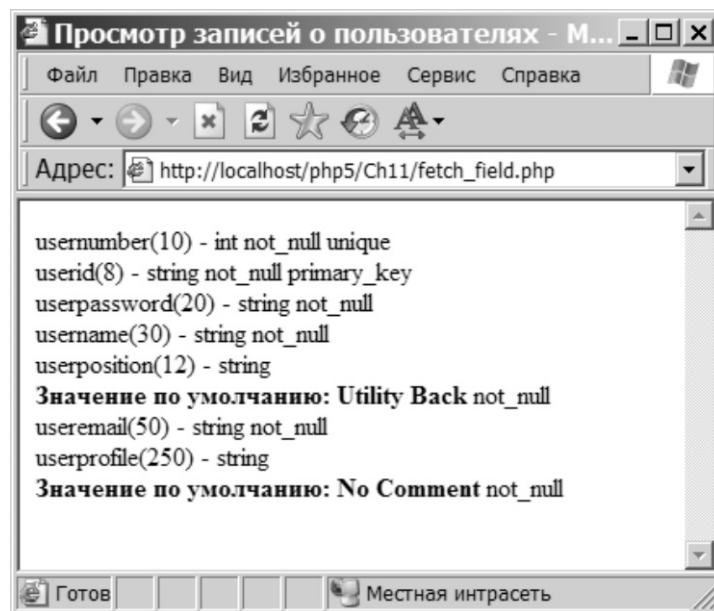


Рис. 11.5.

Практика Получение ENUM-значений

Следующий сценарий создает массив, элементы которого содержат все ENUM-значения. В последнем элементе массива содержится стандартное значение из предопределенного набора. Сохраните следующий код в файле `enum_options.php`:

```
<?php
include "../common_db.inc";
$link_id = db_connect();
mysql_select_db("sample_db");

$query = "SHOW COLUMNS FROM user LIKE 'userposition'";
$result = mysql_query($query);
$query_data = mysql_fetch_array($result);

if (ereg("(.*)", $query_data["Type"], $match)) {
    $enum_str = ereg_replace(" ", "", $match[1]);
    $enum_options = explode(',', $enum_str);
}

echo "ENUM-варианты и значение по умолчанию:<BR>";
foreach($enum_options as $value) {
    echo "-$value<BR>";
}

echo "<BR>Значение по умолчанию: <b>$query_data[Default]</b>";
echo "<P>";

?>
```

Результат работы данного сценария показан на рис. 11.6.

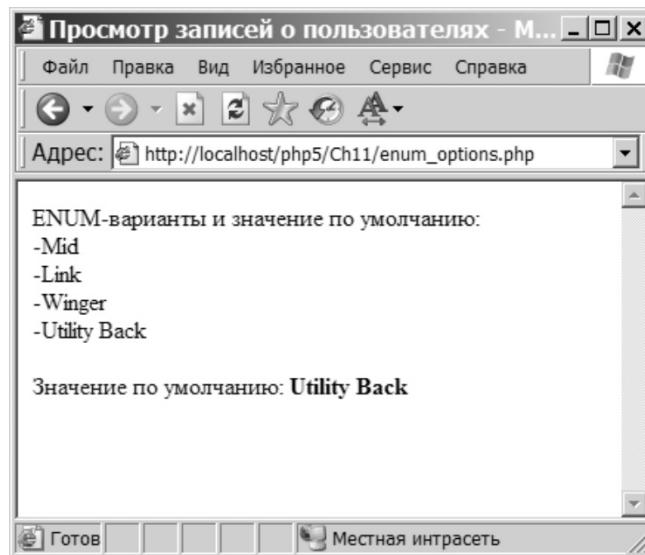


Рис. 11.6.

Как это работает

Сценарий начинается с ввода SQL-запроса, который находит запись, содержащую определение поля `userposition`. Данные извлекаются в форме ассоциативного массива `$query_data`:

```
$query = "SHOW COLUMNS FROM user LIKE 'userposition'";
$result = mysql_query($query);
$query_data = mysql_fetch_array($result);
```

Теперь к значениям `Type` и `Default` в определении поля можно получить доступ как к элементам массива `$query_data['Type']` и `$query_data['Default']` соответственно. В первом из них должна содержаться следующая строка:

```
enum('Mid','Link','Winger','Utility Back')
```

Чтобы извлечь значения, необходимо избавиться от предшествующей строки `enum`, скобок и кавычек. Сначала для поиска строки, заключенной в скобки, используется функция `eregi()` с регулярным выражением `('.*')`:

```
eregi("('.*)'", $query_data["Type"], $match)
```

Третий аргумент данной функции представляет собой массив, который содержит все совпадения с заданным образцом, а второй элемент массива, `$match[1]`, содержит искомую строку:

```
'Mid','Link','Winger','Utility Back'
```

Теперь с помощью функции `eregi_replace()` следует избавиться от одинарных кавычек и записать модифицированную строку в переменную `$enum_str`:

```
$enum_str = eregi_replace("'", "", $match[1]);
```

которая в результате должна выглядеть так:

```
Mid, Link, Winger, Utility Back
```

После чего переменная `$enum_str` передается функции `explode()`, которая помещает разделенные запятыми значения в массив `$enum_options`:

```
$enum_options = explode(',', $enum_str);
```

В результате этот массив содержит все доступные варианты выбора из ENUM-поля в виде отдельных элементов.

Остальная часть кода просматривает в цикле элементы массива `$enum_options` и выводит их. После того как выведены все enum-значения, жирным шрифтом выводится значение данного поля по умолчанию:

```
echo "ENUM-варианты и значение по умолчанию:<BR>";
foreach($enum_options as $value) {
    echo "-$value<BR>";
}
echo "<BR>Значение по умолчанию: <b>$query_data[Default]</b>";
echo "<P>";
```

Этот код можно поместить в функцию, которая возвращает массив, содержащий все возможные варианты значений из заданного ENUM-поля. Для этого следует добавить приведенный ниже код в конец подключаемого файла `common_db.inc`:

```
function enum_options($field, $link_id)
{
    $query = "SHOW COLUMNS FROM user LIKE '$field'";
    $result = mysql_query($query, $link_id);
    $query_data = mysql_fetch_array($result);
    if(ereg("(.*)", $query_data["Type"], $match)) {
        $enum_str = ereg_replace(" ", "", $match[1]);
        $enum_options = explode(',', $enum_str);
        return $enum_options;
    }else{
        return 0;
    }
}
```

Протестировать данную функцию можно с помощью следующего сценария (назовем его `test_enum.php`):

```
<?php
include_once "../common_db.inc";
$link_id = db_connect();
mysql_select_db("sample_db");

$array = enum_options('userposition', $link_id);
foreach($array as $var){
    echo $var."<BR>";
}
?>
```

Создание сценария для регистрации пользователей

Если предполагается, что Web-сайтом будет пользоваться некоторое сообщество зарегистрированных пользователей, то необходимо обеспечить способ для регистрации новых пользователей. Ниже рассматривается сценарий регистрации, работа которого состоит из следующих процедур:

1. Отображается страница с “Условиями использования”: посетитель сайта должен принять данные условия, если хочет стать зарегистрированным пользователем.
2. Выводится одна или несколько регистрационных форм, запрашивающих у пользователя необходимую информацию.
3. Создается учетная запись пользователя: в таблице `user` создается новая запись и в нее вставляется соответствующая информация.
4. Выводится сообщение, уведомляющее пользователя о том, что его учетная запись была создана.
5. По электронной почте отправляется запрос на подтверждение пользовательского e-mail-адреса (пользователь вводит свой адрес во время регистрации).

Здесь основное внимание будет уделено аспекту работы с данными в создаваемом сценарии. По этой причине первый пункт (в котором описываются условия доступа к регистрационной форме) и последний пункт приведенного выше списка здесь не рассматриваются. Несомненно, это базовое приложение впоследствии можно будет расширить.

Сценарий `register.php`

Создание новой учетной записи пользователя предполагает создание новой записи в первичном ключе таблицы `user`. Эта запись впоследствии будет использоваться во всех таблицах при регистрации действий пользователя на сайте. Кроме того, учетная запись используется для аутентификации пользователя при попытке входа в защищенную паролем область сайта. Поэтому учетная запись состоит из идентификатора пользователя и пароля.

Ничто не мешает хранить пользовательские пароли в виде простого текста, который будет доступен всем, но зашифрованные пароли обеспечивают более высокий уровень безопасности.

Ниже приводится исходный код сценария регистрации пользователей `register.php` с пояснениями. Сначала подключается файл `common_db.inc`:

```
<?php
//register.php
include_once "../common_db.inc";
```

Функция `in_use()`

Функция `in_use()` проверяет, существует ли уже в MySQL-базе вводимый пользователем идентификатор (`userid`), и возвращает 1, если это так:

```
function in_use($userid)
{
    global $user_tablename;

    $query = "SELECT userid FROM $user_tablename WHERE userid = '$userid'";
    $result = mysql_query($query);
    if(!mysql_num_rows($result)){
        return 0;
    }else{
        return 1;
    }
}
```

Функция register_form()

Функция register_form() отображает HTML-форму, в которую пользователь вводит необходимую для регистрации информацию. Сначала в функции выполняется подключение к базе данных. Затем определяются позиции, которые можно представить на выбор будущему игроку:

```
function register_form()
{
    global $userposition;
    global $PHP_SELF;

    $link_id = db_connect();
    mysql_select_db("sample_db");
    $position_array = enum_options('userposition', $link_id);
    mysql_close($link_id);
    ?>
```

Теперь массив \$position_array содержит все позиции, из которых пользователь может выбирать. Затем создается HTML-таблица, которая послужит в качестве пользовательского интерфейса. Обратите внимание, что в представленном примере для передачи данных используется метод POST:

```
<center><H3>Создание учетной записи!</H3></center>
<form method="post" action="<?php echo $PHP_SELF ?>">
<input type="hidden" name="action" value="register">
  <div align="center"><center><table border="1" width="90%">

    <tr>
      <th width="30%" nowrap>Идентификатор</th>
      <td width="70%"><input type="text" name="userid"
        size="8" maxlength="8"></td>

    </tr>
    <tr>
      <th width="30%" nowrap>Пароль</th>
      <td width="70%"><input type="password"
        name="userpassword" size="15"></td>

    </tr>
    <tr>
      <th width="30%" nowrap>Повторите пароль</th>
      <td width="70%"><input type="password"
        name="userpassword2" size="15"></td>

    </tr>
    <tr>
      <th width="30%" nowrap>Полное имя</th>
      <td width="70%"><input type="text" name="username" size="20"></td>
    </tr>
    <tr>
      <th width="30%" nowrap>Позиция</th>
      <td width="70%"><select name="userposition" size="1">
<?php
```

Здесь используется два поля для ввода пароля. Поскольку тип данного HTML-поля определен как PASSWORD, вводимое в поле значение заменяется звездочками, поэтому даже сам пользователь не видит то, что он вводит. Два обязательных для заполнения поля такого типа позволят исключить любые опечатки (далее в коде выполняется проверка на совпадение двух значений).

Выпадающий список для выбора позиции создается на лету из значений, записанных в массив `$position_array`, который создается в начале функции:

```
for($i=0; $i < count($position_array); $i++) {
    if(!isset($userposition) && $i == 0) {
        echo "<OPTION SELECTED VALUE=\"\". $position_array[$i] .
            \">\" . $position_array[$i] . "</OPTION>\n";
    }else if($userposition == $cposition_array[$i]) {
        echo "<OPTION SELECTED VALUE=\"\". $position_array[$i] . \">\" .
            $position_array[$i] . "</OPTION>\n";
    }else{
        echo "<OPTION VALUE=\"\". $position_array[$i] . \">\" .
            $position_array[$i] . "</OPTION>\n";
    }
}
?>
```

Далее выводятся другие поля формы:

```
</select></td>
</tr>
<tr>
    <th width="30%" nowrap>Email</th>
    <td width="70%"><input type="text" name="useremail" size="20"
    </td>
</tr>
<tr>
    <th width="30%" nowrap>Профиль</th>
    <td WIDTH="70%"><textarea rows="5" cols="40"
        name="userprofile"></textarea></td>
</tr>
<tr>
    <th width="30%" colspan="2" nowrap>
        <input type="submit" value="Отправить">
        <input type="reset" value="Очистить"></th>
</tr>
</table>
</center></div>
</form>
<?php
}
```

Функция `create_account()`

Функция `create_account()` вставляет новую запись в таблицу `user`. Эта функция может быть вызвана только в случае, если была нажата кнопка `Отправить`, созданная в функции `register_form()`. В результате массив `$_POST` будет заполнен значениями, которые затем можно будет использовать:

```
function create_account() {
    $userid = $_POST['userid'];
    $username = $_POST['username'];
    $userpassword = $_POST['userpassword'];
    $userpassword2 = $_POST['userpassword2'];
    $userposition = $_POST['userposition'];
    $useremail = $_POST['useremail'];
    $userprofile = $_POST['userprofile'];

    global $default_dbname, $user_tablename;
```

Введенные пользователем значения проверяются:

```
if (empty($userid)) {
    error_message("Введите желаемый идентификатор!");
}
if (empty($userpassword)) {
    error_message("Введите желаемый пароль!");
}
if (strlen($userpassword) < 4 ) {
    error_message("Короткий пароль!");
}
if (empty($userpassword2)) {
    error_message("Введите пароль снова для проверки!");
}
if (empty($username)) {
    error_message("Введите свое полное имя!");
}
if (empty($useremail)) {
    error_message("Введите свой e-mail-адрес!");
}

if (empty($userprofile)) {
    $userprofile = "Введите профиль.";
}

if ($userpassword != $userpassword2) {
    error_message("Повторно введенный пароль не совпадает с первоначально введенным!");
}
```

Затем создается подключение к базе данных и вызывается функция `in_use()` для проверки уникальности введенного пользователем идентификатора (так как идентификатор используется в качестве первичного ключа):

```
$link_id = db_connect($default_dbname);

if (in_use($userid)) {
    error_message("Идентификатор $userid уже используется. Пожалуйста, выберите другой идентификатор.");
}
```

При вставке данных нового пользователя в поле `usernumber` передается `NULL`, чтобы увеличить последнее значение в этом поле на 1. Кроме того, используется серверная функция `password()` для шифрования введенного пользователем пароля:

```
$query = "INSERT INTO user VALUES(NULL, '$userid', password('$userpassword'),
    '$username', '$userposition', '$useremail', '$userprofile')";
$result = mysql_query($query);
if (!$result) {
    error_message(sql_error());
}
```

С помощью функции `mysql_insert_id()` определяется только что созданный номер пользователя:

```
$usernumber = mysql_insert_id($link_id);
html_header();
?>
```

После этого выводится HTML-таблица, содержащая регистрационную информацию пользователя:

```

<center><h3>
Создана учетная запись для пользователя <?php echo $username ?>
</h3></center>

<div align="center"><center><table border="1" width="90%">
  <tr>
    <th width="30%" nowrap>Номер пользователя</th>
    <td width="70%"><?php echo $usernumber ?></td>
  </tr>
  <tr>
    <th width="30%" nowrap>Идентификатор</th>
    <td width="70%"><?php echo $userid ?></td>
  </tr>
  <tr>
    <th width="30%" nowrap>Пароль</th>
    <td width="70%"><?php echo $userpassword ?></td>
  </tr>
  <tr>
    <th width="30%" nowrap>Полное имя</th>
    <td width="70%"><?php echo $username ?></td>
  </tr>
  <tr>
    <th width="30%" nowrap>Позиция</th>
    <td width="70%"><?php echo $userposition ?></td>
  </tr>
  <tr>
    <th width="30%" nowrap>E-mail</th>
    <td width="70%"><?php echo $useremail ?></td>
  </tr>
  <tr>
    <th width="30%" nowrap>Профиль</th>
    <td width="70%"><?php echo htmlspecialchars($userprofile) ?></td>
  </tr>
</table>
</center></div>
<?php
    html_footer();
}

```

Выбор следующего действия

Наконец используется параметр `action` для определения того, какую функцию необходимо вызвать. В случае если массив `$_POST` не заполнен, параметру `action` следует присвоить значение по умолчанию, которое приводит к вызову функции `register_form()`:

```

if (empty($_POST)) {
    $_POST['action'] = "";
}
switch($_POST['action']) {
    case "register":
        create_account();
        break;
    default:
        html_header();
        register_form();
        html_footer();
        break;
}
?>

```

Рассмотрим работу данного сценария в действии. Сначала функция `register_form()` выводит HTML-форму (рис. 11.7), в которую пользователь вводит свои данные.

Просмотр записей о пользователях - Microsoft Internet E...

Файл Правка Вид Избранное Сервис Справка

Адрес: http://localhost/php5/Ch11/register.php

Создание учетной записи

Идентификатор	<input type="text"/>
Пароль	<input type="password"/>
Повторите пароль	<input type="password"/>
Полное имя	<input type="text"/>
Позиция	Utility Back ▾
Email	<input type="text"/>
Профиль	<div><div></div></div>
<div>Отправить Очистить</div>	

Готово Местная интрасеть

Рис. 11.7.

Затем пользователь может ввести свои данные, если сочтет нужным. На последнем этапе сценарий отображает сообщение о создании учетной записи и таблицу, в которой показана введенная пользователем информация, использованная для создания учетной записи. На рис. 11.8 показана страница с сообщением о создании учетной записи для пользователя Darren.

Следует признать, что отображение пароля после создания учетной записи — не очень хорошая идея. Относительно безопасный способ подтвердить данную информацию — включить ее в e-mail-сообщение, которое отправляется пользователю, создавшему учетную запись.

Итак, PHP можно использовать для добавления пользователей в базу данных сайта. Очевидно, что при расширении сценария или его интеграции в существующий сайт, возможно, придется добавить дополнительные кнопки, направляющие пользователя на другие страницы после подтверждения регистрации. Однако для учебных целей нынешней реализации сценария вполне достаточно.

Теперь рассмотрим протоколирование пользовательской активности на сайте.

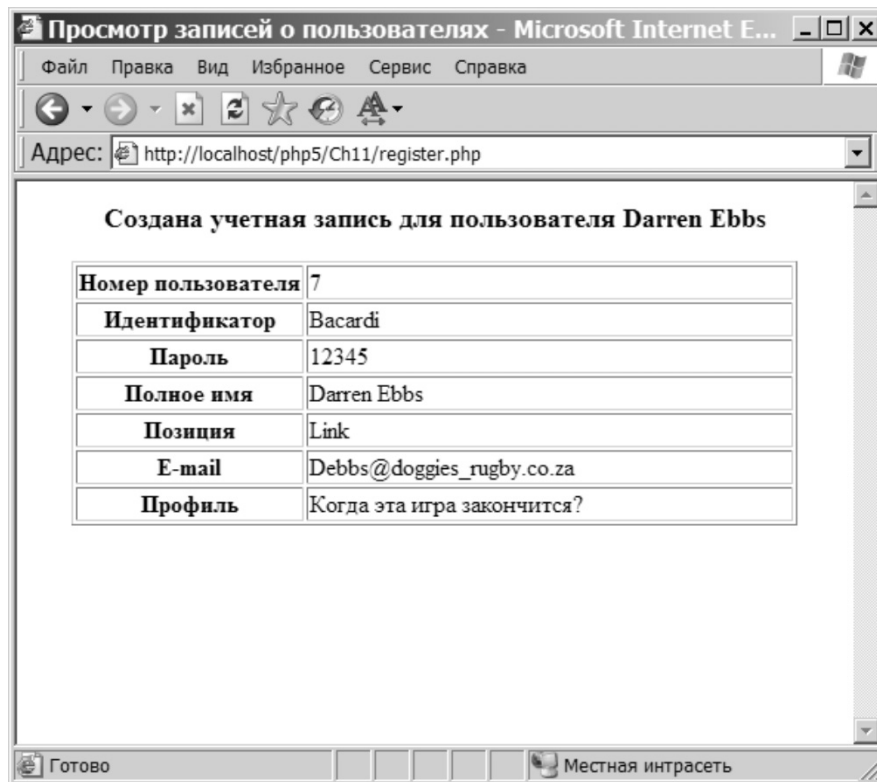


Рис. 11.8.

Создание сценария для протоколирования посещений страниц

Чтобы отслеживать активность пользователей на сайте, нужно знать, кто сделал тот или иной запрос. Для этого пользователей необходимо регистрировать. Регистрация дает возможность идентифицировать пользователей, запрашивающих какие-либо Web-страницы.

Самый удобный способ аутентификации пользователей заключается в том, чтобы присвоить каждому пользователю уникальный идентификатор и пароль. Когда пользователь входит в защищенную паролем область сайта, все что нужно сделать, это сверить введенные пользователем значения (имя пользователя и пароль) со значениями, хранящимися в базе данных.

Следующий сценарий для аутентификации пользователя выполняет запрос к таблице user. Код начинается с подключения файла common_db.inc и создания ссылки на страницу регистрации:

```
<?php
include_once "../common_db.inc";
$register_script = "../register.php";
```

Файл auth_user.php

Затем определяются функции `auth_user()` и `login_form()`, которые рассматриваются позднее, но основная часть сценария начинается с вызова функции `session_start()`. Необходимо сохранить введенные пользователем значения идентификатора и пароля, а также переменные сеанса, чтобы избавить пользователя от необходимости заново вводить эти значения каждый раз при посещении других страниц сайта.

Если переменная `$_POST['userid']` не была проинициализирована, то вызывается функция `login_form()`, отображающая форму входа на сайт, в которую пользователь может ввести свой идентификатор и пароль:

```
if(!isset($_POST['userid'])) {
    login_form();
    exit;
```

Если переменная `$userid` определена (а это возможно в том случае, когда сценарий был вызван из формы входа), то переданные пользователем значения для идентификатора (в HTML-форме поле `userid`) и пароля (`userpassword`) регистрируются как переменные сеанса для дальнейшего использования и передаются в качестве аргументов функции `auth_user()`:

```
}else{
    session_start();
    session_register("userid", "userpassword");
    $username = auth_user($_POST['userid'], $_POST['userpassword']);
```

Функция `auth_user()` может возвращать одно из двух значений:

- ☐ имя пользователя, которое хранится в таблице `user`; оно возвращается, только если переданные значения идентификатора и пароля совпадают со значениями, сохраненными в определенной записи таблицы;
- ☐ если совпадение не найдено, то функция возвращает 0.

Возвращенное функцией `auth_user()` значение сохраняется в переменной `$username` и используется для формирования ответа:

```
if(!$username) {
    $PHP_SELF = $_SERVER['PHP_SELF'];
    session_unregister("userid");
    session_unregister("userpassword");
    echo "Попытка авторизации неудачна. " .
        "Вы должны ввести правильный идентификатор пользователя и пароль. " .
        "Чтобы попытаться снова, щелкните следующую ссылку.<BR>\n";
    echo "<A HREF=\"\$PHP_SELF\">Вход в систему</A><BR>";
    echo "Если Вы еще не зарегистрированы, для регистрации щелкните следующую" .
        "ссылку." .
        "<BR>\n";
    echo "<A HREF=\"\$register_script\">Зарегистрироваться</A>";
    exit;
}else{
    echo "Добро пожаловать, $username!";
}
```

Если попытка аутентификации оказалась неудачной, необходимо разрегистрировать переменные сеанса, и гарантировать, таким образом, что в случае, когда неавторизованный пользователь попытается войти в защищенную паролем область сайта, будет

вызвана функция `login_form()`. Функция `login_form()` представляет собой HTML-таблицу, которая позволяет пользователю ввести идентификатор и пароль, чтобы впоследствии эти значения можно было использовать в функции `auth_user()`:

```
function login_form() {
    global $PHP_SELF;
    ?>
    <html>
    <head>
    <title>Вход в систему</title>
    </head>
    <body>
    <form method="post" action="<?php echo "$PHP_SELF"; ?>">
        <div align="center"><center>
            <h3>Для просмотра запрашиваемой страницы необходимо
                зарегистрироваться.</h3>
            <table border="1" width="200" cellpadding="2">
                <tr>
                    <th width="18%" align="right" nowrap>Идентификатор</th>
                    <td width="82%" nowrap>
                        <input type="text" name="userid" size="8">
                    </td>
                </tr>
                <tr>
                    <th width="18%" align="right" nowrap>Пароль</th>
                    <td width="82%" nowrap>
                        <input type="password" name="userpassword" size="8">
                    </td>
                </tr>
                <tr>
                    <td width="100%" colspan="2" align="center" nowrap>
                        <input type="submit" value="Отправить" name="login">
                    </td>
                </tr>
            </table>
        </center></div>
    </form>
    </body>
    </html>
    <?
    }
```

Основа сценария находится в функции `auth_user()`, которая принимает два аргумента — переменные `$userid` и `$userpassword` — и сверяет их значения со значениями, сохраненными в таблице `user`:

```
function auth_user($userid, $userpassword)
{
    global $default_dbname, $user_tablename;

    $link_id = db_connect($default_dbname);
```

После подключения к базе данных выбирается запись, в которой значения полей `userid` и `password` соответствуют значениям, введенным пользователем, и извлекается значение поля `username` этой записи:

```
$query = "SELECT username FROM $user_tablename WHERE userid = '$userid'
          AND userpassword = password('$userpassword')";
$result = mysql_query($query);
```

Если ни в одной из записей нет обоих совпадающих критериев, то возвращается 0; в противном случае выбирается и возвращается имя пользователя:

```
if(!mysql_num_rows($result)){
    return 0;
}else{
    $query_data = mysql_fetch_row($result);
    return $query_data[0];
}
```

Необходимо удостовериться, что значение \$userpassword зашифровано *до* сравнения со значением в таблице user, для того чтобы сравнивать два зашифрованных значения, возвращенных одним и тем же процессом, и, следовательно, идентичных. Не все типы шифрования позволяют это сделать, но алгоритм шифрования в MySQL является односторонним. Иначе говоря, две идентичных строки при данном алгоритме всегда дают две идентично зашифрованные строки.

На рис. 11.9 показано, как должен выглядеть интерфейс.

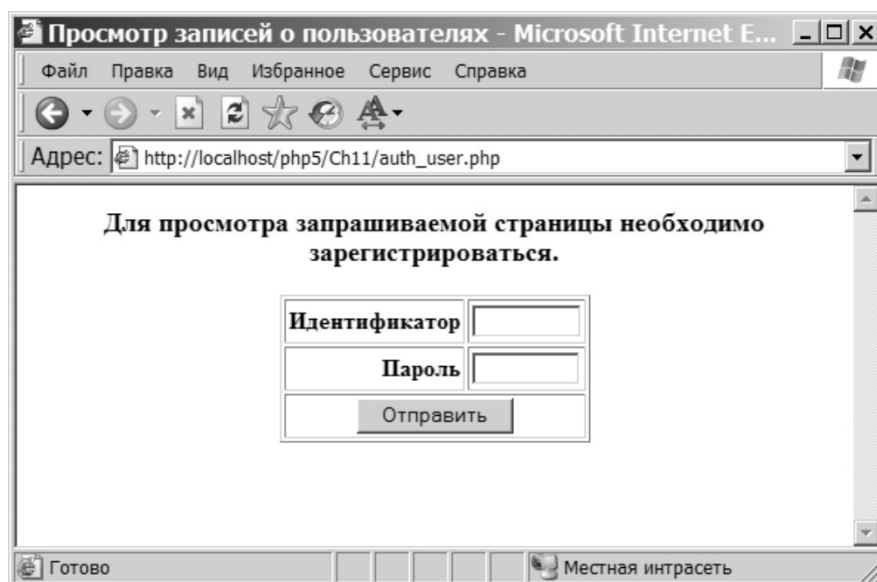


Рис. 11.9.

Когда пользователь успешно входит в систему, появляется приветствие (рис. 11.10).

Файл access_logger.php

Последний сценарий можно улучшить так, чтобы он авторизовал пользователей и протоколировал их визиты на Web-страницы сайта. Назовем эту модифицированную версию access_logger.php. Регистрировать посещения страниц следует путем автоматического подключения файла access_logger.php в начало всех Web-страниц сайта. Это можно сделать очень просто: установив параметр auto_prepend_file в конфигурационном файле php.ini равным абсолютному пути к данному сценарию. Например:

```
auto_prepend_file = /home/james/access_logger.php
```

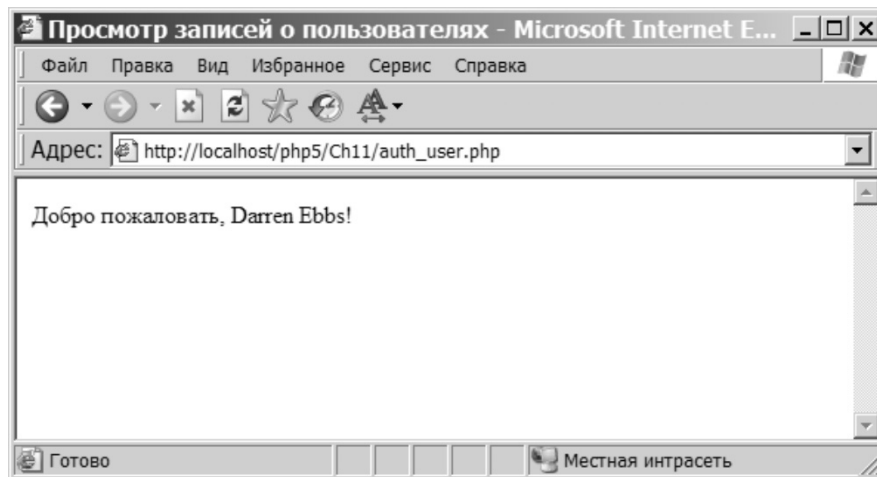



Рис. 11.10.

Другой вариант состоит в подключении протоколирующего сценария в начало каждой Web-страницы, находящейся в защищенной паролем области сайта. Сценарий должен подключаться в начале каждой страницы, до отправки каких-либо HTML-тегов или текста, в противном случае функция `header()` не будет работать.

Предположим, что используется первый подход, и для подключения сценария ко всем `.php`-страницам на сервере устанавливается параметр `auto_prepend_file`. Чтобы исключить в таком случае некоторые файлы и каталоги из процесса аутентификации, можно перечислить их соответственно в двух массивах. Когда пользователь пытается зайти на какую-либо страницу, сценарий протоколирования может сверять имя страницы и путь к ней с элементами массивов и требовать аутентификации только в том случае, если совпадений не найдено.

Начнем с подключения файла `common_db.inc` и определения массивов исключений и имен таблиц:

```
<?php
require_once ('common_db.inc');
$exclude_dirs = array('/info', '/contact');

$exclude_files = array('index.html', 'info.html', 'register.php');
$user_tablename = 'user';
$access_log_tablename = 'access_log';
$PHP_SELF = $_SERVER['PHP_SELF'];
```

Когда пользователь регистрируется на сайте, можно начать сеанс и зарегистрировать в качестве переменной сеанса идентификатор этого пользователя. Впоследствии идентификатор и пароль пользователя будут использоваться в функции `do_authenticate()`. При первой инициализации сеанса и регистрации его переменных переменные для имени пользователя и пароля не содержат значений, поэтому необходимо проверить, записана ли страница в список исключений. Если страница в списке исключений не значится, то вызывается функция `do_authenticate()`:

```
session_start();
session_register('userid', 'userpassword');
if (!$_SESSION['userid']){
```

```

$filepath = dirname($_SERVER['SCRIPT_FILENAME']);
$filename = basename($_SERVER['SCRIPT_FILENAME']);
if($filepath == ""){
    $filepath = '/';
}

$auth_done = 0;

for($j=0; $j < count($exclude_dirs); $j++) {
    if($exclude_dirs[$j] == $filepath){
        break;
    }else{
        for($i=0; $i< count($exclude_files); $i++) {
            if($exclude_files[$i] == $filename){
                break;
            }
            if ($i == (count($exclude_files) - 1)){
                do_authentication();
                $auth_done = 1;
                break;
            }
        }
    }
}
if($auth_done){
    break;
}
}
?>

```

Рассмотрим отслеживание посещений страниц пользователем, который уже зарегистрировался на сайте и передал свой идентификатор и пароль. Снова вызывать функцию `do_authenticate()`, конечно, нежелательно. Однако известно, что если пользователь успешно зарегистрировался на сайте, то переменные сеанса, в которых хранится идентификатор и пароль пользователя, проинициализированы, поэтому можно использовать следующий код:

```

if ($_SESSION['userid'] && $_SESSION['userpassword']){
    $userid = $_SESSION['userid'];
    $filename = basename($_SERVER['SCRIPT_FILENAME']);
    $link_id = db_connect($default_dbname);

    $query = "SELECT userid FROM $access_log_tablename
    WHERE page = '$filename' AND userid = '$userid'";
    $result = mysql_query($query);

    if(!mysql_num_rows($result)){
        $query = "INSERT INTO $access_log_tablename
        VALUES ('$filename', '$userid', 1, NULL)";
    }else{
        $query = "UPDATE $access_log_tablename
        SET visitcount = visitcount + 1, accessdate = NULL
        WHERE page = '$filename' AND userid = '$userid'";
    }

    mysql_query($query);

    $num_rows = mysql_affected_rows($link_id);
    if($num_rows != 1){
        die(sql_error());
    }
}

```

Теперь по мере того как пользователи будут перемещаться по сайту, все посещения страниц будут протоколироваться.

Рассмотрим объявление функции, которая проверяет отправку пользователем своего идентификатора. Если пользователь не передал идентификатор, то необходимо вызывать функцию `login_form()`, ту же самую, которая использовалась в `auth_user.php` (нужно просто перенести код этой функции в данный файл; здесь ее код не показывается):

```
function do_authentication() {
    global $default_dbname, $user_tablename, $access_log_tablename;
    global $MYSQL_ERROR, $MYSQL_ERRNO;
    global $filename;
    global $PHP_SELF;

    if(!isset($_POST['userid'])) {
        login_form();
        exit;
    }
```

Если сценарий получил переданные пользователем идентификатор и пароль, то эти значения используются в запросе к базе данных. Здесь также определяются переменные сеанса. Переменные сеанса сохраняются между запросами страниц.

```
}else {
    $_SESSION['userpassword'] = $_POST['userpassword'];
    $_SESSION['userid'] = $_POST['userid']; }
$userid = $_POST['userid'];
$userpassword = $_POST['userpassword'];
$link_id = db_connect($default_dbname);
$query = "SELECT username FROM $user_tablename
    WHERE userid = '$userid' AND userpassword = password('$userpassword')";
$result = mysql_query($query);
}
```

Если запрос был выполнен неудачно (например, если пользователь вводит только свое имя), то сеанс аннулируется и пользователю выводится сообщение о том, что он не прошел авторизацию:

```
if(!mysql_num_rows($result)) {
    session_unregister("userid");
    echo "Попытка авторизации неудачна. " .
        "Вы должны ввести допустимый идентификатор пользователя и пароль. " .
        "Чтобы попытаться снова, щелкните следующую ссылку.<BR>\n";
    echo "<A HREF=\"$PHP_SELF\"> Вход в систему</A><BR>";
    echo "Если Вы еще не зарегистрированы, для регистрации щелкните следующую ссылку." .
        "<BR>\n";
    echo "<A HREF= \"register.php\">Зарегистрироваться</A>";
    exit;
}
```

В случае успешного выполнения запроса необходимо проверить, посещал ли пользователь данную страницу раньше, и либо вставить, либо обновить соответствующие значения в таблице:

```
}else{
    $query = "SELECT userid FROM $access_log_tablename
    WHERE page = '$filename' AND userid = '$userid'";
    $result = mysql_query($query);
    if(!mysql_num_rows($result)) {
        $query = "INSERT INTO $access_log_tablename
        VALUES ('$filename', '$userid', 1, NULL)";
    }
```

```

    }else{
        $query = "UPDATE $access_log_tablename
        SET visitcount = visitcount + 1, accessdate = NULL
        WHERE page = '$filename' AND userid = '$userid'";
    }
    mysql_query($query);

    $num_rows = mysql_affected_rows($link_id);
    if($num_rows != 1) die(sql_error());
}
}

```

Если вызвать этот сценарий в браузере, то будет выведена пустая страница (предполагается, что пользователь вводит корректный идентификатор и пароль). Однако если изучить информацию в таблице `access_log`, то станет ясно, что это посещение было запротоколировано. В приведенном ниже примере администратор сайта открыл страницу `access_logger.php`, чтобы убедиться в ее работоспособности:

```
mysql> select * from access_log;
```

page	userid	visitcount	accessdate
stats.html	Brian	1	20040304143102
score.html	Pads	2	20040312160114
refs.html	Nicrot	5	20040124122744
log.html	Nicrot	3	20040124122642
log.html	Greeny	4	20040124112654
access_logger.php	Dodge	1	20040314192430

```
6 rows in set (0.00 sec)
```

Создание сценария для управления пользователями

Наконец, можно собрать все примеры сценариев вместе и усовершенствовать сценарий просмотра записей (созданный в предыдущей главе) так, чтобы он позволял управлять данными в связанных таблицах. Используя следующий сценарий `userman.php`, можно редактировать и удалять регистрационные записи пользователей и/или соответствующие записи о посещении страниц.

Файл `userman.php`

Данный сценарий начинается, как и `register.php`, с выборки вариантов для ENUM-поля `userposition`:

```

<?php
//userman.php
include_once "../common_db.inc";

$link_id = db_connect();
mysql_select_db("sample_db");
$position_array = enum_options('userposition', $link_id);
mysql_close($link_id);

```

Функция user_message()

Функция `user_message()` сообщает о результатах заданной операции. Если ей был передан необязательный аргумент URL, то она загружает определенную страницу:

```
function user_message($msg, $url = "")
{
    html_header();

    if (empty($url)) {
        echo "<SCRIPT>alert(\"$msg\");history.go(-1)</SCRIPT>";
    } else {
        echo "<SCRIPT>alert(\"$msg\");self.location.href='$url'</SCRIPT>";
    }

    html_footer();
    exit;
}
```

Функция list_records()

В переработанной функции `list_records()` появляется возможность удалять определенную запись. Остальная часть функции такая же, как и в предыдущей ее версии (в главе 10):

```
function list_records() {
...
    echo "<td WIDTH=\"25%\" ALIGN=\"CENTER\">
        <a href=\"javascript:open_window('$PHP_SELF?action=view_record&
                                     userid=$userid');\">
            Просмотреть запись</a>

        <a href=\"$PHP_SELF?action=delete_record&userid=$userid\"
            onClick=\"return confirm('Вы уверены?');\">
            Удалить запись</a>

    </td>\n";
    echo "</tr>\n";
...
}
```

Функция delete_record()

Функция `delete_record()` удаляет запись определенного пользователя из таблицы `user`, а также соответствующие ей записи из таблицы `access_log`:

```
function delete_record()
{
    global $default_dbname, $user_tablename, $access_log_tablename;
    $userid = $_GET['userid'];

    if (empty($userid)) {
        error_message('Введите идентификатор пользователя!');
    }

    $link_id = db_connect($default_dbname);
    if (!$link_id) {
        error_message(sql_error());
    }

    $query = "DELETE FROM $user_tablename WHERE userid = '$userid'";
    $result = mysql_query($query);
    if (!$result) {
        error_message(sql_error());
    }
}
```

```

    }

    $num_rows = mysql_affected_rows($link_id);
    if($num_rows != 1){
        error_message("Пользователя $userid не существует");
    }

    $query = "DELETE FROM $access_log_tablename WHERE userid = '$userid'";
    $result = mysql_query($query);

    user_message("Все записи, относящиеся к пользователю $userid удалены!");
}

```

Функция edit_record()

Функция edit_record() обновляет запись определенного пользователя в таблице user. Если изменяется идентификатор пользователя, то она также обновляет связанные записи в таблице access_log, отражая это изменение:

```

function edit_record()
{
    global $default_dbname, $user_tablename, $access_log_tablename;
    $PHP_SELF = $_SERVER['PHP_SELF'];
    $userid = $_GET['userid'];
    $newuserid = $_GET['new_userid'];
    $username = $_GET['username'];
    $userpassword = $_GET['userpassword'];
    $userposition = $_GET['userposition'];
    $useremail = $_GET['useremail'];
    $userprofile = $_GET['userprofile'];

    if(empty($userid)){
        $userid = $_GET['new_userid'];
    }

    $link_id = db_connect($default_dbname);
    if(!$link_id){
        error_message(sql_error());
    }

    $field_str = "

```

Действительно, значение поля userid тоже можно изменить. С точки зрения администраторов баз данных, как правило, недопустимо давать пользователю возможность изменять свой идентификатор, так как он (идентификатор) выполняет функции номера социального страхования (Social Security number) — он уникален и одновременно неизменен. Тем не менее, если бы пользователь настаивал на изменении своего идентификатора, то пришлось бы изменить каждую запись в каждой связанной таблице независимо от количества этих записей, или, в конце концов, в базе данных накопилось бы множество бесполезных записей, не имеющих владельца.

Новый идентификатор пользователя следует сравнить с существующим и в случае необходимости обновить переменную \$field_str:

```

if($userid != $new_userid) $field_str = " userid = '$newuserid', ";

```

Если переменная \$userpassword содержит какое-либо значение, то необходимо обновить поле userpassword:

```

if(!empty($userpassword)) {
    $field_str .= " userpassword = password('$userpassword'), ";
}

```

Подобная проверка не позволит администратору случайно оставить в поле `userpassword` пустое значение. Затем необходимо обновить остальные поля и проверить, все ли идет по плану:

```
$field_str .= " username = '$username', ";
$field_str .= " userposition = '$userposition', ";
$field_str .= " useremail = '$useremail', ";
$field_str .= " userprofile = '$userprofile'";

$query = "UPDATE IGNORE $user_tablename SET $field_str WHERE userid = '$userid'";
$result = mysql_query($query);
if (!$result) {
    error_message(sql_error());
}

$num_rows = mysql_affected_rows($link_id);
if (!$num_rows) {
    error_message("Ни одна запись не изменена!");
}
```

Следует отметить использование в данном запросе оператора `IGNORE`. Поле `userid` является частью первичного ключа и MySQL препятствует его изменению, а оператор `IGNORE` позволяет продолжить выполнение запроса и внести изменения. Кроме того, чтобы база данных была согласованной, для отражения изменений необходимо обновить все остальные таблицы.

Если значение поля `userid` изменяется, то функция `edit_record` соответствующим образом обновляет все связанные записи в таблице `access_log`:

```
if ($userid != $new_userid) {
    $query = "UPDATE $access_log_tablename SET userid = '$newuserid'
    WHERE userid = '$userid'";
    $result = mysql_query($query);
    if (!$result) {
        error_message(sql_error());
    }

    user_message("Все записи, связанные с $userid, были изменены!",
        "$PHP_SELF?action=view_record&userid=$newuserid");
} else {
    user_message("Все записи, связанные с $userid, были изменены!",
        "$PHP_SELF?action=view_record&userid=$userid");
}
}
```

Если поле `userid` было изменено, то вызывается функция `user_message()`, второй аргумент которой определяет часть URL `userid=$new_userid`, так как возвращение в сценарий со старым значением `userid` привело бы к неожиданному результату — выводу формы для редактирования данных несуществующего пользователя.

Функция `edit_log_record()`

Функция `edit_log_record()` обновляет в таблице `access_log` записи о посещениях страниц для заданного пользователя:

```
function edit_log_record() {
    global $default_dbname, $access_log_tablename;
    $userid = $_GET['userid'];
    $newpage = $_GET['new_page'];
    $visitcount = $_GET['visitcount'];
```

```

$accessdate = $_GET['accessdate'];
$orgpage = $_GET['org_page'];
$PHP_SELF = $_SERVER['PHP_SELF'];

if(empty($userid)){
    error_message('Введите идентификатор пользователя!');
}

$link_id = db_connect($default_dbname);
if(!$link_id){
    error_message(sql_error());
}

$field_str = "";

$field_str .= " page = '$newpage', ";
$field_str .= " visitcount = '$visitcount', ";
$field_str .= " accessdate = '$accessdate' ";

```

Данная функция работает почти так же, как функция `edit_record()`. Основное отличие заключается в том, что функция `edit_log_record()` использует для выбора соответствующих записей таблицы `access_log` два поля: `userid` и `page`.

Существующее значение поля `page` сохраняется в переменной `$org_page` на случай, если администратор изменит значение этого поля в форме для редактирования:

```

$query = "UPDATE $access_log_tablename SET $field_str WHERE userid = '$userid'
        AND page = '$orgpage'";

$result = mysql_query($query);
if(!$result){
    error_message(sql_error());
}

```

Если ни одна запись не обновилась, то это значит, что администратор нажал кнопку Отправить, ничего не изменив в соответствующей форме для редактирования.

Функция `mysql_affected_rows()` возвращает 0, так как в результате предыдущей UPDATE-операции ни одна запись не изменилась. Команда UPDATE не выполняет никаких изменений, если новая запись содержит те же значения, что и существующая:

```

$num_rows = mysql_affected_rows($link_id);
if(!$num_rows){
    error_message("Данные не изменены!");
}

user_message("Все записи, относящиеся к пользователю $userid, были изменены!",
            "$PHP_SELF?action=view_record&userid=$userid");
}

```

Функция view_record()

Усовершенствованная версия функции `view_record()` позволяет администратору редактировать пользовательские записи:

```

function view_record() {
    global $default_dbname, $user_tablename, $access_log_tablename;
    global $position_array;    $userid = $_GET['userid'];
    $PHP_SELF = $_SERVER['PHP_SELF'];

    if(empty($userid)){
        error_message('Введите идентификатор пользователя!');
    }
}

```



```

}

$link_id = db_connect($default_dbname);

if(!$link_id){
    error_message(sql_error());
}
$query = "SELECT usernumber, userid, username, userposition, useremail,
userprofile FROM $user_tablename WHERE userid = '$userid'";
$result = mysql_query($query);
if(!$result){
    error_message(sql_error());
}

$query_data = mysql_fetch_array($result);
$usernumber = $query_data["usernumber"];
$userid = $query_data["userid"];
$username = $query_data["username"];
$userposition = $query_data["userposition"];
$useremail = $query_data["useremail"];
$userprofile = $query_data["userprofile"];

```

Наконец, отображается несколько форм, с помощью которых администратор может редактировать пользовательские записи и данные о посещении страниц:

```

html_header();
echo "<center><H3>
Запись для пользователя №. $usernumber - $userid($username)
</h3></center>";
?>

<form method="get" action="<?php echo $PHP_SELF ?>">
<input type="hidden" name="action" value="edit_record">
<input type="hidden" name="userid" value="<? echo $userid ?>">
<div align="center"><center>
<table border="1" width="90%" cellpadding="2">
    <tr>
        <th width="30%" nowrap>Идентификатор</th>

```

Скрытое поле `new_userid` используется в случае, если администратор изменяет значение поля `userid`:

```

<td width="70%">
    <input type="text" name="new_userid"
        value="<?php echo $userid ?>"
        size="8" maxlength="8"></td>
</tr>
<tr>
    <th width="30%" nowrap>Пароль</th>

```

Зашифрованный пароль не выводится, так как он не используется:

```

<td width="70%"><input type="text" name="userpassword" size="15"></td>
</tr>
<tr>
    <th width="30%" nowrap>Полное имя</th>
    <th width="70%"><input type="text" name="username"
        value="<?php echo $username ?>" SIZE="20"></td>
</tr>
<tr>
    <th width="30%" nowrap>Позиция</th>
    <td width="70%"><select name="userposition" size="1">
<?php

```

Для создания выпадающего списка возможных позиций используется переменная `$position_array`:

```
for($i=0; $i < count($position_array); $i++) {
    if(!isset($userposition) && $i == 0) {
        echo "<OPTION SELECTED VALUE=\"". $position_array[$i] . "\">" .
            $position_array[$i] . "</OPTION>\n";
    }else if($userposition == $position_array[$i]) {
        echo "<OPTION SELECTED VALUE=\"". $position_array[$i] . "\">" .
            $position_array[$i] . "</OPTION>\n";
    }else{
        echo "<OPTION VALUE=\"". $position_array[$i] . "\">" .
            $position_array[$i] . "</OPTION>\n";
    }
}
?>
</select></td>
</tr>
<tr>
    <th width="30%" nowrap>E-mail-адрес</th>
    <td width="70%"><input type="text" name="useremail" size="20"
        value="<?php echo $useremail ?>"></td>
</tr>
<tr>
    <th width="30%" nowrap>Профиль</th>
```

Функция `htmlspecialchars()` гарантирует, что любые специальные HTML-символы в переменной `$userprofile` выводятся в виде HTML-последовательностей и не способны нарушить окружающую разметку:

```
<td width="70%">
    <textarea rows="5" cols="40" name="userprofile">
        <?php echo htmlspecialchars($userprofile) ?>
    </textarea>
</td>
</tr>
<tr>
    <th width="100%" colspan="2" nowrap>
        <input type="submit" value="Записать">
        <input type="reset" value="Очистить">
    </th>
</tr>
</table>
</center></div>
</form>
<?php
    echo "<HR SIZE=\"2\" WIDTH=\"90%\">\n";
```

Каждая запись из таблицы `access_log` представляется в отдельной форме:

```
$query = "SELECT page, visitcount, accessdate, date_format(accessdate, '%M, %e, %Y') as formatted_accessdate FROM $access_log_tablename WHERE userid = '$userid'";
$result = mysql_query($query);

if(!$result){
    error_message(sql_error());
}
if(mysql_num_rows($result)){
    echo "<center>У пользователя $userid ($username) нет записей о посещении страниц.</center>";
}else{
    echo "<center>Записи о посещении страниц пользователем $userid ($username).</center>";
}
```

```

    }
?>
<div align="center"><center>
<table border="1" width="90%" cellpadding="2">
  <tr>
    <th width="20%" nowrap>Страница</th>
    <th width="20%" nowrap>Посещения</th>
    <th width="30%" nowrap>Последнее посещение</th>
    <th width="30%" nowrap>Действие</th>
  </tr>
</table>
<?php

```

Результаты данного запроса выводятся в хорошо отформатированной таблице, которую администратор может использовать для изменения access_log-значений:

```

while($query_data = mysql_fetch_array($result)) {
    $page = $query_data["page"];
    $visitcount = $query_data["visitcount"];
    $accessdate = $query_data["accessdate"];
    $formatted_accessdate = $query_data["formatted_accessdate"];

    echo "<FORM METHOD=\"GET\" ACTION=\"$_PHP_SELF\">";
    echo "<INPUT TYPE=\"HIDDEN\" NAME=\"action\"VALUE=\"edit_log_record\">";
    echo "<INPUT TYPE=\"HIDDEN\" NAME=\"userid\" VALUE=\"$userid\">";
    echo "<INPUT TYPE=\"HIDDEN\" NAME=\"org_page\" VALUE=\"$page\">";
    echo "<TR>\n";
    echo "<TD WIDTH=\"20%\"><INPUT TYPE=\"TEXT\"NAME=\"new_page\" SIZE=\"30\"";
    echo "VALUE=\"$page\"></TD>\n";
    echo "<TD WIDTH=\"20%\" ALIGN=\"CENTER\">";
    echo "<INPUT TYPE=\"TEXT\" NAME=\"visitcount\" SIZE=\"3\"";
    echo "VALUE=\"$visitcount\"></TD>\n";
    echo "<TD WIDTH=\"30%\" ALIGN=\"CENTER\">";
    echo "<INPUT TYPE=\"TEXT\" NAME=\"accessdate\" SIZE=\"14\"";
    echo "MAXLENGTH=\"14\" VALUE=\"$accessdate\">";
    echo "<BR>$formatted_accessdate</TD>\n";
    echo "<TD WIDTH=\"30%\" ALIGN=\"CENTER\">";
    echo "<INPUT TYPE=\"SUBMIT\" VALUE=\"Записать\">";
    echo "<INPUT TYPE=\"RESET\" VALUE=\"Очистить\"></TD>\n";
    echo "</TR>\n";
    echo "</FORM>\n";
}
?>
</tr>
</table>
</center></div>
<?php
    html_footer();
}

```

Выбор действия

Наконец, чтобы определить, какая функция должна быть вызвана далее, используется переменная \$action:

```

if (empty($_GET['action'])) {
    $_GET['action'] = "";
}
switch($_GET['action']) {
    case "edit_record":
        edit_record();
        break;
    case "edit_log_record":
        edit_log_record();
        break;
}

```

```

case "delete_record":
    delete_record();
    break;
case "view_record":
    view_record();
    break;
default:
    list_records();
    break;
}
?>

```

Предположим, что администратору нужно вывести список записей, отсортированных по номеру пользователя. На рис. 11.11 показана страница после нажатия на заголовок соответствующего поля:

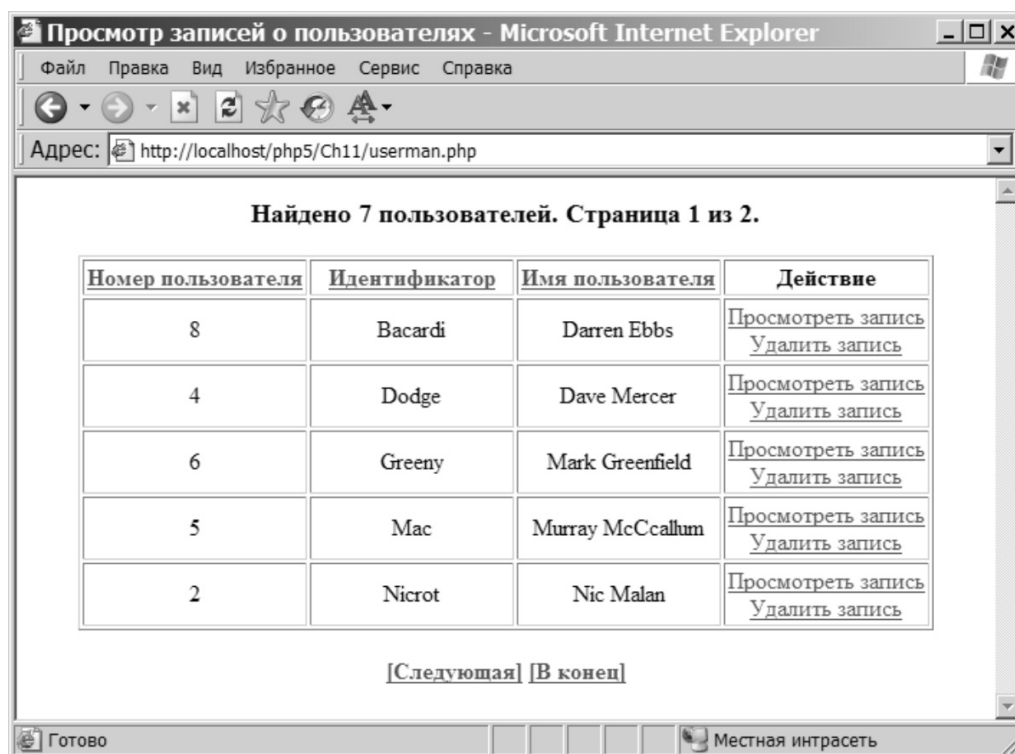


Рис. 11.11.

Если щелкнуть на ссылке [Просмотреть запись](#), то откроется новое окно, в котором будут показаны записи, относящиеся к пользователю, связанному с данной ссылкой (рис. 11.12).

Просмотр записей о пользователях - Microsoft Internet Explorer

Файл Правка Вид Избранное Сервис Справка

Адрес: http://localhost/php5/Ch11/userman.php?action=view_record&%20

Ссылки Windows »

Запись для пользователя №. 2 - Nicrot(Nic Malan)

Идентификатор	Nicrot
Пароль	
Полное имя	Nic Malan
Позиция	Mid
E-mail-адрес	therot@mywebsiteaddr
Профиль	Высококласный нападающий

Записать Очистить

Записи о посещении страниц пользователем Nicrot (Nic Malan).

Страница	Посещения	Последнее посещение	Действие
/log.html	3	2005-12-08 20:41:0 December, 8, 2005	Записать Очистить
/refs.html	2	2005-12-08 20:38:4 December, 8, 2005	Записать Очистить

Готово Местная интрасеть

Рис. 11.12.

Резюме

В данной главе рассматривался процесс управления информацией, имеющейся в таблицах базы данных, создание сценария для регистрации пользователей и сценария для протоколирования посещения страниц, которые можно использовать совместно для отслеживания активности пользователей на Web-сайте. Кроме того, материал данной главы наглядно показывает, насколько просто в PHP можно реализовать аутентификацию пользователей, используя возможности управления сессиями.

Серия глав, описывающих MySQL, заканчивается примером сценария, который позволяет управлять пользовательскими записями. MySQL в качестве сервера для хранения данных позволяет создавать высокопроизводительные приложения, размах которых ограничен только воображением разработчика.

Упражнения

1. Создайте список подходящих дескрипторов для каждого столбца в таблице. *Совет: помните о том, что ресторанам ежедневно приходится иметь дело с множеством отдельных заказов (счетов):*

Имя	Идентификатор заказа	Идентификатор пользователя	Ресторан	Пароль	Фамилия	Всего (\$)	E-mail
David	2	1	Nandos	12345	Mercer	21.45	davidm@contechst.com
David	4	1	Mimos	12345	Mercer	20.95	davidm@contechst.com
Nic	3	2	St Elmos	23212	Malan	15.45	therot@doggiestouch.co.za
Brian	5	4	Spur	32123	Reid	22.00	pads@doggiestouch.co.za
Darren	1	3	Home	43212	Ebbs	11.85	Bacardi@doggiestouch.co.za

2. Администратор сайта решил создать страницу, на которой пользователи смогут просматривать свои предыдущие заказы и итоговые суммы. Используя принципы нормализации, создайте новые таблицы, в каждой из которых будет представлен один элемент. Присвойте каждой таблице подходящее имя и определите первичный ключ. (Проверьте, уменьшат ли эти изменения избыточность при больших объемах данных.)
3. Создайте данные таблицы с помощью SQL-операторов.
4. После полугодового использования Web-сайта клиенты отметили, что сайт начинает работать медленнее, когда они пытаются просмотреть свои предыдущие заказы. Администратор сайта изучил таблицу заказов и выяснил, что в ней теперь хранится несколько тысяч записей. Почему сайт работает медленно и как администратор может повысить скорость работы сайта? Обновите таблицы соответствующим образом.
5. Директор решил узнать, насколько популярен среди клиентов ресторан “Nandos”. Какой запрос можно использовать, чтобы получить общие суммы по всем заказам: а) сделанным одним клиентом и б) в целом по ресторану “Nandos”. *Совет: попытайтесь использовать запрос SELECT SUM (столбец).*
6. Создайте сценарий, с помощью которого пользователи во время регистрации на сайте смогут вводить в базу данных информацию о себе. База данных должна быть настроена так, чтобы идентификаторы пользователей назначались авто-

матически. Сразу после регистрации пользователь должен иметь возможность разместить заказ в любом из пяти ресторанов. (Для начала только в одном. Беспокоиться о создании завершенного интерфейса не следует — важна возможность вводить цену.) После нажатия на кнопку Заказать в таблице заказов должна создаваться новая запись с увеличенным на 1 идентификатором заказа (`Order_ID`) и корректным идентификатором пользователя (`User_ID`). *Совет: желательно использовать сеансы для хранения идентификаторов пользователей (`User_ID`) при регистрации.*

12

Введение в объектно-ориентированное программирование

Объектно-ориентированная разработка программного обеспечения может быть сложной темой для программистов, создающих преимущественно процедурный код. В данной главе рассматривается некоторая базовая теория, лежащая в основе объектно-ориентированного подхода, а также многосложная (иногда обескураживающая) терминология. Вы узнаете, почему объектно-ориентированные методики интересны многим программистам, как на практике можно увеличить скорость разработки сложных приложений и как легко их можно модифицировать.

В следующей главе изучается тема реализации этих методик в PHP5, а также показана их эффективность на примере разработки реального приложения.

Что такое объектно-ориентированное программирование?

Объектно-ориентированное программирование (Object-oriented programming, OOP) представляет собой особый способ мышления при создании приложений. Объекты позволяют разработчику более тщательно моделировать сущности реального мира, а также процессы и идеи, которые должны поддерживаться разрабатываемым приложением. Вместо того, чтобы представлять приложение как поток управления, который передает блоки данных от одной функции к другой, OOP-подход позволяет моделировать приложение в виде множества сотрудничающих объектов, которые независимо выполняют определенные действия.

Например, при постройке здания водопроводчики занимаются трубопроводами, а электрики электрической проводкой. Водопроводчикам не нужно знать, на какой

ток рассчитана проводка в спальне. Они заботятся только о своих задачах. Генеральный конструктор гарантирует, что все субподрядчики выполняют работу, которую необходимо закончить, но не обязательно интересуются подробностями чужих задач. ООР-подход очень похож на подход, применяемый при строительстве зданий, тем, что каждый объект скрывает от остальных объектов детали своей реализации. То, как он выполняет свою работу, не касается остальных компонентов системы. Значение имеют только службы, которые способен предоставить объект.

Понятия классов и объектов, а также способы, посредством которых можно использовать эти идеи в разработке программного обеспечения, представляют собой фундаментальные основы объектно-ориентированного программирования. ОО-программирование в некотором смысле противоположно процедурному программированию, т.е. программированию с использованием функций и глобальных структур данных. Как будет показано ниже, объектно-ориентированный подход имеет значительные преимущества перед процедурным программированием, а, кроме того, новая реализация объектно-ориентированных возможностей в PHP5 также дает значительный выигрыш в производительности.

Рассмотрим некоторые из огромного количества преимуществ, которые дает разработчикам ОО-подход.

Первое преимущество — легкость, с которой можно воплотить требования бизнеса в модулях кода. Так как объектно-ориентированный подход позволяет моделировать приложение, опираясь на идею объектов реального мира, часто можно создавать непосредственное отображение людей, объектов и понятий в PHP-классы. Такие классы имеют те же свойства и поведение, что и представленные ими сущности реального мира, это позволяет разработчику быстро определить, какой код нужно писать и как должны взаимодействовать различные части приложения.

Второе преимущество ОО-программирования заключается в повторном использовании кода. В разных частях одного приложения часто требуется использовать одни и те же типы данных. Например, в приложении, которое позволяет управлять записями пациентов в больнице, определенно потребуется класс `Person`. В процесс лечения вовлекается множество человек — пациенты, доктора, медсестры, администраторы, страховые агенты и т.д. На каждом этапе лечения пациента в его учетные записи необходимо вносить сведения о том, кто проводил определенное действие (например, выписку рецепта, обработку ран или отправку счета страховой компании), и проверять, имеет ли данное лицо право выполнять это действие. Определив класс `Person`, заключающий в себе все свойства и методы, общие для всех этих людей, разработчик получает возможность очень широкого применения повторно используемого кода, что не всегда возможно в процедурном программировании.

А кроме того, сколько можно создать приложений, в которых на определенном этапе обрабатывается информация об отдельных людях? Вероятно, довольно много. Хорошо написанный класс `Person` можно было бы легко с минимальными изменениями или вообще без изменений скопировать из одного проекта в другой, и таким образом, немедленно получить ранее созданную развитую функциональность для обработки информации о людях. Это одно из наибольших преимуществ ОО-подхода — возможность повторного использования кода не только внутри одного приложения, но и в различных проектах.

Еще одно преимущество ОО-программирования связано с модульной организацией классов. Если в классе `Person` обнаруживается дефект или требуется добавить или изменить работу функций класса, то это необходимо сделать только в одном месте.

Вся функциональность класса содержится в одном РНР-файле. Изменение класса `Person` немедленно повлияет на все зависимые от этого класса процессы приложения, что значительно упрощает поиск и устранение ошибок и делает добавление новых функций сравнительно безболезненной задачей.

В сложной программной архитектуре преимущества модульности могут оказаться неопределимыми. Однажды автору пришлось работать над проектом с более чем 200 000 строк процедурного РНР-кода. Около 65 процентов времени, потраченного на устранение ошибок, пошло на поиск расположения некоторых функций и определение того, какие данные взаимодействуют с той или иной функцией. Результатом воссоздания этого программного обеспечения в ОО-архитектуре стало резкое уменьшение количества кода, что в свою очередь означало не только уменьшение объемов работы (если бы это ставилось на первое место), но и меньшее количество ошибок (чем меньше кода, тем меньше источников проблем), а также более короткий цикл их устранения.

Поскольку ОО-подход заставляет разработчика продумывать организацию кода, новым членам коллектива разработчиков впоследствии будет гораздо проще понять структуру существующего приложения; появляется оболочка, которая показывает расположение новых функций.

В крупных проектах часто задействованы большие коллективы разработчиков, обычно состоящие из программистов с различным уровнем подготовки. И снова очевидны преимущества ОО-подхода перед процедурным программированием. Объекты скрывают от пользователей детали своей реализации. Вместо необходимости разбираться в сложных структурах данных и всех тонкостях бизнес-логики новички могут, пользуясь лишь краткой документацией, начать использование объектов, созданных ведущими членами команды. Сами объекты отвечают за изменение данных или состояния системы.

Если бы в упомянутом выше крупном приложении использовался бы процедурный код и дальше, то достаточное для продуктивной работы ознакомление новых членов коллектива с приложением заняло бы до двух месяцев. Как только данное программное обеспечение было переработано с использованием объектно-ориентированного кода, новые члены команды смогли вносить в код существенные дополнения уже через несколько дней. Они могли быстро использовать даже самые сложные объекты, потому что для этого не нужно было полностью понимать детали реализации содержащейся в данных объектах функциональности.

Теперь читатель хорошо представляет себе цель использования ОО-принципов в качестве основного метода программирования. В последующих разделах предлагается более подробный разбор фундаментальных понятий, лежащих в основе объектно-ориентированного программирования. В двух следующих главах читатель, вероятно, найдет для себя полезные идеи и познакомится с преимуществами ОО-подхода.

Основные понятия ОО-программирования

В данном разделе представлены элементарные понятия объектно-ориентированного программирования, а также их взаимосвязь. (В главе 13 рассматривается специфическая реализация этих понятий в РНР5.) Далее рассматриваются:

- ❑ классы, которые являются “чертежами” для объектов и фактическим кодом, определяющим свойства и методы этих объектов;
- ❑ объекты, которые представляют собой работающие экземпляры класса и содержат все внутренние данные и информацию о состоянии, необходимую для работы приложения;

- ❑ наследование, которое позволяет определять один класс как подвид другого класса (почти так же, как квадрат является подвидом прямоугольника);
- ❑ интерфейсы, представляющие собой наборы четко определенных условий для выполнения несвязанными объектами одной общей функции;
- ❑ инкапсуляция, которая представляет собой способность объекта ограничивать доступ к своим внутренним данным.

Попутно рассматривается полиморфизм, позволяющий определять класс как член одного или нескольких категорий классов (так же как автомобиль — это “объект, имеющий двигатель” и одновременно “объект, имеющий колеса”).

Классы

В реальном мире объекты обладают определенными свойствами и поведением. Автомобиль имеет цвет, вес, марку и бензобак определенного объема. Все это характеристики автомобиля. Автомобиль может набирать скорость, останавливаться, подавать сигналы поворота и звуковой сигнал. Эти действия — поведение автомобиля. Указанные характеристики и поведение являются общими для всех автомобилей. Несмотря на то, что разные автомобили могут быть окрашены в различные цвета, цвет имеют все автомобили. ОО-программирование, используя конструкцию, которая называется *классом*, позволяет выразить идею о том, что автомобиль — это объект, имеющий все вышеупомянутые характеристики. Класс представляет собой состоящий из переменных и функций блок кода, который описывает характеристики и поведение всех членов какого-либо множества. Например, класс с именем `Car` может описывать общие для всех автомобилей свойства и методы.

В ОО-терминологии характеристики класса называются *свойствами* (*properties*). Свойства имеют имена и значения. Значения некоторых свойств можно изменять, значения других нельзя. Например, у класса `Car`, вероятно, были бы такие свойства, как `color` (цвет) и `weight` (вес). Хотя цвет машины можно изменить, перекрасив ее, вес машины (без груза и пассажиров) является фиксированным значением.

Некоторые свойства представляют состояние объекта. Состояние отражает те характеристики, которые изменяются благодаря некоторым событиям, но не обязательно изменяемые непосредственно. В приложении, которое имитирует производительность транспортного средства, класс `Car` может иметь свойство с именем `velocity` (скорость). Скорость автомобиля не может измениться сама по себе, изменение скорости представляет собой следствие работы определенного количества топлива, переданного двигателю с учетом его производительности и характеристик дороги, по которой движется машина.

Поведение класса, т.е. действия, связанные с классом, называются *методами* (*methods*). Методы реализованы в виде функций. Как и функции, методы могут принимать параметры любого допустимого типа данных. Методы воздействуют на внешние данные, переданные им в виде параметров, либо на свойства своего объекта. Методы могут использовать свойства своего объекта, чтобы определить возможность выполнения того или иного действия (например, метод `accelerate` определяет количество оставшегося топлива, чтобы выяснить, может ли машина разогнаться). Кроме того, методы могут изменять состояние объекта путем изменения значений его свойств (например, метод `accelerate` изменяет значение свойства `скорость`).

Объекты

Класс можно представить как чертеж для создания объекта. Почти так же как по одному чертежу можно построить множество домов, из класса можно создать множество объектов. Однако на чертеже не указаны такие характеристики, как цвет стен или тип пола. На чертеже просто показано, что у дома будут стены и пол. Классы работают почти так же. Класс определяет поведение и характеристики объекта, но не обязательно значения этих характеристик. *Объект (object)* — это определенная сущность, созданная с помощью чертежа, представленного классом. Так идея дома аналогична классу, хотя сам дом (частный экземпляр идеи дома) аналогичен объекту.

При наличии чертежа и определенных строительных материалов можно построить дом. В ОО-программировании для создания объекта используется класс. Процесс создания объекта из класса называется *созданием экземпляра класса*. Для создания объекта требуется две вещи:

- ❑ область памяти, в которую помещается объект; она автоматически выделяется РНР;
- ❑ данные, которые станут значениями свойств; эти данные могут поступать из базы данных, неструктурированного текстового файла, другого объекта или какого-либо иного источника.

Класс в отличие от его объекта никогда не имеет значений свойств или состояния. Прежде чем клеить обои или делать наружную обшивку, необходимо с помощью чертежа построить дом. Аналогично, прежде чем взаимодействовать со свойствами объекта или вызывать его методы, необходимо создать объект класса. Новички в ОО-программировании часто вместо слова “объект” употребляют слово “класс” и наоборот. Запомните, что классы используются во время разработки, когда вносятся изменения в методы или свойства, а с объектами мы имеем дело во время выполнения программы, когда свойствам присваиваются значения и вызываются методы.

Как только объект создан, его можно использовать в работе, реализуя бизнес-требования приложения. Рассмотрим, как это делается в РНР.

Создание класса

Начнем с простого примера. Сохраните следующий код в файле с именем `class.Demo.php`:

```
<?php
class Demo {
}
?>
```

Этот код — определение класса `Demo`. Пока еще данный код не представляет собой ценности, однако он демонстрирует базовый синтаксис для объявления нового класса в РНР. Чтобы определить в РНР новый класс, используется ключевое слово `class`, за которым следует имя класса и фигурные скобки, указывающие на начало и конец кода этого класса.

Создать объект класса `Demo` можно с помощью следующего кода:

```
<?php
require_once('class.Demo.php');
```

```
$objDemo = new Demo();

?>
```

Чтобы создать объект, сначала необходимо убедиться, что PHP “знает”, где искать объявление класса. Это можно сделать, подключив файл, в котором содержится класс (в данном случае `class.Demo.php`). Затем следует вызвать оператор `new` и ввести имя класса, а также открывающую и закрывающую круглые скобки. Возвращаемое данным оператором значение присваивается новой переменной, в данном случае — `$objDemo`, после чего можно вызывать методы объекта `$objDemo`, а также считывать и устанавливать его свойства, если он таковые имеет.

Очень важно иметь ясное соглашение по организации файлов с исходным кодом. Существует хорошее правило: помещать каждый класс в отдельный файл и называть этот файл `class.[ИмяКласса].php`.

Несмотря на то, что только что созданный класс еще ничего не делает, его код все равно представляет собой правильное определение класса.

Добавление методов

Класс `Demo` останется бесполезным, если не сможет выполнять какие-либо действия, поэтому рассмотрим возможность создания методов класса. Необходимо помнить, что метод класса, по существу, представляет собой функцию. В главе 6, “Создание высококачественного кода”, уже было сказано, как создавать функции. Добавляя функцию внутри фигурных скобок определения класса, программист добавляет этому классу метод, например:

```
<?php
```

```
class Demo {
```

```
function sayHello($name) {
    print "Hello $name!";
}
```

```
}
```

```
?>
```

Теперь объект, происходящий из данного класса, способен распечатывать приветствие всем, кто вызовет его метод `sayHello`. Чтобы вызвать метод объекта `objDemo`, необходимо использовать оператор `->`, позволяющий получить доступ к только что созданной функции. Сохраните следующий код в файле `testdemo.php`:

```
<?php
```

```
require_once('class.Demo.php');
```

```
$objDemo = new Demo();
```

```
$objDemo->sayHello('Steve');
```

```
?>
```

Теперь объект способен выводить приветствие. Оператор `->` используется для доступа ко всем методам и функциям объектов.

Примечание для тех, кто познакомился с ОО-программированием в других языках: оператор `->` всегда используется для доступа к методам и свойствам объекта. В ОО-синтаксисе PHP оператор точка (`.`) вообще не применяется.

Добавление свойства

Добавить в класс свойство так же легко, как добавить метод — необходимо объявить внутри класса переменную для хранения значения данного свойства. В процедурном коде, чтобы сохранить какое-либо значение, требуется присвоить это значение переменной. В ОО-программировании для хранения значения какого-либо свойства также используется переменная. Эта переменная объявляется в начале объявления класса, внутри фигурных скобок, обрамляющих код класса. Имя данной переменной будет именем свойства, т.е. если переменная имеет имя `$color`, то свойство будет называться `color`.

Откройте файл `class.Demo.php` и добавьте в него выделенный код:

```
<?php
```

```
class Demo {
```

```
    public $name;
```

```
    function sayHello() {  
        print "Hello $this->name!";  
    }  
}
```

```
?>
```

Добавление новой переменной `$name` — вот все, что требуется сделать, чтобы создать свойство `name` класса `Demo`. Для использования этого свойства применяется тот же оператор `->` и имя свойства. Модифицированный метод `sayHello` показывает, как получить доступ к значению данного свойства.

Создайте новый файл `testdemo1.php` и введите в него следующий код:

```
<?php
```

```
require_once('class.Demo.php');
```

```
$objDemo = new Demo();  
$objDemo->name = 'Steve';
```

```
$objAnotherDemo = new Demo();  
$objAnotherDemo->name = 'Ed';
```

```
$objDemo->sayHello();  
$objAnotherDemo->sayHello();
```

```
?>
```

Сохраните файл и откройте его в Web-браузере. На экране будут отпечатаны строки “Hello Steve!” и “Hello Ed!”.

Ключевое слово `public` используется для того, чтобы впоследствии можно было вне класса получить доступ к данной переменной. Некоторые переменные класса существуют только для использования самим классом и не должны быть доступны внешнему коду. В данном примере можно устанавливать и считывать значение свойства `name`. Необходимо отметить, что работа метода `sayHello` изменилась. Теперь он получает значение из свойства, а не в виде своего параметра.

Чтобы объект мог получить информацию о самом себе, используется ключевое слово `$this`. В программе может использоваться несколько объектов одного класса, а поскольку имя объектной переменной заранее при создании класса неизвестно, то переменная `$this` позволяет обращаться к текущему экземпляру класса.

В данном примере первый вызов метода `sayHello` распечатывает строку `Steve`, а второй распечатывает `Ed`, потому что переменная `$this` позволяет каждому объекту обращаться к самому себе, не зная имени запрашиваемой в текущий момент объектной переменной. Необходимо помнить о том, что некоторые свойства влияют на работу определенных методов, например, метод `accelerate` класса `Car` должен определить количество оставшегося топлива. Код внутри метода `accelerate` в таком случае должен использовать конструкцию `$this->amountOfFuel`, чтобы получить доступ к указанному свойству.

Для получения доступа к свойствам необходим только один знак `$`. Синтаксис должен быть таким: `$obj->property`, а не `$obj->$property`. Часто это сбивает с толку новичков в PHP. Переменная свойства *объявляется* как `public $property`, а *доступ* к ней осуществляется как `$obj->property`.

В дополнение к переменным, в которых хранятся значения свойств класса, могут существовать другие переменные, объявленные для внутренних операций класса. Оба этих вида данных называются *внутренними переменными класса* (*internal member variables*). Некоторые из них доступны коду за пределами класса в виде свойств. Другие не доступны и предназначены строго для внутреннего пользования. Например, если класс `Car` по какой-либо причине должен получить информацию из базы данных, то во внутренней переменной этого класса может храниться дескриптор подключения к базе данных. Очевидно, что дескриптор подключения не является свойством класса `Car`, но он необходим классу для выполнения определенных операций.

Ограничение доступа к переменным экземпляра

Существует три уровня видимости методов или переменных экземпляра: открытый (`public`), частный (`private`) и защищенный (`protected`). Открытые члены (т.е. переменные) доступны для любого кода. Частные члены доступны только для самого класса. Обычно такие члены используются для внутренних целей, например, для хранения дескриптора соединения с базой данных. Защищенные члены доступны самому классу и всем его наследникам. (Определение и подробное рассмотрение наследования представлено в настоящей главе далее).

По умолчанию для любой переменной класса или функции, видимость которой не установлена явно, используется открытый доступ. Однако хорошая практика предполагает явное указание видимости всех членов класса.

Создание функций `get` и `set` для всех свойств класса в будущем значительно облегчает добавление проверки данных, новой бизнес-логики или внесение других изменений в работу объектов. Даже если текущие требования к приложению не включают в себя проверку данных для определенного свойства, все равно это свойство следует реализовывать с функциями `get` и `set`, так чтобы в будущем можно было добавить проверку данных или какую-либо функциональность.

Как показывает предыдущий пример, значение свойства `name` может быть любым, включая объект, массив целых чисел, дескриптор файла и др., однако во время установки свойства `name` невозможно выполнить какую-либо проверку данных или обновить другие значения.

Чтобы обойти эту проблему, рекомендуется реализовывать свойства в форме функций `get [имя_свойства]` и `set [имя_свойства]`, как показано ниже в разделе “Практика”.

Практика Доступ к свойствам посредством методов `get` и `set`

Внесите в файл `class.Demo.php` выделенные изменения:

```
<?php
```

```
class Demo {

    private $_name;

    public function sayHello() {
        print "Hello {$this->getName()}!";
    }

    public function getName() {
        return $this->_name;
    }

    public function setName($name) {
        if(!is_string($name) || strlen($name) == 0) {
            throw new Exception("Недопустимое имя");
        }

        $this->_name = $name;
    }

}
```

```
?>
```

Отредактируйте файл `testdemo.php` как показано ниже:

```
<?php
```

```
require_once('class.Demo.php');

$objDemo = new Demo();

$objDemo->setName('Steve');
$objDemo->sayHello();

$objDemo->setName(37); //генерирует ошибку

?>
```

Как это работает

Уровень доступа к свойству `name` изменился с `public` на `private` и в начале имени переменной появился символ подчеркивания. Рекомендуется использовать символ подчеркивания для обозначения частных переменных и функций экземпляра. Вместе с тем это лишь соглашение и PHP этого не требует. Ключевое слово `private` не по-

звolyет коду, находящемуся за пределами объекта, модифицировать данное значение. Частные внутренние переменные экземпляра недоступны снаружи класса. Так как получить непосредственный доступ к этим переменным невозможно, для работы с данной информацией приходится использовать методы `getName()` и `setName()`, которые гарантируют, что класс сможет проверить значение до того, как оно будет присвоено свойству. В данном примере генерируется исключительная ситуация, если для свойства `name` вводится недопустимое значение. Кроме того, в код добавлен спецификатор доступа `public` для функций.

Для работы со свойствами рекомендуется всегда использовать функции `get` и `set`. Это позволит в будущем значительно упростить реализацию требований бизнес-логики и проверки данных.

Использование функций `__get` и `__set`

Настоятельно рекомендуется придерживаться совета об использовании методов `get` и `set` для всех свойств. Вместе с тем для крупных объектов с десятками свойств это может привести к необходимости создавать большое количество кода. Это особенно нежелательно, если для большинства свойств нет непосредственных требований бизнес-логики. В таких ситуациях оказываются полезными методы `__get` и `__set`.

Если сценарий пытается получить доступ к свойству объекта, но открытой переменной с таким именем нет, то РНР проверяет, определена ли для данного объекта функция `__get`. Если такая функция определена, РНР автоматически вызывает ее с целью определить значение данного свойства. Аналогично, когда значение присваивается несуществующему свойству, РНР вызывает функцию `__set`, если она была определена.

Практика Использование методов `__get` и `__set`

Данный пример показывает, как использовать функции `__get` и `__set` для облегчения работы со свойствами. Создайте файл с именем `class.PropertyObject.php` и введите в него следующий код:

```
<?php

class PropertyObject {

    private $_properties = array(
        'name' => null,
        'dateofbirth' => null
    );

    function __get($propertyName) {
        if(!array_key_exists($propertyName, $this->_properties))
            throw new Exception('Недопустимое значение свойства!');

        if(method_exists($this, 'get' . $propertyName)) {
            return call_user_func(array($this, 'get' . $propertyName));
        } else {
            return $this->_properties[$propertyName];
        }
    }

    function __set($propertyName, $value) {
        if(!array_key_exists($propertyName, $this->_properties))
            throw new Exception('Недопустимое значение свойства!');
```

```

        if(method_exists($this, 'set' . $propertyName)) {

            return call_user_func(
                array($this, 'set' . $propertyName),
                $value
            );
        } else {
            $this->_properties[$propertyName] = $value;
        }
    }

    function setDateOfBirth($dob) {
        if(strtotime($dob) == -1) {
            throw new Exception("Недопустимое значение даты рождения!");
        }
        $this->_properties['dateofbirth'] = $dob;
    }

    function sayHello() {
        //$this->_properties['name'] и $this->_properties['dateofbirth']
        //доступны посредством метода __get
        print "Привет! Меня зовут $this->name. Я родился $this->dateofbirth";
    }
}
?>

```

Чтобы протестировать этот новый класс, создайте файл `testpropertyobject.php` со следующим кодом:

```

<?php
require_once('class.PropertyObject.php');

$obj = new PropertyObject();
$obj->name = 'Борис'; //значение "Борис" присваивается переменной
                  //$_properties['name'] с помощью метода __set
$obj->dateofbirth = '5 марта, 1977'; //Метод setDateOfBirth вызывается
                                функцией __set
$obj->sayHello();

$obj->dateofbirth = 'blue'; //Генерирует исключение

?>

```

Откройте страницу `testpropertyobject.php` в Web-браузере. На странице должно появиться сообщение: “Привет! Меня зовут Борис. Я родился 5 марта, 1977” и сообщение об ошибке, которое дает понять, что значение 'blue' не подходит для свойства `dateofbirth`.

Как это работает

Класс позволяет определить все допустимые свойства в массиве, элементы которого инициализируются с некоторым приемлемым значением по умолчанию (в данном случае Null). PHP автоматически направляет функции `__get` любые попытки обратиться к свойствам, для которых не существует открытых переменных.

В данном примере функция `__get` сначала проверяет, определено ли это свойство в частном массиве `$_properties`, и если нет, то генерируется исключительная ситуация. Если свойство определено, то проверяется наличие функции `get [имя_свойства]`. Если такая функция существует, то она вызывается посредством метода `call_user_func`. Он вызывает процедурную функцию, если первый из переданных ему параметров

является строкой (и эта строка содержит имя вызываемой функции). Если первым параметром является массив в форме `array($объект, $имяМетода)`, то функция `__get` вызывает метод `$имяМетода` объекта `$объект`. В таком случае вызывается функция `get[имяСвойства]` для объекта `$this`. Если функции `get[имяСвойства]` нет, то просто возвращается значение элемента массива `$_properties`, индекс которого соответствует имени свойства.

То же самое происходит и с методом `__set`. Сначала проверяется определение свойства, затем вызывается функция с именем `set[имяСвойства]` (если она существует), которой передается значение, присваиваемое свойству. Если такой функции нет, то значение присваивается элементу массива `$_properties`, ключ которого равен имени данного свойства.

В представленном примере метод `setDateOfBirth` проверяет дату, которая присваивается этому свойству. Попытка присвоить строку 'blue' свойству, в котором должна храниться дата рождения, приводит к генерированию исключительной ситуации.

Эффективность подобной методики заключается в том, что она позволяет использовать простые переменные для хранения значений свойств, а кроме того, создавая новую функцию, можно легко добавить бизнес-логику и проверку данных без необходимости переписывать код, использующий данный класс. Стоит также упомянуть, что синтаксис `$obj->myProperty='foo'` для присвоения значений свойств более понятен, чем вызов функции для присвоения простого свойства. Класс `PropertyObject` в этой и в следующей главе рассматривается еще несколько раз и постепенно улучшается. К концу следующей главы будет создан мощный вспомогательный класс, который можно будет использовать во многих проектах.

Инициализация объектов

При первом создании объектов отдельных классов требуется некоторая специальная подготовка. Например, иногда требуется выбрать информацию из базы данных или проинициализировать некоторые свойства. Создание специальной функции, которая называется `__construct()`, позволяет реализовать любые действия, необходимые при создании нового экземпляра класса. РНР автоматически вызывает эту функцию во время создания объекта.

Например, класс `PropertyObject` можно переписать следующим образом:

```
<?php
class PropertyObject {

    private $_properties;

    public function __construct() {
        $this->_properties = array();
        $this->_properties['name'] = null;
        $this->_properties['dateofbirth'] = null;
    }
}
```

. . . //остальной код, опущенный для краткости

Функция `__construct` вызывается автоматически, когда создается новый объект класса `PropertyObject`.

Для пользователей РНР4: в РНР4 конструктором объекта была функция, имя которой совпадало с именем класса. В РНР5 используется унифицированный конструктор.

В целях обеспечения обратной совместимости PHP сначала ищет функцию с именем `__construct`, а если не находит ее, то как и раньше ищет функцию, которая имеет то же имя, что и класс.

Кроме того, конструктору можно передавать параметры, которые можно использовать для инициализации значений свойств или для получения некоторой информации из базы данных. В следующем примере показано применение MySQL-функций для получения из базы данных информации о пользователях. В этом классе используются методы `__get` и `__set`, код которых взят из класса `PropertyObject`.

Практика Определение свойств объекта в конструкторе

Сохраните следующий код в файле `class.User.php`. Невыделенный код можно скопировать из файла `class.PropertyObject.php`. Для использования данного примера понадобится работающая MySQL-база данных. Чтобы использовать другую СУБД, можно просто заменить `mysql_`-функции функциями, соответствующими используемой платформе. Кроме того, следует удостовериться, что строка подключения соответствует требованиям используемой среды.

```
<?php

class User {

    private $_properties;
    private $_hDB;

    public function __construct($userID) {
        $this->_properties = array();
        $this->_changedProperties = array();
        $this->_properties['id'] = null;
        $this->_properties['username'] = null;
        $this->_properties['realname'] = null;

        $this->_hDB = mysql_connect('localhost', 'dbuser', 'mypassword');
        if(! is_resource($this->_hDB)) {
            throw new Exception("Невозможно подключиться к базе данных!");
        }

        $connected = mysql_select_db('mydatabase', $this->_hDB);

        if(! $connected) {
            throw new Exception("Невозможно использовать базу данных 'mydatabase!'");
        }

        $sql = "select * from users where id = $userID";
        $rs = mysql_query($sql, $this->_hDB);
        if(! mysql_num_rows($rs)) {

            throw new Exception("Пользователя с идентификатором $userID
                                не существует!");
        }
        $row = mysql_fetch_assoc($rs);

        $this->_properties['id'] = $row['id'];
        $this->_properties['username'] = $row['username'];
        $this->_properties['realname'] = $row['realname'];
    }
}
```

```
// Методы __get и __set для краткости опущены
```

```
//Запрещаем изменение идентификатора пользователя

function setID($value) {

    throw new Exception('Изменение идентификатора пользователя не допускается!');
}
function sayHello() {

    print "Привет! Меня зовут {$this->realname}. Мой идентификатор: {$this->id}";
}
}
?>
```

Создайте файл `testuser.php` и введите в него следующий код:

```
<?php
require_once('class.User.php');

$obj = new User(27); //Пользователь Боб Смит
$obj->sayHello();
//Выводит на экран строку "Привет! Меня зовут Боб Смит. Мой идентификатор: 27"

?>
```

Чтобы создать в MySQL-базе данных таблицу пользователей, можно ввести следующий SQL-оператор:

```
CREATE TABLE users (
    id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    username varchar(50),
    realname varchar(255)
);
```

Используя приведенный ниже оператор `INSERT`, можно заполнить только что созданную таблицу данными для тестирования класса:

```
mysql> INSERT INTO users (id, username, realname) VALUES(
    -> 27,
    -> 'bsmith',
    -> 'Боб Смит');
```

Если передаваемый конструктору идентификатор пользователя отсутствует в базе данных и код создания объекта не был заключен в блок `try... catch...`, то на экране появится примерно следующее сообщение:

```
Fatal error: Uncaught exception 'exception' with message 'No user exists
with id 12345!' in /path/to/class.User.php:28 Stack trace: # 0
/path/to/testuser.php(4): User->__construct() # 1 {main} thrown in
/path/to/class.User.php on line 28
(Неисправимая ошибка: не перехваченное исключение 'exception'
с сообщением 'Пользователя с идентификатором 12345 не существует!' файл
/path/to/class.User.php:28 Трассировка стека: # 0
/path/to/testuser.php(4): User->__construct() # 1 {main}
генерируется в файле /path/to/class.User.php строка 28)
```

Как это работает

Для хранения дескриптора ресурса подключения к базе данных создается частная переменная экземпляра `$_hDB`. Затем добавляется метод-конструктор `__construct`, которому передается параметр `$userID`. При создании новых объектов класса `User` конструктору передается идентификатор пользователя, данные которого необходимо выбрать из базы. Это значение представляет первичный ключ таблицы, в которой хранится информация о данном пользователе.

Конструктор открывает соединение с базой данных и сохраняет дескриптор ресурса в переменной `$this->_hDB`. Затем необходимо вызвать функцию `mysql_select_db` для указания имени базы данных, с которой требуется установить связь. После этого вводится оператор `SELECT` для получения из базы данных всей информации о пользователе. Полученная информация записывается в массив `$_properties`, где она будет использоваться функциями `__get`, `__set` либо функциями `get [имяСвойства]` или `set [имяСвойства]`.

В реальных приложениях информация, необходимая для подключения к базе данных, обычно хранится в глобально подключаемом файле (см. главы 9, 10 и 11). В случае надобности это позволяет легко изменять параметры подключения к базе данных или имя самой базы данных. Такой файл обычно является единственным местом хранения этой информации и избавляет от необходимости заново просматривать множество разных файлов для обновления настроек приложения.

Использование конструктора с параметром позволяет инициализировать все значения свойств, извлекая их из базы данных. В данном случае извлекается информация о пользователе. Конечно, в реальных приложениях необходимо убедиться, что строка 4 файла `testuser.php` заключена в блок `try... catch...`, и перехватываются все ошибки, которые могут возникать при подключении к базе данных или выборке информации о пользователе.

Кроме того, следует отметить, что функция `setID` генерирует исключение при попытке изменить свойство, в котором хранится идентификатор пользователя. Так как данное свойство является первичным идентификатором пользователя, не рекомендуется разрешать его изменение со стороны пользователей класса.

В случае создания класса, который не требует какого-либо особого кода инициализации, создавать конструктор нет необходимости. Как показано в предыдущем примере очень простого класса `Demo`, PHP автоматически выполняет все нужные для создания объекта операции. Функцию конструктора рекомендуется создавать, только когда это действительно необходимо.

Уничтожение объекта

Создаваемая в сценарии объектная переменная удаляется из системной памяти, когда запрашиваемая страница полностью завершила свою работу, когда переменная выходит за пределы области видимости или когда она явно устанавливается в `Null`. В PHP5 можно перехватывать уничтожение объекта и предпринимать в этот момент какие-либо действия. Чтобы это сделать, необходимо создать функцию с именем `__destruct`, не имеющую параметров. Функция-деструктор, если она существует, вызывается автоматически перед удалением объекта.

Это дает возможность в последний момент выполнять любые операции по очистке, например, закрытие дескрипторов файлов или соединений с базами данных, которые могли быть открыты данным классом, или любые другие внутренние операции, которые требуется выполнить перед уничтожением объекта.

Практика Создание деструктора

Эффективность данной методики иллюстрируется на следующем примере, в котором используется код из созданного ранее класса `User` с добавленным деструктором. Модернизация кода позволяет объектам класса `User` автоматически сохранять свои собственные изменения в базе данных. Изменения свойств фиксируются. Когда объект уничтожается, сценарий проверяет, какое свойство было изменено, а затем обновляет базу данных, используя новые значения свойств.

Внесите в файл `class.User.php` следующие изменения:

```
<?php

class User {

    private $_properties;

    private $_changedProperties; //содержит список
                               //измененных свойств

    private $_hDB;

    // методы __construct и __get для краткости опущены

    function __set($propertyName, $value) {
        if(!array_key_exists($propertyName, $this->_properties))
            throw new Exception('Недопустимое значение свойства!');

        if(method_exists($this, 'set' . $propertyName)) {
            return call_user_func(
                array($this, 'set' . $propertyName),
                $value
            );
        } else {

            //Если значение свойства действительно было изменено,
            //но еще не попало в массив changedProperties,
            //то добавить его.
            if($this->_properties[$propertyName] != $value &&
                !in_array($propertyName, $this->_changedProperties)) {
                $this->_changedProperties[] = $propertyName;
            }

            //Устанавливаем значение свойства
            $this->_properties[$propertyName] = $value;
        }
    }

    //запрещаем изменение значения пользовательского идентификатора
    function setID($value) {
        throw new Exception('Изменение пользовательского идентификатора не допускается!');
    }

    function sayHello() {
        print "Привет! Меня зовут {$this->realname}. Мой идентификатор: {$this->id}";
    }
}
```

```

function __destruct() {

    //Проверяем наличие изменений. Если они есть,
    //то сохраняем их в базе данных.
    if(sizeof($this->_changedProperties)) {

        $sql = "UPDATE users SET ";

        //Формируем SQL-оператор путем создания
        //массива операторов и их последующего объединения.
        //В конце каждого оператора добавляется запятая.
        $setStatements = array();
        foreach($this->_changedProperties as $prop) {
            $setStatements[] = "$prop = '{$this->_properties[$prop]}'";
        }

        //создаем строку

        $sql .= join(', ', $setStatements);

        //присоединяем предложение WHERE

        $sql .= " WHERE userid = $this->id";
        $hRes = mysql_query($sql);

    }
    //Закрываем подключение к базе данных - все закончено.
    mysql_close($this->_hDB);
}
}

```

Измените файл `testuser.php`, как показано ниже:

```

<?php
require_once('class.User.php');

$obj = new User(27); //Пользователь Роберт Смит

$obj->realname = 'Боб Смит';

$obj->sayHello();

$obj2 = new User(34); //Джейн Доу
$obj2->sayHello();

?>

```

Как это работает

Сначала создается новая частная переменная экземпляра, которая называется `$_changedProperties`. В метод `__set` вносятся изменения, позволяющие сохранять в этой переменной изменения любых свойств. Когда эта функция вызывается, проверяется как имя свойства, так и его и его значение. Если новое значение отличается от старого, то изменение регистрируется и свойству присваивается новое значение. Нет необходимости регистрировать изменение более одного раза, поэтому также проверяется, не было ли данное свойство уже добавлено в массив `$_changedProperties`.

Затем добавляется метод-деструктор `__destruct`. В данном примере он выполняет две функции:

1. Деструктор проверяет наличие изменений в свойствах объекта. Если какие-либо изменения есть, то формируется UPDATE-оператор для ввода в базу данных. UPDATE-оператор обновляет только те значения, которые действительно были изменены.
2. Деструктор закрывает соединение с сервером баз данных. После записи изменений в базу данных остается открытым ресурс соединения с ней, который необходимо закрыть. Соединение первоначально было открыто в конструкторе. Сохраняя данное соединение открытым в течение времени существования объекта, можно использовать это соединение для выполнения других запросов, избегая при этом издержек открытия новых соединений. Закрытие соединения по окончании работы объекта позволяет сэкономить системные ресурсы.

После того как объект `$obj2` вывел приветственное сообщение, сценарий завершает работу и РНР-интерпретатор начинает процесс уничтожения всех неиспользуемых переменных, включая `$obj` и `$obj2`. В ходе этого процесса проверяется наличие определения метода `__destruct`. Если данный метод определен, то он вызывается для объекта перед удалением этого объекта из системной памяти. В данном примере это происходит в конце сценария `testuser.php`.

Насколько действенной может быть эта методика? Сценарий `testuser.php` позволяет извлечь из базы данных информацию о пользователе, изменить какое-либо свойство данного пользователя и автоматически записать изменения обратно в базу данных, используя всего две строки кода. Если никаких изменений не было, как в случае со вторым объектом (`$obj2`), то базу данных можно не использовать, сокращая, таким образом, нагрузку на сервер баз данных и повышая производительность приложения.

Пользователи объекта не обязательно должны понимать его внутреннее устройство. Если ведущий разработчик написал класс `User`, он может передать объект новичку, который, возможно, не разбирается в SQL, а новичок в свою очередь сможет применить объект, даже не зная о том, откуда поступают данные или как они должны сохраняться. Фактически можно заменить источник данных с MySQL-базы на PostgreSQL-базу или даже на XML-файл, не сообщая об этом новичкам и даже не изменяя ни одной строки кода, использующего данный класс.

Наследование

Предположим, что требуется создать приложение для контроля складских запасов торгового представительства автомобильного завода. Для реализации подобного приложения, вероятно, понадобится создать такие классы, как `Sedan`, `PickupTruck` и `MiniVan`, которые будут соответствовать одноименным типам автомобилей в реестре торгового представительства. Приложение должно отображать не только количество имеющихся автомобилей, но и их характеристики, чтобы продавцы могли предоставлять соответствующую информацию клиентам.

Седан — четырехдверный автомобиль, и желательно было бы записывать в базу количество задних мест, а также объем багажника для каждого седана. Пикап не имеет багажника, но у него есть грузовой отсек определенной емкости и грузоподъемности. Фургон (`MiniVan`) имеет одну или две сдвижные двери и несколько посадочных мест.

Однако каждое из этих транспортных средств на самом деле представляет собой просто разные типы автомобилей, и поэтому в разрабатываемом приложении они будут

иметь множество общих характеристик, например, цвет, марку, модель, год выпуска, номер и т.д. Чтобы гарантировать, что каждый из классов имеет такие свойства, можно было бы скопировать код, создающий данные свойства, во все файлы, содержащие определения классов. Однако, как уже отмечалось, одно из преимуществ ОО-программирования заключается в возможности повторного использования кода. Следовательно, можно не копировать код, создающий свойства, а вместо этого повторно использовать свойства и методы данных классов. Такую возможность предоставляет процесс, который называется *наследованием (inheritance)*. Наследование — возможность класса получать методы и свойства родительского по отношению к нему класса.

Наследование позволяет определить базовый класс — в данном случае класс `Automobile` — и указать, что другие классы представляют собой типы автомобилей, и поэтому имеют все те же свойства и методы, которые реализованы в классе `Automobile`. Таким образом, седан — это автомобиль, и поэтому он автоматически наследует все свойства и методы, определенные в классе `Automobile`, и копировать код для этого не требуется. Придется только написать дополнительные свойства и методы класса `Sedan`, которые нехарактерны для всех остальных автомобилей. Все что требуется сделать, это определить различия; сходные черты всех классов будут унаследованы от базового класса.

Возможность повторного использования кода — не единственное достоинство применения наследования; есть еще одно значительное преимущество. Предположим, что существует класс `Customer`, имеющий метод `buyAutomobile`. Данный метод принимает один параметр — объект класса `Automobile` и распечатывает все необходимые документы для оформления продажи автомобиля, а затем уменьшает количество машин, оставшихся на складе. Так как все седаны, пикапы и фургоны являются автомобилями, можно передавать объекты соответствующих классов в функцию, которая ожидает объект класса `Automobile`. Так как объекты трех указанных типов наследуют характеристики более общего родительского класса, известно, что все они будут иметь одинаковый базовый набор свойств и методов. До тех пор, пока будут нужны только методы и свойства, общие для всех автомобилей, можно будет принимать объекты любого класса-наследника `Automobile`.

Рассмотрим другой пример: кошки. Все кошки имеют ряд общих свойств — вес, окрас, длину усов и скорость бега. Вместе с тем, львы имеют гриву определенной длины (по крайней мере, самцы) и они рычат. Гепарды имеют пятнистый окрас. Обычные домашние кошки не имеют этих свойств, но все указанные животные принадлежат семейству кошачьих.

В РНР можно указать, что класс представляет собой подмножество другого класса, с помощью ключевого слова `extends` (расширяет), которое сообщает РНР-машине, что объявляемый класс должен наследовать все свойства и методы родительского по отношению к нему класса, и что кроме этого в объявляемый класс будут добавлены дополнительные свойства или методы.

Если бы пришлось создавать приложение, “имитирующее” животных в зоопарке, то, вероятно, понадобилось бы использовать классы `Cat` (кошка), `Lion` (лев) и `Cheetah` (гепард). Прежде чем писать какой-либо код, следует спланировать иерархию классов, используя UML-диаграммы. Эти диаграммы впоследствии послужат начальной точкой для написания кода и документации разрабатываемых классов. (Язык UML подробнее рассматривается в главе 13.) Диаграмма классов должна отражать родительский класс `Cat` и его подклассы `Lion` и `Cheetah`, наследующие его свойства и методы. Пример UML-диаграммы показан на рис. 12.1.

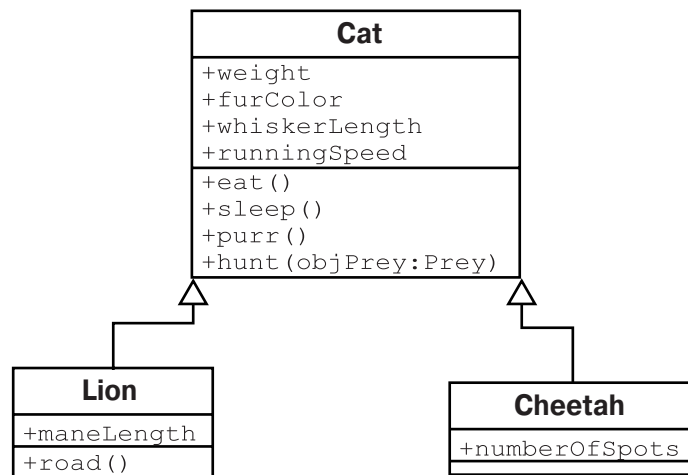


Рис. 12.1.

Оба класса, `Lion` и `Cheetah`, — наследники класса `Cat`, но в классе `Lion`, кроме того, реализовано свойство `maneLength` (длина гривы) и метод `roar()` (рычать), тогда как в класс `Cheetah` добавлено свойство `numberOfSpots` (количество пятен).

Класс `Cat` может быть реализован следующим образом:

```

<?php

class Cat {
    public $weight;          //вес в кг
    public $furColor;        //масть
    public $whiskerLength;   //длина усов
    public $maxSpeed;        //максимальная скорость, км/час

    public function eat() {
        //код метода "есть"...
    }

    public function sleep() {
        //код метода "спать"...
    }

    public function hunt(Prey $objPrey) {
        //код для метода "охотиться" за объектами
        //типа Prey (добыча), который не определен...
    }

    public function purr() {
        //функция мурлыкать
        print "муррррррр...";
    }
}

?>
  
```

В данном простом классе устанавливаются общие для всех кошек свойства и методы. Чтобы создать классы `Lion` и `Cheetah`, можно было бы скопировать в них весь код класса `Cat`, но такой подход привел бы к двум проблемам. Во-первых, если в классе `Cat`

обнаружится дефект, то этот дефект также придется исправлять в классах `Lion` и `Cheetah`. Это увеличивает объем необходимой работы, а не уменьшает его (уменьшение объема работы — одно из основных преимуществ объектно-ориентированного подхода).

Во-вторых, предположим, что есть метод какого-либо другого объекта, который выглядит следующим образом:

```
// . . .дополнительный блок кода
// функция "погладить кошечку"
public function petTheKitty(Cat $objCat) {
    $objCat->purrr();
}
```

Хотя гладить льва или гепарда небезопасно, они, вероятно, промурлыкали бы, если бы позволили себя погладить. Должна быть возможность передавать функции `petTheKitty()` объект класса `Lion` или `Cheetah`.

Поэтому необходимо создавать классы `Lion` и `Cheetah` другим путем, а, следовательно, использовать наследование. Используя ключевое слово `extends` и указав имя расширяемого класса, можно легко создать два новых класса, которые будут иметь все те же свойства, что и обычные кошки, а также некоторые дополнительные возможности. Например:

```
<?php
require_once('class.Cat.php');

class Lion extends Cat {
    public $maneLength; //длина гривы, см

    public function roar() {
        print "Pppppppp!";
    }
}
?>
```

Это все. Используя класс `Lion`, расширяющий класс `Cat`, можно создать следующий сценарий, `testlion.php`:

```
<?php
include('class.Lion.php');

$objLion = new Lion();
$objLion->weight = 200; //кг или ~450 фунтов.
$objLion->furColor = 'коричневый';
$objLion->maneLength = 36; //см или ~14 дюймов
$objLion->eat();

$objLion->roar();
$objLion->sleep();
?>
```

Теперь можно использовать свойства и методы дочернего класса `Lion`, не переписывая весь код. Обратите внимание, что ключевое слово `extends` указывает РНР-интерпретатору автоматически включить всю функциональность класса `Cat` наряду со всеми свойствами и методами, характерными для класса `Lion`. Кроме того, использование данного ключевого слова означает, что объект класса `Lion` также является `Cat`-объектом и поэтому теперь можно вызывать функцию `petTheKitty()`, передавая ей `Lion`-объект, несмотря на то, что в определении данной функции в качестве параметра фигурирует объект класса `Cat`:

```
<?php
include('class.Lion.php');
$objLion = new Lion();
```

```
$objLion->petTheKitty($objLion);
```

```
?>
```

Таким образом, любые изменения в классе `Cat` автоматически наследуются классом `Lion`. Исправление ошибок, изменения внутренней реализации функций или новые методы и свойства передаются подклассам родительского класса. В крупной хорошо организованной объектной иерархии такие изменения могут значительно упростить исправление ошибок и усовершенствование кода. Небольшое изменение родительского класса может сильно повлиять на все приложение.

Практика Создание класса `Cheetah`

В данном примере показано, как можно использовать метод-конструктор для расширения и специализации класса. Создайте новый файл с именем `class.Cheetah.php` и введите в него следующий код:

```
<?php
require_once('class.Cat.php');

class Cheetah extends Cat {
    public $numberOfSpots;

    public function __construct() {
        $this->maxSpeed = 100;
    }
}
?>
```

Введите в файл `testcheetah.php` следующий код:

```
<?php
require_once('class.Cheetah.php');

function petTheKitty(Cat $objCat) {
    if($objCat->maxSpeed < 5) {
        $objCat->purr();
    } else {
        print "Невозможно погладить эту кошечку - она двигается со скоростью " .
            $objCat->maxSpeed . " километров в час!";
    }
}

$objCheetah = new Cheetah();
petTheKitty($objCheetah);

$objCat = new Cat();
petTheKitty($objCat);
?>
```

Как это работает

В класс `Cheetah` добавляется новая общедоступная переменная `$numberOfSpots` и конструктор, которого не было в родительском классе `Cat`. Теперь, когда создается новый объект класса `Cheetah`, свойству `maxSpeed` (унаследованному от `Cat`) присваивается значение 100 км/час, которое примерно соответствует максимальной скорости гепарда на короткой дистанции. Так как стандартное значение этого свойства

в классе `Cat` не указано, в функции `petTheKitty()` свойство `maxSpeed` вычисляется равным 0 (фактически `Null`). Учитывая то, как долго спят домашние кошки, их максимальная скорость, вероятно, стремится к нулю.

Путем добавления новых функций, свойств или конструкторов и деструкторов можно легко расширять функциональность подклассов родительского класса. Иначе говоря, с минимальным количеством кода в приложение можно добавлять новые возможности. Более развитый пример рассматривается в следующей главе при расширении класса `PropertyObject` до классов приложения для работы с контактной информацией.

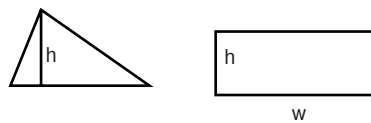
Если один класс является особым типом другого класса, то использование наследования максимально увеличит потенциальную возможность повторного применения кода и повысит гибкость приложения.

Переопределение методов

Только лишь тот факт, что дочерний класс наследует от родительского свойства и методы, не означает, что дочерний класс должен обязательно использовать ту же реализацию функций, что и у родительского класса. Например, для разработки приложения, вычисляющего площади различных геометрических фигур, могут понадобиться классы `Rectangle` (прямоугольник) и `Triangle` (треугольник). Обе эти фигуры являются многоугольниками, и поэтому данные классы будут наследовать свойства и методы от родительского класса, который называется `Polygon` (многоугольник).

В классе `Polygon` должно быть свойство `numberOfSides` (количество сторон) и метод `getArea` (вычислить площадь). Можно вычислить площадь любого многоугольника, но методы вычисления площади для типов многоугольников могут отличаться друг от друга. Например, для вычисления площади прямоугольника используется формула $w \cdot h$, где w — ширина прямоугольника, а h — его высота. Площадь треугольника вычисляется по формуле $(h/2) \cdot b$, где h — высота треугольника с основанием b . На рис. 12.2 иллюстрируются оба примера.

Площадь треугольника = $(1/2h) \times b$



Площадь прямоугольника = $w \times h$

Рис. 12.2.

Для каждого создаваемого подкласса класса `Polygon`, вероятно, потребуется метод вычисления площади, который будет отличаться от реализации, используемой по умолчанию в родительском классе. Создать собственную реализацию можно путем переопределения этого метода для определенного класса.

Например, в случае класса `Rectangle` создается два новых свойства, `height` и `width` (высота и ширина соответственно), и переопределяется родительская реализация метода `getArea` (общая формула для вычисления площади сложного многоугольника, которую реализовывал бы метод `getArea` класса `Polygon`, довольно сложна, и поскольку данный метод все равно будет переопределен для всех подклассов, здесь

углубляться в тонкости геометрических формул нет необходимости). Для класса `Triangle`, вероятно, добавились бы свойства для хранения информации о трех углах, высоте треугольника и длине его основания, а для переопределения метода `getArea` использовалась бы высота и длина основания. Используя наследование и подменяя методы родительского класса, можно создать в подклассах их собственные реализации этих методов.

Функция, которая принимает в качестве параметра многоугольник и должна рассчитывать его площадь, автоматически вызывает метод `getArea` того подкласса класса `Polygon`, который был ей передан (например, `Rectangle` или `Triangle`). Способность объектно-ориентированного языка программирования автоматически во время выполнения программы определять, какой метод `getArea` необходимо вызвать, называется *полиморфизмом* (*polymorphism*). Полиморфизм — способность приложения выполнять различные функции в зависимости от конкретного объекта, на который воздействует это приложение. В данном случае это означает вызов различных методов `getArea`.

Подменять метод в подклассе следует тогда, когда реализация этого метода в родительском классе не удовлетворяет требованиям подкласса. Это позволяет специализировать действия подкласса.

Иногда необходимо оставить реализацию какого-либо метода, предоставленную родительским классом, а в подклассе добавить в этот метод некоторые дополнительные действия. Предположим, например, что существует приложение для контроля членов какой-либо некоммерческой организации. Приложение содержит класс `Volunteer`, в котором имеется метод `signup`. Он позволяет добровольцам вступать в состав общественной миссии, а затем добавляет пользователей в список добровольцев этой миссии.

Однако добровольцами могут оказаться пользователи с некоторыми ограничениями, например, люди с уголовным прошлым, которых не следует допускать к определенным проектам. В таком случае полиморфизм позволяет создать класс `RestrictedUser` с переопределенным методом `signup`, который сначала сверяет ограничения учетной записи пользователя со свойствами проекта и не допускает пользователей к регистрации, если их ограничения запрещают им заниматься определенными видами деятельности. Если пользовательские ограничения не запрещают добровольцам участвовать в данном проекте, то для завершения регистрации вызывается метод родительского класса.

При подмене метода родительского класса не обязательно полностью переписывать этот метод. Можно продолжать использовать родительскую реализацию, добавив дополнительную специализацию для метода подкласса. Это позволяет повторно использовать код, а также обеспечить соответствие требованиям бизнес-логики.

Способность одного класса наследовать методы и свойства другого класса является одной из самых выдающихся особенностей объектно-ориентированной системы и позволяет достигать невероятных уровней эффективности и гибкости приложений.

Практика Подмена наследуемых методов

В данном примере создается два класса: `Rectangle` (прямоугольник) и `Square` (квадрат). Квадрат является особым видом прямоугольника. Все допустимые для прямоугольника операции допустимы и для квадрата, но прямоугольник имеет две различные длины сторон, а квадрат имеет только одну длину стороны, поэтому некоторые операции отличаются.

Создайте файл `class.Rectangle.php` и введите в него следующий код:

```
<?php

class Rectangle {
    public $height;
    public $width;

    public function __construct($width, $height) {
        $this->width = $width;
        $this->height = $height;
    }

    public function getArea() {
        return $this->height * $this->width;
    }
}

?>
```

Данный код представляет собой довольно простую реализацию класса, моделирующего прямоугольник. Конструктор принимает параметры для ширины и высоты, а функция вычисления площади умножает эти значения друг на друга.

Ниже приведен код файла `class.Square.php`:

```
<?php
require_once('class.Rectangle.php');

class Square extends Rectangle {
    public function __construct($size) {
        $this->height = $size;
        $this->width = $size;
    }

    public function getArea() {
        return pow($this->height, 2);
    }
}

?>
```

В данном классе подменяется конструктор и метод `getArea()`. У квадрата все стороны должны иметь одинаковую длину. Поэтому для конструктора нужен только один параметр. Если конструктору передается несколько параметров, то все значения после первого будут проигнорированы.

Проверим работу класса с помощью следующего сценария (`testsquare.php`):

```
<?php
require_once('class.Square.php'); $obj = new Square(7); $a = $obj->getArea();
echo "$a";
?>
```

Данный код распечатывает ожидаемое значение площади квадрата с длиной стороны 7.

PHP не генерирует ошибку, если количество параметров, переданное пользовательской функции, превышает количество параметров в объявлении данной функции (в некоторых случаях такое поведение желательно). Подробнее этот момент рассматривается в документации по встроенной функции `func_get_args()`.

В классе `Square` также переопределяется функция `getArea()`. Реализация класса `Rectangle` сама по себе возвращает идеально верный результат для вычисления площади квадратов — метод `getArea()` был подменен для повышения производительности приложения (хотя в рассматриваемом случае выигрыш производительности мизерный). РНР-интерпретатор быстрее определяет значение одного свойства и вычисляет квадрат числа, чем два свойства и их произведение.

Переопределяя конструкторы, деструкторы и другие методы, можно изменять разные показатели работы подклассов.

Сохранение функциональности родительского класса

Иногда необходимо сохранить функциональность, предоставленную родительским классом, т.е. не подменять его функции полностью, а лишь добавить в них некоторые операции. Можно было бы скопировать весь код из родительского метода в метод подкласса, но, как уже было показано выше, ОО-свойства РНР предлагают более разумные способы решения данной задачи, чем просто копирование кода.

Для того чтобы вызвать функцию, предоставленную родительским классом, необходимо использовать следующий синтаксис: `parent::[имя_функции]`. Чтобы просто добавить дополнительные действия в метод, сначала вызывается `parent::[имя_функции]`, а затем добавляется дополнительный код. При расширении функции таким способом рекомендуется всегда сначала вызывать родительский метод, а затем выполнять остальные действия. Это гарантирует, что любые изменения работы родительского метода не нарушат код подкласса.

Поскольку родительский класс может ожидать передачи ему объекта в определенном состоянии или изменять состояние объекта, переписывать значения свойств или манипулировать внутренними данными объекта, при расширении наследуемых методов необходимо всегда вызывать родительский метод перед добавлением расширяющего кода.

Рассмотрим пример использования данной методики.

Практика Сохранение функциональности родительского метода

В данном примере задействовано два класса: `Customer` и `SweepstakesCustomer`. В супермаркете используется приложение, которое время от времени в зависимости от проводимых акций переключает кассовые аппараты на использование одного или другого класса. Каждый новый покупатель получает идентификатор (значение из базы данных), а также номер, который показывает, сколько покупателей пришло в супермаркет до него. Миллионный покупатель выигрывает приз.

Создайте файл `class.Customer.php` и введите в него следующий код:

```
<?php

class Customer {
    public $id;
    public $customerNumber;
    public $name;

    public function __construct($customerID) {
        //получение из базы данных информации о покупателе
        //
        //Представленные ниже значения, очевидно, взяты не из
        //базы данных, но в реальном приложении их
```

```

        //следовало бы извлекать из базы данных.
        $data = array();
        $data['customerNumber'] = 1000000;
        $data['name'] = 'Jane Johnson';

        //Присваиваем объекту значения из базы данных
        $this->id = $customerID;
        $this->name = $data['name'];
        $this->customerNumber = $data['customerNumber'];
    }
}
?>

```

Создайте файл `class.SweepstakesCustomer.php` и введите в него следующий код:

```

<?php
require_once('class.Customer.php');

class SweepstakesCustomer extends Customer {
    public function __construct($customerID) {
        parent::__construct($customerID);

        if($this->customerNumber == 1000000) {
            print "Поздравляем, $this->name! Вы - наш миллионный покупатель! " .
                "Вы выигрываете приз - годовой запас рыбных палочек! ";
        }
    }
}
?>

```

Чтобы посмотреть данный класс в работе, создайте файл `testCustomer.php` и введите в него следующий код:

```

<?php

require_once('class.SweepstakesCustomer.php');
//поскольку файл class.SweepstakesCustomer.php
//уже включает файл class.Customer.php,
//нет необходимости еще раз подключать здесь последний.

function greetCustomer(Customer $objCust) {
    print "$objCust->name, добро пожаловать в наш магазин снова!";
}

//Измените данное значение, чтобы изменить класс,
//используемый для создания объекта Customer
$promotionCurrentlyRunning = true;

if($promotionCurrentlyRunning) {
    $objCust = new SweepstakesCustomer(12345);
} else {
    $objCust = new Customer(12345);
}
greetCustomer($objCust);

?>

```

Запустите сценарий `testCustomer.php` в браузере, присвоив сначала переменной `$promotionCurrentlyRunning` значение `false`, а затем `true`. В последнем случае должно появиться сообщение о выигрыше приза.

Как это работает

Класс `Customer` инициализирует значения из базы данных на основе идентификатора покупателя. Идентификатор покупателя можно получить из дисконтной карточки, которые используются в большинстве крупных сетей супермаркетов. Имея идентификатор покупателя, можно получить его персональную информацию из базы данных (в этом примере номер карточки жестко запрограммирован), а также целое число, представляющее количество покупателей, которые совершили покупки до этого покупателя. Вся эта информация сохраняется в общедоступных переменных экземпляра.

В классе `SweepstakesCustomer` функциональность конструктора расширяется. Сначала вызывается конструктор родительского класса (`parent::__construct()`), которому передаются необходимые параметры. Затем проверяется значение свойства `customerNumber`. Если оно равно 1000 000, то приложение сообщает покупателю о выигрыше приза.

Интерфейсы

Нередко используются группы классов, не связанных отношениями наследования. Возможны ситуации, когда используются полностью отличающиеся классы, которые случайно имеют некоторые общие функции. Например, и банка и дверь может быть закрытой или открытой — и это единственное, чем они похожи. Независимо от типа банки и двери, оба эти предмета могут выполнять упомянутые действия, но между ними нет никаких других общих черт.

В объектно-ориентированном программировании *интерфейс* (*interface*) позволяет указать, что объект способен выполнять определенную функцию, но не обязательно сообщать, как он должен это делать. Интерфейс представляет собой соглашение между несвязанными объектами о выполнении общей функции. Объект, реализующий интерфейс, гарантирует своим пользователям, что он может выполнять все функции, определенные спецификацией интерфейса. Велосипеды и футбольные мячи — абсолютно разные вещи, тем не менее, объекты, представляющие их в складской программе спортивного магазина, должны быть способны взаимодействовать с данной программой.

Объявление интерфейса и его последующая реализация в объектах позволяет передавать полностью различные классы одним и тем же функциям. В следующем примере иллюстрируется аналогия банки и двери.

Практика Использование интерфейсов

Создайте файл `interface.Openable.php`:

```
<?php
interface Openable {
    abstract function open();
    abstract function close();
}

?>
```

Класс называется `class.[имяКласса].php`. То же самое соглашение следует использовать для именования файлов с интерфейсами, присваивая им имена типа `interface.[имяИнтерфейса].php`.

Интерфейс объявляется с помощью синтаксиса, аналогичного объявлению класса, за исключением того, что вместо ключевого слова `class` используется слово `interface`. Как правило, интерфейс не имеет переменных экземпляра и в нем не описана реализация функций экземпляра.

Поскольку реализация не определена, такие функции объявляются как абстрактные (`abstract`). Ключевое слово `abstract` сообщает РНР, что любой класс, реализующий данный интерфейс, должен обеспечить реализацию соответствующих функций. Если в классе, реализующем интерфейс, не реализованы *все* абстрактные функции, то РНР-интерпретатор генерирует ошибку времени выполнения. Выборочная реализация некоторых абстрактных методов не допускается — необходимо реализовать все абстрактные методы.

Абстрактный метод — метод, для которого в интерфейсе не предусмотрена реализация. Если какой-либо метод класса объявлен как абстрактный, то сам класс также должен быть объявлен как абстрактный (в объявлении класса перед словом `class` должно присутствовать ключевое слово `abstract`). Невозможно создать объект самого абстрактного класса — абстрактный класс должен иметь подклассы, которые обеспечивают конкретную реализацию абстрактных методов. Обратите внимание, что объявления абстрактных методов не содержат фигурных скобок и заканчиваются точкой с запятой.

Как это работает

Интерфейс `Openable` представляет собой соглашение с остальными частями приложения о том, что класс, реализующий данный интерфейс, обеспечит два метода — `open()` и `close()`, — которые не принимают параметров. Данное согласованное множество методов позволяет передавать различные объекты в одну и ту же функцию. При этом отношение наследования между ними не требуется.

Создайте файл `class.Door.php`:

```
<?php
require_once('interface.Openable.php');

class Door implements Openable {
    private $_locked = false;

    public function open() {
        if($this->_locked) {
            print "Дверь не открывается. Она заперта.";
        } else {
            print "скрип...<br>";
        }
    }

    public function close() {
        print "Хлоп!!<br>";
    }

    public function lockDoor() {
        $this->_locked = true;
    }

    public function unlockDoor() {
        $this->_locked = false;
    }
}

?>
```

и файл `class.Jar.php`:

```
<?
require_once('interface.Openable.php');

class Jar implements Openable {
    private $contents;

    public function __construct($contents) {
        $this->contents = $contents;
    }

    public function open() {
        print "Банка открыта<br>";
    }

    public function close() {
        print "Банка закрыта<br>";
    }
}
?>
```

Для того чтобы можно было использовать эти классы, создайте новый файл `testOpenable.php` в том же каталоге:

```
<?php
require_once('class.Door.php');
require_once('class.Jar.php');

function openSomething(Openable $obj) {
    $obj->open();
}

$objDoor = new Door();
$objJar = new Jar("желе");

openSomething($objDoor);
openSomething($objJar);

?>
```

Так как оба класса `Door` и `Jar` реализуют интерфейс `Openable`, объекты обоих этих классов можно передавать функции `openSomething()`. Поскольку она принимает только объекты, реализующие интерфейс `Openable`, можно утверждать, что внутри этой функции можно вызывать методы `open()` и `close()`. Однако пытаться использовать свойство `contents` класса `Jar` или методы `lock()` либо `unlock()` класса `Door` внутри функции `openSomething()` не следует, потому что это свойство и методы не являются частью интерфейса. Соглашение интерфейса гарантирует только то, что в реализующем данный интерфейс классе будут присутствовать методы `open()` и `close()`.

Используя в приложении интерфейсы, можно заставить взаимодействовать абсолютно разные и несвязанные друг с другом объекты, гарантируя при этом, что их взаимодействие будет регламентировано спецификацией интерфейса.

Инкапсуляция

Как уже отмечалось в этой главе ранее, объекты позволяют скрывать подробности их реализации от своих пользователей. Например, пользователь не должен знать, где класс `Volunteer` сохраняет информацию, необходимую для вызова метода `signpur` (в базе данных, в неструктурированном текстовом файле, в XML-файле), или использует

иной механизм хранения данных. Аналогично, ему не обязательно знать, где внутри объекта хранится информация о добровольцах — в скалярных переменных, в массиве или даже в другом объекте. Возможность скрывать подробности реализации называется *инкапсуляцией* (*encapsulation*). Вообще инкапсуляция подразумевает две идеи: защита внутренних данных класса от внешнего по отношению к данному классу кода и сокрытие деталей реализации.

Слово “инкапсулировать” буквально означает “помещать что-либо в капсулу или внешний контейнер”. Хорошо организованный класс обеспечивает полноценную внешнюю оболочку вокруг своих внутренних данных и предоставляет внешнему коду интерфейс, который полностью отделен от деталей внутренней реализации класса. Это дает два преимущества: во-первых, можно в любое время изменять детали реализации, не влияя при этом на код, использующий данный класс. Во-вторых, поскольку внешний по отношению к данному классу код не может случайно незаметно изменить состояние или свойства объекта данного класса, можно быть уверенным, что состояние объекта и значения его свойств являются достоверными и имеют смысл.

Выше уже говорилось о том, что переменные экземпляра класса и его функции характеризуются *видимостью* (*visibility*) (извне класса). Частные переменные и функции экземпляра класса не доступны коду вне этого класса и используются для внутренней реализации. Защищенные переменные и функции экземпляра класса видимы только для самого класса и его подклассов. Открытые (или общедоступные) переменные и функции экземпляра класса могут использоваться как внутренним кодом класса, так и кодом за его пределами.

Вообще говоря, все внутренние переменные класса должны быть объявлены как частные (с ключевым словом `private`). Любой доступ к этим переменным извне класса должен осуществляться посредством общедоступных методов. Вы ведь не позволяете всем, кто предлагает вам попробовать новое блюдо, заталкивать его вам прямо в желудок. Для этого есть весомая причина — хочется самому посмотреть на это блюдо и решить, стоит ли его есть. Аналогично, если объект позволяет внешнему коду изменять его свойства или иначе влиять на его внутренние данные, то инкапсуляция доступа к этим данным в общедоступной функции (и сохранение внутренних данных частными) дает возможность оценить изменения, а затем принять или отвергнуть их.

Например, при создании банковского приложения, которое обрабатывает информацию о счетах клиентов, можно использовать объект класса `Account`, имеющий свойство `totalBalance` (итоговый баланс), а также методы `makeDeposit` (сделать вклад) и `makeWithdrawal` (снять деньги). Свойство `totalBalance` должно быть доступным только для чтения. Единственный способ повлиять на баланс — снять деньги или сделать вклад. Если бы переменная `totalBalance` была реализована как общедоступная переменная экземпляра, то можно было бы написать код, который изменял бы значение данной переменной без фактического внесения или снятия денег со счета. Очевидно, что такая ситуация крайне нежелательна для банка. Поэтому данное свойство реализовано в виде частной переменной экземпляра, а метод `getTotalBalance` возвращает значение этой переменной. То, что переменная, в которой хранится итоговый баланс, объявлена как частная, не позволяет манипулировать ею непосредственно. Поскольку повлиять на баланс могут только общедоступные методы `makeWithdrawal` и `makeDeposit`, пользователь вынужден внести вклад, чтобы увеличить сумму на своем счете.

Инкапсуляция внутренних данных и реализации методов позволяет объектно-ориентированным программным системам защищать свои данные и управлять доступом к ним, а также скрывать детали реализации, создавая, таким образом, гибкие и стабильные приложения.

Изменения объектно-ориентированных возможностей RНР5

Поддержка объектов была реализована в РНР, начиная с третьей версии языка. Разработчики не имели намерений поддерживать идею классов или объектов, но некоторая ограниченная поддержка позднее все-таки была добавлена с целью создания (как сказал Зив Сураски (Zeev Suraski)) “синтаксического сахара” для ассоциативных массивов. Поддержка объектов в РНР первоначально задумывалась как удобный способ группировки данных и функций, однако в РНР было включено только небольшое подмножество функций, которые обычно связывают с полнофункциональным объектно-ориентированным языком программирования.

По мере того как РНР становился все более популярным, объектно-ориентированный подход стал все чаще использоваться в крупных приложениях. Вместе с тем, слаборазвитая внутренняя реализация оказалась серьезным ограничивающим фактором. В частности, не было поддержки реальной инкапсуляции. Переменные или методы экземпляра невозможно было объявить как частные или защищенные. Все свойства и методы были общедоступными, а это, как уже было сказано, может создавать определенные проблемы.

Кроме того, не было поддержки абстрактных интерфейсов или методов. Методы и переменные экземпляра невозможно было объявить как статические. Не было деструкторов. Все эти понятия хорошо знакомы программистам, имеющим опыт работы с объектно-ориентированными языками, а недостаток этих возможностей в объектной модели РНР мог усложнять преобразование кода из таких языков, как Java (которые действительно поддерживают все эти понятия), в РНР.

В РНР5 вводится множество значительных изменений в объектно-ориентированные возможности языка. Эти изменения решают перечисленные выше и многие другие проблемы, обеспечивая РНР реальными ОО-возможностями и повышенной производительностью. Ниже представлен перечень нововведений.

- ❑ Ключевые слова для управления видимостью переменных экземпляра и методов, позволяющие объявлять свойства и методы как частные, защищенные и общедоступные.
- ❑ РНР5 обеспечивает поддержку разыменования, которая позволяет писать код наподобие `$obj->getObj()->doSomething()`. В предыдущих версиях РНР поддержка разыменования была ограниченной.
- ❑ Теперь в РНР поддерживаются статические методы и переменные классов, что улучшает проверку во время трансляции и выполнения сценариев. Статические методы вызываются с помощью оператора `::`.
- ❑ Унифицированные конструкторы — использование метода `__construct()` вместо метода с тем же именем, что и имя класса, — значительно упрощает изменение наследования, когда в дерево наследования вовлечено несколько классов.
- ❑ Классы в РНР теперь имеют деструкторы — методы `__destruct()`, которые позволяют выполнять определенные действия во время уничтожения объекта.
- ❑ В РНР5 была добавлена поддержка абстрактных классов и интерфейсов. Это позволяет определять необходимые методы в родительском классе, а реализовывать их позднее — в производных классах. Невозможно создать объект

абстрактного класса, только неабстрактный подкласс данного класса может порождать объекты.

- В функциях и методах можно использовать контроль типов для принимаемых параметров. Теперь можно указать класс для параметров функции, которая ожидает передачи ей объекта. Например, функция `function foo(Bar $objBar) {...` гарантирует, что типом данных параметра `$objBar` будет объект класса `Bar`.

Резюме

В данной главе рассматривались понятия объектно-ориентированного программирования. Класс был представлен как чертеж для создания объектов. Объекты представляют собой существующие во время выполнения программы данные и функции, созданные из определения класса. Объекты имеют характеристики, которые называются свойствами, и поведение — методы. Свойства, по сути, являются переменными, а методы функциями.

Некоторые классы имеют общего родителя. Когда класс объявляется как подтип родительского класса, он наследует методы и свойства последнего. Существует возможность переопределять унаследованные методы. Эти методы в случае необходимости можно реализовать заново, а можно продолжать использовать их родительскую реализацию, но добавить в подкласс некоторые особенности (или вообще не подменять методы).

Инкапсуляция — важное понятие объектно-ориентированного программирования. Она означает способность класса ограничивать доступ к своим внутренним переменным и ограждать детали своей внутренней реализации от пользователей. Существует три уровня видимости данных и функций класса: частные члены, которые могут использоваться только во внутренних операциях класса; защищенные члены, которые видимы подклассам; общедоступные члены, которые могут использоваться кодом за пределами самого класса.

С введением пятой версии PHP и Zend Engine 2 поддержка объектно-ориентированных возможностей была радикально пересмотрена. Новые функции и значительное повышение производительности делают PHP настоящим объектно-ориентированным языком программирования.

Упражнения

1. В чем разница между классом и объектом?
2. Объясните идею наследования и дайте пример того, когда его следует использовать, не повторяя при этом примеров, рассмотренных в данной главе.
3. Опишите полезные особенности интерфейса и его практическое применение в программной архитектуре. Чем интерфейс отличается от наследуемого класса? Приведите дополнительные примеры возможного применения интерфейсов.

13

Работа с UML и классами

Одним из самых полезных инструментов для моделирования объектно-ориентированных программ является UML (Unified Modeling Language — унифицированный язык моделирования). Данная глава представляет собой введение в UML; в ней объясняется, почему этот язык стоит использовать, как различные UML-диаграммы способствуют разработке РНР-приложений, а также описывается доступное программное обеспечение, позволяющее РНР-программистам создавать UML-диаграммы. Создаваемые в качестве учебных примеров UML-диаграммы в конце главы составят полномасштабный пример приложения: диспетчера контактов.

Диспетчер контактов предназначен для управления данными об организациях и частных лицах, а также позволяет пользователям просматривать и редактировать контактную информацию этих объектов. На примере этой программы обсуждаются различные проблемы, связанные с созданием работоспособных объектно-ориентированных приложений. Попутно в ходе этой дискуссии иллюстрируются главные принципы, лежащие в основе идеи объектного ориентирования, такие как повторное использование кода, инкапсуляция и абстракция.

Унифицированный язык моделирования

Громоздкая терминология ОО-программирования может сильно затруднять письменное описание множества объектов. Вместе с тем тщательное планирование и четкое документирование объектов является ключевым фактором для успешного создания приложения. Это особенно справедливо в случае крупных приложений, в которых используются десятки объектов. Однако как сгенерировать такую документацию, которая была бы доступной для тех, кто должен ее читать? Как говорится, одна картина стоит тысячи слов. UML (унифицированный язык моделирования) представляет собой спецификацию, описывающую стандартный процесс создания и визуализации объектно-ориентированных программных систем.

UML — язык, но не в обычном смысле этого слова. Это система для представления в визуальной форме классов и взаимодействия между ними. В данной системе используются стандартизированные обозначения для построения полной графической модели приложения. В UML используются изображения для очевидного обозначения комплексных взаимосвязей классов, а также создаются документы, которые впоследствии помогают понять это приложение и возможности его использования и модификации (это полезно как другим людям, так и самому разработчику по прошествии некоторого времени с момента первоначального выпуска приложения). До опубликования UML-спецификации существовало несколько систем, представляющих аналогичные идеи. В UML-спецификацию вошли лучшие черты каждой из этих систем. В результате был сформирован единый стандарт, который отверг предыдущие системы и установил обобщенный визуальный язык.

В текущей версии UML-спецификации определено 12 различных типов диаграмм, которые описывают структуру, поведение или организацию приложения. В этой главе подробно рассматривается только одна из этих диаграмм, диаграмма классов. Кратко представлены несколько других видов диаграмм. Полное описание UML выходит за рамки данной книги; доступно множество отличных ресурсов, позволяющих подробнее изучить UML, включая книгу *Освой самостоятельно UML 2 за 24 часа* (ИД “Вильямс”, 2005 г.)

Зачем использовать UML?

Многие разработчики программного обеспечения озабочены необходимостью написания документации. Часто документация предназначена для клиентов, специалистов по развитию бизнеса и остального нетехнического персонала, и создание такой документации отвлекает разработчиков от прямых обязанностей — написания кода. Более того, документы, ориентированные главным образом на разработчиков, такие как технические спецификации, часто остаются нетронутыми и неп прочитанными после создания. UML-диаграммы задуманы как работоспособное средство для описания проекта, и их создание на самом деле приносит разработчику практическую выгоду во время *всего процесса* разработки, а не только на стадии планирования.

UML-диаграммы позволяют разработчику перенести на бумагу идеи, касающиеся архитектуры приложения, при этом не требуется тратить время на написание пояснительных текстов. Во время планирования проекта диаграммы упрощают визуальное воспроизведение крупномасштабных изменений программной архитектуры, позволяя разработчику не заботиться в это время о коде. Диаграммы предоставляют “носитель”, посредством которого архитекторы программного обеспечения могут передавать свои идеи тем, кто будет их реализовывать. Но более всего впечатляет то, что в зависимости от инструмента, который используется для создания диаграмм, на базе этих диаграмм фактически можно генерировать большой объем кода для классов; иными словами, разработчик может буквально “нарисовать” код.

Инструменты для создания UML-диаграмм

Существуют десятки приложений, как с открытым исходным кодом, так и коммерческих, которые способны генерировать UML-диаграммы. К сожалению, поддержка автоматического создания РНР-кода в настоящий момент крайне ограничена. Ниже кратко описывается несколько стоящих внимания инструментов.

- ❑ Dia (www.lysator.liu.se/~alla/dia/) — основанное на библиотеке GTK средство, распространяемое под лицензией GPL, которое поддерживает UML-диаграммы. Существует скомпилированная версия для Windows. Приложение не поддерживает автоматическую генерацию кода. Все UML-диаграммы в книге созданы с помощью Dia.
- ❑ Visio (www.microsoft.com/office/visio) — часть офисного пакета Microsoft, которая находит широкое применение от UML-диаграмм и сетевых диаграмм до компоновочных планов и организационных схем. Продукт Dia задумывался как открытая альтернатива Visio. Visio также не имеет средств для автоматической генерации кода.
- ❑ IBM Rational Rose (www.rational.com) — давний золотой стандарт UML-разработки, компания Rational Rose предоставляет не только программные инструменты, но и полную методологию разработки программного обеспечения. Имеются генераторы кода для Java и .NET, но поддержка PHP отсутствует.
- ❑ ArgoUML (<http://argouml.tigris.org>) — этот продукт написан на Java, но в настоящее время в нем имеется экспериментальная поддержка генерации PHP-кода. Этот продукт распространяется бесплатно под лицензией BSD-стиля (более подробная информация представлена на Web-сайте ArgoUML).
- ❑ Umbrello (<http://uml.sourceforge.net>) — исключительно Unix-инструмент (нет версии для Windows), имеющий полную поддержку генерации PHP-кода. Продукт распространяется в виде открытого исходного кода.
- ❑ Poseidon For UML, Professional Edition (www.gentleware.com/products/) — коммерческий UML-инструмент с полной поддержкой генерации PHP-кода.

Хотя для изучения оставшегося материала книги не требуется ни один из этих инструментов, рекомендуется загрузить один из них, чтобы вы смогли самостоятельно создавать UML-диаграммы, подобные примерам в этой главе. Для создания примеров диаграмм использовалось приложение Dia, поэтому проще всего использовать именно этот продукт. Для этого достаточно посетить соответствующий Web-сайт и следовать инструкциям, в процессе инсталляции и инициализации программы проблем возникнуть не должно.

Существует две причины, по которым генерация PHP-кода ограничена или вообще отсутствует в большинстве упомянутых инструментов. Обычно разработчики программ на языках сценариев, таких как PHP, редко придерживаются строгих процессов разработки программного обеспечения, которые характерны для Java- и C++-разработчиков. UML — технологическое средство и поэтому разработчики UML-инструментов не обращали внимания на PHP. Другая причина заключается в том, что до выхода PHP5 в PHP была лишь зачаточная поддержка объектно-ориентированного программирования. Теперь, когда PHP имеет настоящую ОО-поддержку, очевидно, следует ожидать того, что большинство из UML-инструментов будет поддерживать генерацию PHP-кода.

Диаграммы классов

Как уже отмечалось, в этой книге описывается только одна из 12 типов диаграмм, определенных в спецификации UML, — диаграмма классов. Диаграмма классов показывает классы в приложении и их взаимосвязь.

Базовым обозначением классов (рис. 13.1) является прямоугольник, разделенный на три части. В первой части указывается имя класса, во второй — его свойства, а в третьей — методы. Части отделяются друг от друга горизонтальными линиями.

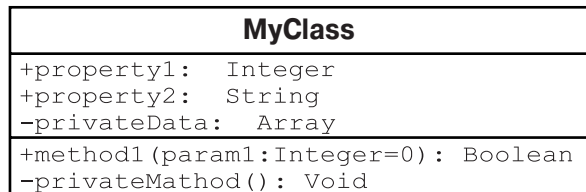


Рис. 13.1.

В примере представлен класс с именем MyClass, имеющий два общедоступных свойства (property1 — целое число и property2 — строка) и одну частную переменную экземпляра (массив privateData). Также в классе присутствуют два метода: method1() — общедоступный метод, принимающий один целочисленный параметр param1 (по умолчанию принимается значение 0) и возвращающий булево значение; частный метод privateMethod() не принимает никаких параметров и не возвращает значений (Void-метод). Знак “плюс” перед свойством или методом указывает на то, что данное свойство или метод является общедоступным, тогда как знак “минус” обозначает частный элемент. Защищенные методы обозначаются символом фунта (#).

Очевидно, что понять все это намного проще, глядя на диаграмму, чем читая текстовое описание. Именно поэтому UML используется для описания объектов. Текстовое описание и его форматирование занимает приблизительно в три раза больше времени, чем создание диаграммы в Dia.

Конечно, если используется только один класс, то диаграмма не особенно полезна. Польза диаграмм проявляется только при описании иерархии объектов. В следующем разделе рассматривается создание UML-диаграммы для реального создаваемого в этой книге приложения.

Создание диспетчера контактов

В остальной части этой главы описывается создание учебного приложения — диспетчера контактов. Приложение позволяет пользователям хранить информацию об организациях, частных лицах (такую как почтовый и e-mail-адреса, номера телефонов и т.д.), а также предоставляет возможность отслеживать взаимосвязь частных лиц и организаций. Фактически те же задачи решает адресная книга Microsoft Outlook.

Приложение должно хранить информацию об адресатах в базе данных и отображать ее на Web-странице. В составе контактной информации адресата могут отсутствовать или быть в наличии почтовые и e-mail-адреса, а также номера телефонов. В контактной информации частного лица имеются сведения о работодателе, т.е. организации, у которой в свою очередь есть сведения о наемных работниках (частных лицах).

UML-диаграммы диспетчера контактов

Запустите приложение для создания UML-диаграмм и создайте новый файл ContactManager.[extension], где [extension] — стандартное расширение файлов в данном приложении (например, .dia для диаграмм, созданных с помощью Dia).

Сначала следует создать классы, представляющие три различных вида контактной информации — почтовых адресов, e-mail-адресов и номеров телефонов. Ниже описаны свойства этих классов.

<i>Класс</i>	<i>Свойства</i>
Address	street1 street2 city state zipcode type (например, домашний, рабочий и т.д.)
EmailAddress	email type
PhoneNumber	number extension type

Эти классы только хранят и отображают данные, поэтому нет необходимости определять в настоящий момент какие-либо методы, следовательно, третья часть в обозначении каждого класса пуста. Все свойства общедоступны, поэтому в качестве префикса используется знак “плюс”. На рис. 13.2 показано UML-представление описываемых классов.

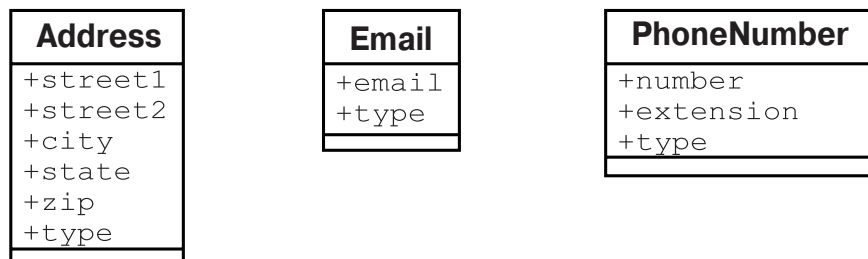


Рис. 13.2.

Теперь следует спланировать классы Individual и Organization. Частное лицо имеет имя, фамилию, уникальный идентификатор (ID-поле из базы данных), набор e-mail-адресов, почтовые адреса, телефонные номера, название предприятия-работодателя и название должности. Также должна присутствовать возможность добавлять типы контактов. На рис. 13.3 показана UML-диаграмма класса Individual.

Организация имеет название, идентификатор, такие же наборы типов контактов, методы для добавления контактов и множество наемных работников. На рис. 13.4 показана UML-диаграмма класса Organization.

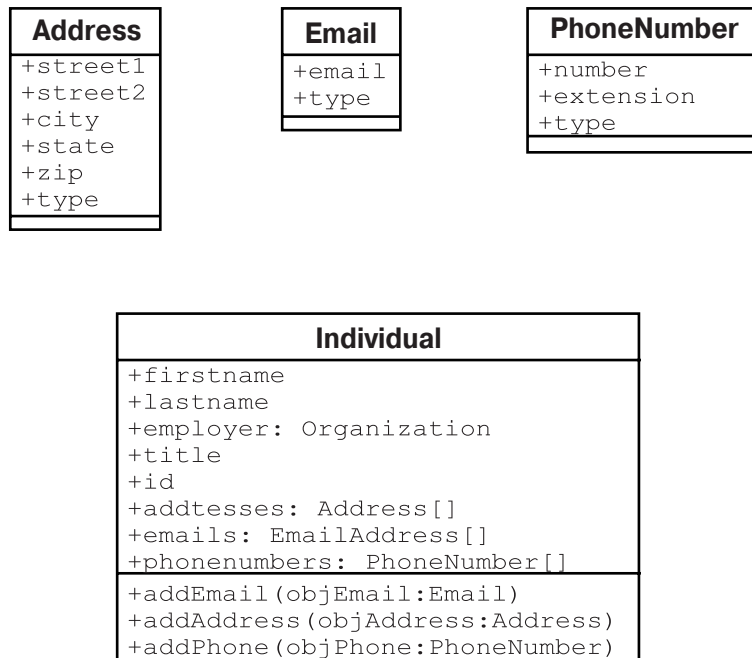


Рис. 13.3.

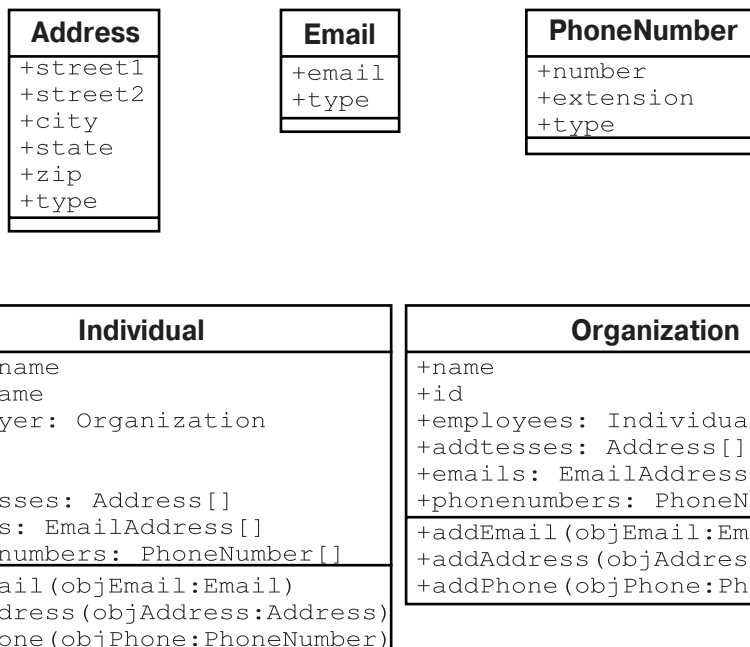


Рис. 13.4.

Диаграмма показывает классы `Individual` и `Organization`, в которых имеются одинаковые свойства и методы. Обычно это свидетельствует о том, что можно сэкономить значительный объем работы и улучшить гибкость приложения, используя наследование. Можно создать еще один класс (в данном примере `Entity`), в котором будут объединены характеристики, общие для классов `Individual` и `Organization`, и позволить этим классам совместно использовать один и тот же код. В UML-диаграмме свойства и методы указываются только для класса, который фактически их реализует. В данном случае необходимо переместить все общие свойства и методы классов `Individual` и `Organization` в прямоугольник, представляющий класс `Entity` (рис. 13.5). Свойства и методы повторяются в прямоугольнике дочернего класса, только если дочерний класс переопределяет их реализацию.

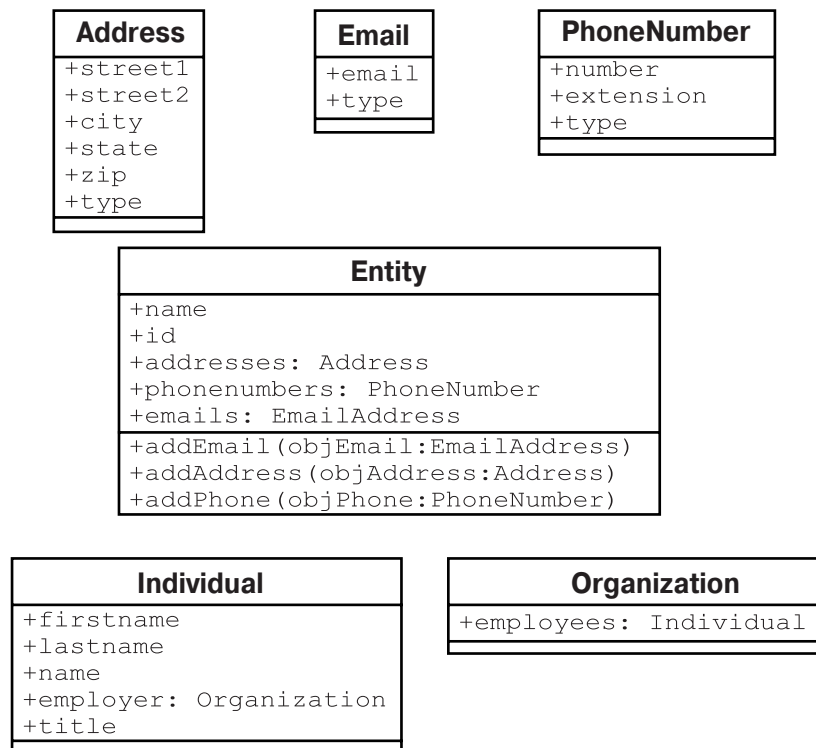


Рис. 13.5.

Здесь свойство `name` класса `Entity` переопределяется в классе `Individual`. Если в дальнейшем запросить значение свойства `name` для частного лица, то в ответ будет возвращено `"lastname, firstname"`. Это позволяет передавать объекты классов `Organization` и `Individual` в одну функцию, которая лишь распечатывает имя, т.е. вместо двух отдельных функций используется одна.

В UML также определены символы для обозначения взаимосвязей. В данном примере необходимо показать, что классы `Individual` и `Organization` наследуют свойства и методы класса `Entity`. В UML-спецификации такая взаимосвязь называется *обобщением* (*generalization*) и обозначается с помощью стрелки с не закрашенным наконечником, которая из дочернего класса (или классов) указывает на родительский класс (рис. 13.6).

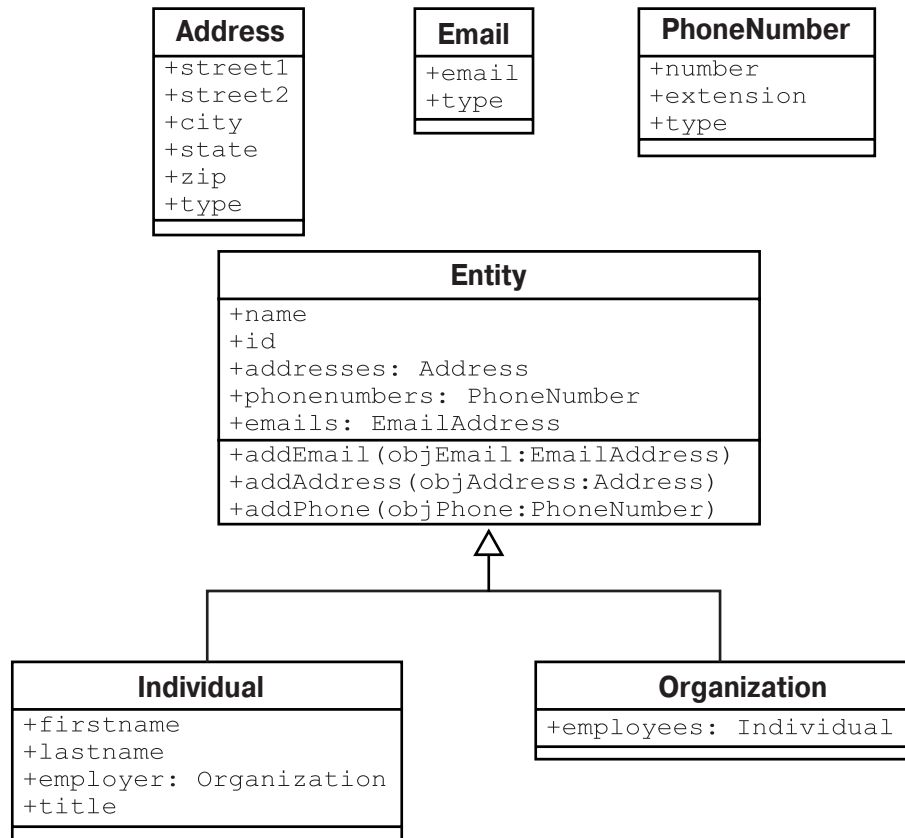


Рис. 13.6.

Теперь ясно, что классы `Individual` и `Organization` наследуют класс `Entity`. Использование соединительных линий для обозначения наследования позволяет легко при беглом просмотре диаграммы понять, какие классы связаны друг с другом.

Часто приходится показывать другой тип соединения: класс `Entity` использует классы `Address`, `Email` и `PhoneNumber`. В UML-спецификации такая взаимосвязь называется *агрегированием* (*aggregation*) и обозначается с помощью черного ромбика на том конце линии, который присоединен к классу-пользователю (рис. 13.7). Используемые классы имеют свойство множественность (*multiplicity*), которое показывает, сколько раз они используются. В этом примере в классе `Entity` может присутствовать один или несколько типов контактов, либо не быть ни одного. Для того чтобы обозначить это, над соединительной линией ближе к используемому классу необходимо написать `0..*`, чтобы показать, что класс может иметь 0 или больше указывающих на него классов. Это обозначение показано на рис. 13.7. Увидев такую связь, разработчик может ясно представить, на какие части приложения повлияет изменение того или иного класса. В данном случае изменение в классе `Email`, `Address` или `PhoneNumber` повлияет на классы `Entity`, `Individual` и `Organization`.

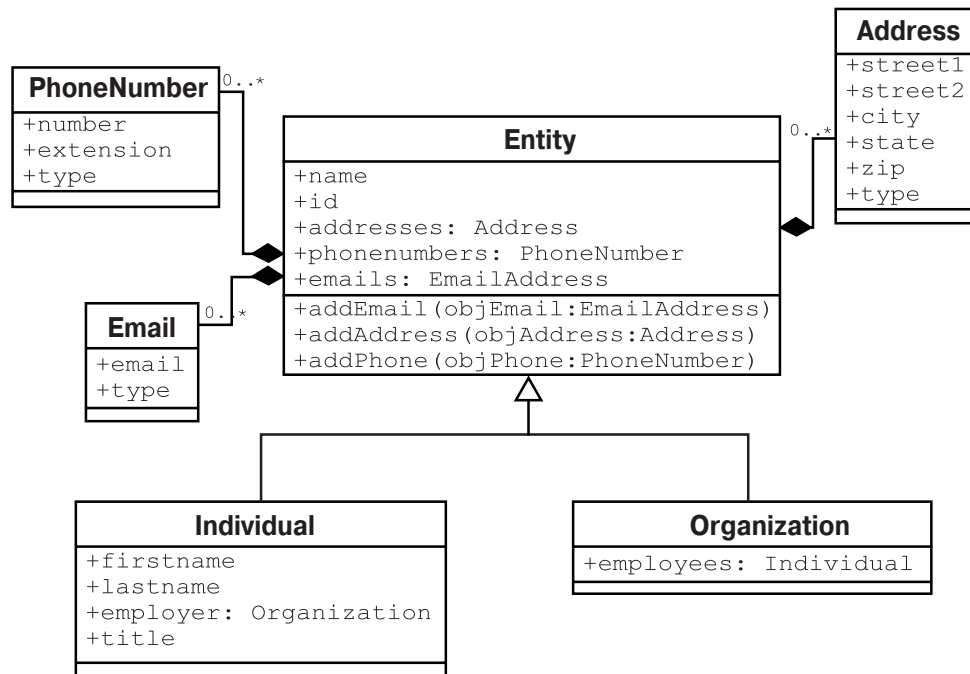


Рис. 13.7.

Объект класса **Entity** имеет телефонный номер (**PhoneNumber**), почтовый и e-mail-адреса (**Address** и **Email**) и во всех трех случаях он может иметь ноль или более типов контактов. Поскольку классы **Individual** и **Organization** наследуют класс **Entity**, они оба также имеют ноль или более типов контактной информации.

Имея полную UML-диаграмму, можно начинать писать код, который будет управлять разрабатываемым приложением. Если используемый UML-инструмент способен генерировать PHP-код, то можно автоматически получить первоначальные скелеты спроектированных классов. После этого пришлось бы только добавить бизнес-логику и обеспечить доступ к данным.

Другие полезные UML-диаграммы

Диаграммы классов не единственные UML-диаграммы, которые можно использовать при разработке или описании приложения. На самом деле существует 11 других типов диаграмм, позволяющих добиться определенного выигрыша. Рассмотрим кратко несколько наиболее широко используемых диаграмм. В Internet-ресурсах и печатных источниках можно найти более подробные сведения об этих типах диаграмм и используемых в них символах, а также о других диаграммах, определенных в спецификации UML.

Диаграммы активности

Диаграммы активности (activity diagrams) используются для планирования бизнес-логики и решений, которые должно принимать приложение. Они особенно полезны при документировании сложных процессов принятия решений. На рис. 13.8 показана диаграмма действий довольно простого процесса принятия решения, который уже рассматривался в главе 12 при разборе классов **Volunteer** и **RestrictedVolunteer**.

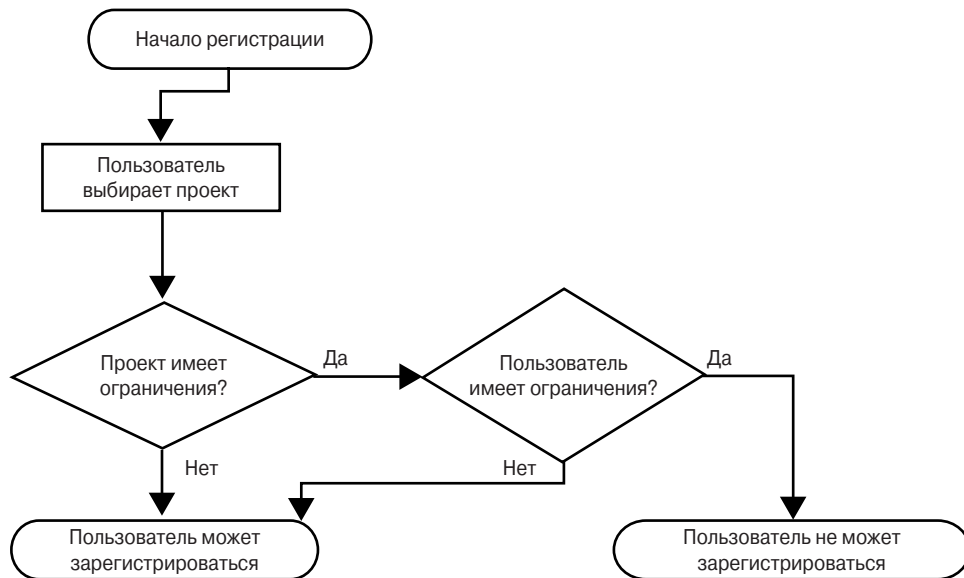


Рис. 13.8.

Эта диаграмма показывает, где принимаются решения (ромбики), необходимые действия (прямоугольники), а также начальное и конечное состояния (эллипсы) процесса. При написании комплексного кода с условной логикой рекомендуется создать диаграмму действий для планирования и документирования кода.

Диаграммы ситуаций

Диаграмма ситуаций (use case diagram) иллюстрирует бизнес-процессы и их пользователей. Это нетехническая диаграмма, она способствует пониманию задач при переговорах с клиентами.

В диаграммах ситуаций используется два базовых обозначения: актор (пользователь процесса) и ситуация (процесс). Актором обычно является класс пользователей приложения. Например, крупный Web-сайт, как правило, содержит общедоступную область и систему администрирования, которая позволяет определенным пользователям манипулировать содержимым сайта. В таком случае имеется два актора: администратор и пользователь. Администраторы изменяют и просматривают содержимое сайта и создают различные отчеты. Пользователи только просматривают содержимое. На рис. 13.9 показана соответствующая диаграмма ситуаций.

На диаграмме представлены два актора, три ситуации и указатели на то, какие акторы участвуют в той или иной ситуации. Такие диаграммы помогают понять роли и ответственность в приложении и перечислить бизнес-процессы, которые будет поддерживать приложение.

Диаграмма последовательностей

Диаграмма последовательностей (sequence diagram) показывает порядок операций для определенного процесса в приложении. Горизонтальная ось представляет жизненный цикл задействованных в процессе объектов, а вертикальная ось — порядок операций (первая операция находится сверху). Диаграмма такого типа показывает

от начала и до конца процесса объекты, сообщаящиеся друг с другом, и передаваемые ими сообщения. На рис. 13.10 показан пример диаграммы последовательности для процесса изменения содержимого Web-сайта.

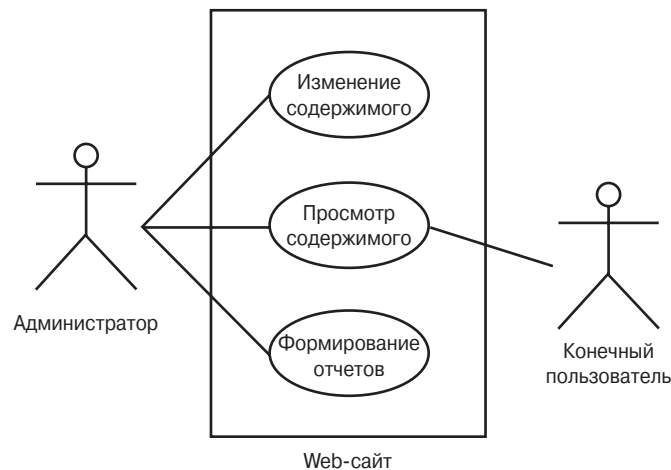


Рис. 13.9.

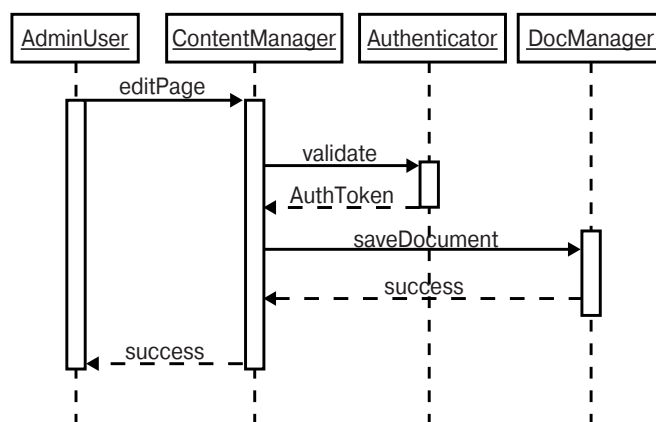


Рис. 13.10.

Сплошные линии показывают сообщения, которые отсылаются объектам, перечисленным сверху. Пунктирные линии показывают ответы этих объектов. Прямоугольники показывают начало и конец участия объекта в той или иной части последовательности. Диаграмма показывает, что когда администратор редактирует страницу, класс ContentManager проверяет параметры доступа пользователя, создавая экземпляр класса Authenticator и вызывая метод `validate()`. Этот метод возвращает метку аутентификации в форме объекта класса `AuthToken`. Затем диспетчер содержимого просит класс DocManager сохранить документ. Метод `saveDocument()` возвращает результат — `success` (успех) или `failure` (неудача) (в этом примере сохранение выполнено успешно) и этот результат передается администратору.

Диаграмма последовательностей упрощает планирование информационного обмена между объектами и документирует этапы выполнения заданного действия.

Приведенные выше примеры показывают, какими полезными могут быть UML-диаграммы при проектировании приложения. Используя общий визуальный язык для представления различных понятий в программной системе, можно в доступной для понимания форме передавать сложные идеи и документировать замысловатые процессы. Стандартизация используемых обозначений гарантирует, что другие разработчики программного обеспечения смогут понять диаграммы и использовать их для создания описанного ими приложения.

Создание класса Entity

Теперь можно применить все полученные ранее знания для создания спроектированного выше приложения — диспетчера контактов. Приложение предназначено для управления информацией о частных лицах и организациях и предоставляет пользователям возможность просматривать и редактировать контактную информацию. На рис. 13.11 показана UML-диаграмма диспетчера контактов.

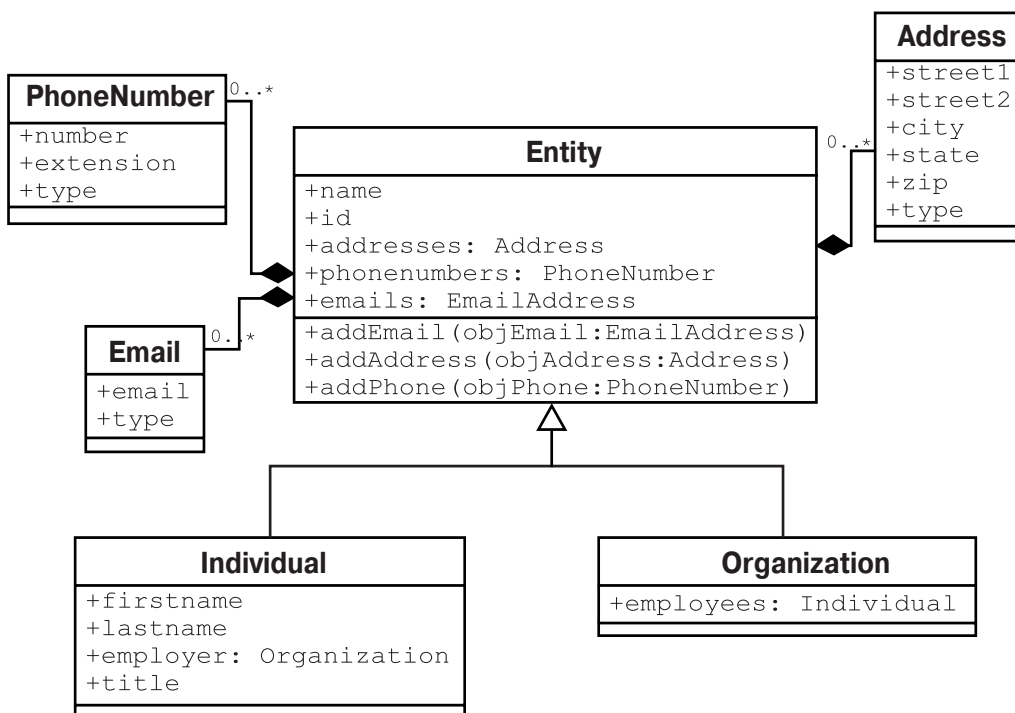


Рис. 13.11.

Класс Entity довольно понятен. Он должен хранить имя и идентификатор объекта наряду с массивами почтовых и e-mail-адресов и телефонных номеров. Этот класс не будет использоваться непосредственно. Он будет родительским классом для классов

Individual и Organization. Класс User, созданный в главе 12, хорошо подходит в качестве основы для класса Entity.

Создайте новый текстовый файл с именем class.Entity.php и введите в него следующий код, большая часть которого взята из класса User (хотя есть несколько важных изменений). Несмотря на то что этот пример кода довольно большой, он достаточно прост. Полное объяснение класса приведено сразу после кода.

```
<?php

class Entity {

    private $_properties;
    private $_changedProperties;
    private $_hDB;
    private $_emails;
    private $_addresses;
    private $_phonenumbers;

    public function __construct($entityID)
    {
        $this->_properties = array();
        $this->_changedProperties = array();

        $this->_properties['id'] = $entityID;
        $this->_properties['name'] = null;
        $this->_properties['type'] = null;
        $this->_properties['addresses'] = array();
        $this->_properties['phonenumbers'] = array();
        $this->_properties['emails'] = array();
        $this->_properties['numberOfEmails'] = 0;
        $this->_properties['numberOfPhonenumbers'] = 0;
        $this->_properties['numberOfAddresses'] = 0;

        $this->_hDB = mysql_connect('localhost', 'dbuser', 'mypassword');

        $this->_init();
    }

    private function _init()
    {
        $this->_initUser();
        $this->_initEmails();
        $this->_initAddresses();
        $this->_initPhones();
    }

    private function _initUser()
    {
        $sql = "select * from entities where entityid = $this->id";

        $rs = mysql_query($sql, $this->_hDB);
        $row = mysql_fetch_assoc($rs);

        $this->_properties['id'] = $row['entityid'];
        $this->_properties['name1'] = $row['name1'];
        $this->_properties['name2'] = $row['name2'];
        $this->_properties['type'] = $row['type'];
    }

    private function _initEmails()
```

```

    {
        //выбрать e-mail-данные...
    }

    private function _initPhones()
    {
        //выбрать данные о телефонах...
    }

    private function _initAddresses()
    {
        //выбрать данные по адресам...
    }

    function __get($propertyName)
    {
        if(!array_key_exists($propertyName, $this->_properties)){
            throw new Exception('Неправильное значение свойства!');
        }
    }

    if(method_exists($this, 'get' . $propertyName)){
        return call_user_func(array($this, 'get' . $propertyName));
    } else {
        return $this->_properties[$propertyName];
    }
}

function __set($propertyName, $value) {
    if(!array_key_exists($propertyName, $this->_properties)){
        throw new Exception('Неправильное значение свойства!');
    }
}

if(method_exists($this, 'set' . $propertyName))
{
    return call_user_func(array($this, 'set' . $propertyName), $value);
} else {
    //Если значение свойства действительно было изменено,
    //но еще не попало в массив changedProperties,
    //то добавить его в этот массив.
    if($this->_properties[$propertyName] != $value &&
    !in_array($propertyName, $this->_changedProperties)) {
        $this->_changedProperties[] = $propertyName;
    }

    //Устанавливаем новое значение свойства
    $this->_properties[$propertyName] = $value;
}
}

function __toString()
{
    return $this->name;
}

function getName()
{
    return $this->_properties['name1'];
}

function setID($val)
{

```

```
        throw new Exception('Изменять значение поля ID нельзя!');
    }

    function phonenumbers($index)
    {
        if(!isset($this->_phonenumbers[$index])) {
            throw new Exception('Указан неправильный номер телефона!');
        } else {
            return $this->_phonenumbers[$index];
        }
    }

    function getNumberOfPhoneNumbers()
    {
        return sizeof($this->_phonenumbers);
    }

    function addPhoneNumber(PhoneNumber $phone)
    {
        $this->_phonenumbers[] = $phone;
    }

    function addresses($index)
    {
        if(!isset($this->_addresses[$index]))
        {
            throw new Exception('Указан неправильный адрес!');
        } else {
            return $this->_addresses[$index];
        }
    }

    function getNumberOfAddresses()
    {
        return sizeof($this->_addresses);
    }

    function addAddress(Address $address)
    {
        $this->_addresses[] = $address;
    }

    function emails($index)
    {
        if(!isset($this->_emails[$index]))
        {
            throw new Exception('Указан неправильный email-адрес!');
        } else
        {
            return $this->_emails[$index];
        }
    }

    function getNumberOfEmails()
    {
        return sizeof($this->_emails);
    }

    function addEmail(Email $email)
    {
        $this->_emails[] = $email;
    }

    function _updateEntity()
    {

```

```

$sql = "UPDATE entities SET ";

//Сформируем SQL-оператор, создав
//массив SET-операторов, а затем объединив элементы этого массива,
//разделяя их запятыми.
$setStatements = array();
foreach($this->_changedProperties as $prop) {
    $setStatements[] = "$prop = '{$this->_properties[$prop]}'";
}
//формируем строку
$sql .= join(' ', $setStatements);

//присоединяем предложение WHERE
$sql .= " WHERE entityid = $this->id";
print $sql;
$hRes = mysql_query($sql, $this->hDB);
$intAffected = mysql_affected_rows($hRes);
}

function _createEntity()
{
    $data = array();
    $data['name1'] = "'" . mysql_escape_string($this->name1) . "'";
    $data['name2'] = "'" . mysql_escape_string($this->name2) . "'";
    $data['type'] = "'" . mysql_escape_string($this->type) . "'";

    $sql = "INSERT INTO entities (" . join(' ', array_keys($data)) . ") ";
    $sql .= " (" . join(' ', array_values($data)) . ")";
    $res = mysql_query($sql, $this->hDB);
    if (!$res) {
        //Обратите внимание, невозможно перехватить исключение, сгенерированное
        //в деструкторе (деструктор вызывает эту функцию), поэтому
        //придется использовать trigger_error для сообщения о проблеме.
        trigger_error("Во время сохранения объекта возникла ошибка!",
            E_USER_WARNING);
    }
}

function __destruct()
{
    //Проверка наличия изменений.
    if(sizeof($this->_changedProperties)) {
        if($this->id) {
            //обновляется существующая запись;
            $this->_updateEntity();
        } else {
            //создается новая запись
            $this->_createEntity();
        }
    }

    //Работа окончена, закрываем соединение с базой данных.
    mysql_close($this->_hDB);
}
}
?>

```

Чтобы создать MySQL-таблицу entities, следует ввести следующий SQL-оператор:

```

CREATE TABLE entities (
    entityid INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name1 varchar(100) NOT NULL,

```



```
name2 varchar(100) NOT NULL,
type char(1) NOT NULL
);
```

Чтобы дать возможность классу `Entity` работать с телефонными номерами, почтовыми и e-mail-адресами, в первоначальный класс `User` было внесено несколько изменений и дополнений. Теперь три частные переменные экземпляра: `$_emails`, `$_addresses` и `$_phonenumbers` — инициализируются в конструкторе как пустые массивы, в которых затем будут храниться объекты `Address`, `Email` и `PhoneNumber`, связанные с текущим объектом `Entity`.

Добавлена частная функция инициализации с именем `_init()`, которая вызывается конструктором. Функция впоследствии вызывает четыре частные `init`-функции, которые обрабатывают заполнение объекта данными. Функция `_initUser()` выполняет такой же поиск в базе данных и те же функции назначения данных, которые были в конструкторе класса `User`. Добавляются также `init`-функции для трех типов контактной информации. Эти функции пока пусты, но будут содержать код для создания объектов `Address`, `Email` и `PhoneNumber` в упомянутых выше массивах.

Для каждого типа контактной информации добавляется три функции: `getNumberOf[x]`, `add[x]` и соответственно `emails`, `addresses` или `phonenumbers`. Функция `getNumberOf[x]` сообщает количество существующих объектов в частных массивах, а `add[x]` позволяет добавлять новые объекты. Остальные функции не так очевидны, но в конечном итоге позволяют написать подобный код:

```
for($x = 0; $x < $obj->numberOfEmails; $x++ ) {
    print "<a href=\"mailto:{ $obj->emails($x)->email}\"> " .
        $obj->emails($x)->email .
        "</a> (" . $obj->emails($x)->type . ")<br>\n";
}
```

В сущности, они позволяют использовать удобный синтаксис для перебора семейства типов контактной информации и получения доступа к отдельным элементам соответствующих массивов. Несмотря на то что эти функции обозначены в UML-диаграмме как свойства, реализация их в виде функций позволяет, используя класс `Entity`, писать более четкий и простой для понимания код.

Кроме того, стоит отметить поля базы данных `name1` и `name2`, а также новую функцию `getName()`. Для организаций используется только одно поле с именем, тогда как для частных лиц — два (имя и фамилия). Чтобы приспособиться к этому, класс `Entity` содержит оба значения, но по умолчанию свойство `name` возвращает только значение поля `name1` (в котором хранится название организации). В классе `Individual` это свойство будет возвращать имя и фамилию.

Каждый объект должен отвечать за свои внутренние операции, поэтому массив `$_changedProperties` при добавлении нового типа контактной информации не обновляется. Вместо этого упомянутые классы будут сами следить за вставкой своих данных в базу и их обновлением.

Рассмотрим новый усовершенствованный деструктор. Объект теперь имеет возможность сохранять любые изменения своих данных, а также создавать новые записи. В зависимости от того, существует ли запись (элемент) или создается новая, вызывается либо функция `_saveEntity()`, либо функция `_updateEntity()`. Новая запись создается, если во время уничтожения объекта отсутствует значение свойства `id`. Можно предположить, что если свойство `id` равно `NULL`, то объект еще не имеет записи в таблице `entities` (потому что поле `entityid` определено в базе данных как `NOT NULL`), и поэтому выполняется `INSERT`-оператор, создающий новую запись. Следует отметить, что функция `setID()` генерирует ошибку при любой попытке изменить значение свойства `id`.

Сборка классов

В главе 12 было сказано, что очень полезно защищать данные в частных переменных экземпляра и использовать функции для доступа ко всем свойствам класса, использование методов `__get` и `__set` весьма упрощает этот процесс. Показанные в этой главе диаграммы класса дают понять, что все классы имеют довольно простые свойства и что для большинства из них можно применить подход, использованный при реализации класса `Entity`.

Поскольку читатели уже знают о наследовании и о том, какие преимущества оно дает в PHP, следует убрать всю специализированную функциональность из класса `Entity` и создать класс, который позволил бы повторно использовать этот код в других классах. Первоначально этот класс назывался `PropertyObject`.

Класс `PropertyObject`

Исходные классы `PropertyObject` и `Entity` не имели унифицированного средства для проверки корректности данных, поэтому требуется интерфейс для всех данных, которые можно проверить. Рассмотрим код такого интерфейса:

```
<?php
interface Validator {
    abstract function validate();
}
?>
```

Назовите этот файл `interface.Validator.php`, он будет использоваться в новом усовершенствованном классе `PropertyObject`. Приведенный ниже код показывает, как интегрировать интерфейс `Validator` в класс `PropertyObject`. Введите следующий код в файл `class.PropertyObject.php`.

```
<?php
require_once('interface.Validator.php');

abstract class PropertyObject implements Validator {

    protected $propertyTable = array(); //Пары имя/значение,
                                         //которые преобразуют свойства
                                         //в имена полей базы данных

    protected $changedProperties = array(); //Перечень свойств, которые
                                             //были изменены

    protected $data; //Реальные данные из базы

    protected $errors = array(); //Возникшие ошибки
                                  //проверки данных

    public function __construct($aData) {
        $this->data = $aData;
    }

    function __get($propertyName) {
        if(!array_key_exists($propertyName, $this->propertyTable))
            throw new Exception("Неправильное свойство \"{$propertyName}\"!");
        if(method_exists($this, 'get' . $propertyName)) {
```

```

        return call_user_func(array($this, 'get' . $propertyName));
    } else {
        return $this->data[$this->propertyTable[$propertyName]];
    }
}

function __set($propertyName, $value) {
    if(!array_key_exists($propertyName, $this->propertyTable))
        throw new Exception("Неправильное свойство \"$propertyName\"!");

    if(method_exists($this, 'set' . $propertyName)) {
        return call_user_func(
            array($this, 'set' . $propertyName),
            $value
        );
    } else {

        //Если значение свойства действительно было изменено,
        //но еще не попало в массив changedProperties,
        //то добавить его в этот массив.
        if($this->propertyTable[$propertyName] != $value &&
            !in_array($propertyName, $this->changedProperties)) {
            $this->changedProperties[] = $propertyName;
        }

        //Устанавливаем новое значение свойства
        $this->data[$this->propertyTable[$propertyName]] = $value;
    }
}

function validate()
{}

}
?>

```

Рассмотрим работу этого кода подробнее. Создается четыре защищенные переменные экземпляра. Защищенные переменные экземпляра видимы только внутри подклассов данного класса; они не видимы для кода, который использует эти объекты.

Переменная `$propertyTable` содержит таблицу преобразования имен свойств в имена полей базы данных. Часто в именах полей используются префиксы, указывающие на тип хранимых в поле данных. Например, поле `entities.sname1` может быть полем таблицы `entities`, имеющим строковый тип и содержащим имя. Однако имя `sname1` не особенно удобно использовать в качестве имени свойства объекта, поэтому необходимо обеспечить механизм преобразования имен полей базы данных в удобные имена свойств.

Переменная `$changedProperties` служит той же цели, что и в предыдущей реализации класса. Она представляет собой массив, в котором хранится список имен измененных свойств.

`$data` — ассоциативный массив имен и значений полей базы данных. Этот массив передается конструктору со структурой данных, поступающей непосредственно от функции `pg_fetch_assoc()` (или `mysql_fetch_assoc()` или любой другой `[x]_fetch_assoc()`-функции). Как будет показано далее, это значительно упрощает создание удобных объектов на основе запросов к базе данных.

В последней переменной `$errors` содержится массив имен полей и сообщений об ошибках на случай, если метод `validate` (требуемый интерфейсом `Validate`) потерпит неудачу.

Класс объявляется как абстрактный по двум причинам. Во-первых, сам по себе класс `PropertyObject` не особенно полезен. Классы, расширяющие `PropertyObject`, все равно должны выполнить некоторую работу до того, как их можно будет использовать. Вторая причина заключается в том, что необходимый метод `validate()` в классе `PropertyObject` не реализован. Поскольку этот метод в классе `PropertyObject` объявляется как абстрактный, то и сам класс должен быть абстрактным, заставляя все наследующие его классы реализовывать упомянутую функцию. Попытки использовать классы, расширяющие `PropertyObject`, но не реализующие функцию `validate`, приведут к ошибкам времени выполнения.

Рассмотрим значительно упрощенный конструктор. Все, что делает конструктор — принимает ассоциативный массив, который, скорее всего, заполняется данными в результате запроса к базе данных, и присваивает этот массив защищенной переменной `$data`. В большинстве подклассов класса `PropertyObject` конструктор необходимо переопределить и заставить его выполнять нечто более интересное.

Еще одним важным изменением является внутреннее устройство функций `__get()` и `__set()`. Так как данные хранятся в переменной `$data`, требуется возможность связывать имена свойств с соответствующими именами полей в базе данных. Эту задачу решают строки, содержащие код `$this->data[$this->propertyTable[$propertyName]]`. Выбор и присвоение значений в массиве `$data` с использованием имен полей базы данных, а не имен свойств, позволяет легко реализовать способность базы данных автоматически создавать и поддерживать перманентные объекты, которая уже была реализована в классе `Entity`.

Если назначение переменных `$data` и `$propertyTable` не совсем ясно, огорчаться не стоит, поскольку все будет понятно после изучения соответствующего примера.

Классы, представляющие типы контактной информации

Теперь можно применить класс `PropertyObject`. Ниже представлен код классов `Address`, `EmailAddress` и `PhoneNumber`.

В коде имеется ссылка на класс `DataManager`, который должен использоваться в качестве оболочки для всех необходимых функций для работы с базами данных. Это позволяет централизованно разместить весь код обработки данных. Упомянутый класс рассматривается далее.

Введите следующий код (класс `Address`) в файл с именем `class.Address.php`:

```
<?php
require_once('class.PropertyObject.php');

class Address extends PropertyObject {

    function __construct($addressid) {
        $arData = DataManager::getAddressData($addressid);

        parent::__construct($arData);

        $this->propertyTable['addressid'] = 'addressid';
        $this->propertyTable['id'] = 'addressid';
        $this->propertyTable['entityid'] = 'entityid';
        $this->propertyTable['address1'] = 'saddress1';
```

```

    $this->propertyTable['address2'] = 'saddress2';
    $this->propertyTable['city'] = 'scity';
    $this->propertyTable['state'] = 'cstate';
    $this->propertyTable['zipcode'] = 'spostalcode';
    $this->propertyTable['type'] = 'stype';
}

function validate() {
    if(strlen($this->state) != 2) {
        $this->errors['state'] = 'Пожалуйста, выберите существующий штат.';
    }

    if(strlen($this->zipcode) < 5 ||
        strlen($this->zipcode) > 10) {
        $this->errors['zipcode'] = 'Пожалуйста, введите почтовый индекс
                                   из 5 или 9 цифр';
    }

    if(!$this->address1) {
        $this->errors['address1'] = 'Поле адрес 1 обязательно для заполнения.';
    }

    if(!$this->city) {
        $this->errors['city'] = 'Поле город обязательно для заполнения.';
    }

    if(sizeof($this->errors)) {
        return false;
    } else {
        return true;
    }
}

function __toString() {
    return $this->address1 . ', ' .
           $this->address2 . ', ' .
           $this->city . ', ' .
           $this->state . ' ' . $this->zipcode;
}
}
?>

```

Поскольку основную работу берет на себя класс `PropertyObject`, в классе `Address` необходимо реализовать только два метода (функция `__toString()` реализована просто ради интереса). В конструкторе впервые используется массив `$propertyTable`. Перечень необходимых свойств класса был определен в UML-диаграмме при первоначальном проектировании приложения (в начале главы). Учитывая свойства объекта, можно также определить структуру таблицы базы данных. Как правило, для каждого поля требуется одно поле, и поскольку данный класс должен быть связан с классом `Entity`, требуется также хранить ссылку на родительский объект `Entity`. Предположим, для создания таблицы `Address` используется следующий SQL-оператор:

```

CREATE TABLE entityaddress (
    addressid int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    entityid int,
    saddress1 varchar(255),
    saddress2 varchar(255),
    scity varchar(255),
    cstate char(2),
    spostalcode varchar(10),
    stype varchar(50),

```

```

CONSTRAINT fk_entityaddress_entityid
    FOREIGN KEY (entityid) REFERENCES entity(entityid)
);

```

Соответствующее именование полей базы данных с использованием односимвольного префикса для указания типа данных позволяет узнать, какой тип данных хранится в том или ином поле. Соглашения по именованию баз данных так же важны, как и соглашения по именованию для кода.

Массив `propertyTable`, создаваемый в классе `Address`, связывает удобные имена свойств (например, `city`, `state` и `zipcode`) с менее удобными именами полей в базе данных (`scity`, `cstate` и `spostalcode`). Следует отметить, что к одному имени поля в массиве `propertyTable` можно привязать несколько имен свойств. Это позволяет обращаться к первичному ключу либо как `$objAddress->addressed`, либо как `$objAddress->id`.

Особенно примечательно в классе `Address` то, что подавляющее большинство кода направлено на реализацию бизнес-логики и проверки корректности данных. Здесь почти нет постороннего кода. Единственная задача этого класса — наполнять свои объекты данными и проверять их содержимое. За все остальное отвечают классы `DataManager` (который будет рассмотрен позднее) и `PropertyObject`.

Ниже приведен код для класса `Email`, который очень похож на предыдущий класс. Введите следующий код в файл `class.EmailAddress.php`.

```

<?php
require_once('class.PropertyObject.php');

class EmailAddress extends PropertyObject {

    function __construct($emailid) {
        $arData = DataManager::getEmailData($emailid);

        parent::__construct($arData);

        $this->propertyTable['emailid'] = 'emailid';
        $this->propertyTable['id'] = 'emailid';
        $this->propertyTable['entityid'] = 'entityid';
        $this->propertyTable['email'] = 'semail';
        $this->propertyTable['type'] = 'stype';
    }

    function validate() {
        if(!$this->email) {
            $this->errors['email'] = 'Поле email обязательно для заполнения.';
        }

        if(sizeof($this->errors)) {
            return false;
        } else {
            return true;
        }
    }

    function __toString() {
        return $this->email;
    }
}
?>

```

Здесь добавляется очень немного кода, только выборка данных из базы и создание массива `propertyTable`. Все остальное — проверка корректности данных. UML-диаграмма в данном случае также подсказывает решения о свойствах класса `Email` и структуре соответствующей таблицы в базе данных.

Таблица `entityemail` выглядит следующим образом:

```
CREATE TABLE entityemail (
    emailid int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    entityid int,
    semail varchar(255),
    stype varchar(50),
    CONSTRAINT fk_entityemail_entityid
    FOREIGN KEY (entityid) REFERENCES entity(entityid)
);
```

Класс `PhoneNumber` работает почти так же, как классы `Address` и `Email`. Ниже приведен код, который следует ввести в файл `class.PhoneNumber.php`:

```
<?php
require_once('class.PropertyObject.php');

class PhoneNumber extends PropertyObject {

    function __construct($phoneid) {
        $arData = DataManager::getPhoneNumberData($phoneid);

        parent::__construct($arData);

        $this->propertyTable['phoneid'] = 'phoneid';
        $this->propertyTable['id'] = 'phoneid';
        $this->propertyTable['entityid'] = 'entityid';
        $this->propertyTable['number'] = 'snumber';
        $this->propertyTable['extension'] = 'sextension';
        $this->propertyTable['type'] = 'stype';
    }

    function validate() {
        if(!$this->number) {
            $this->errors['number'] = 'Необходимо ввести номер телефона.';
        }

        if(sizeof($this->errors)) {
            return false;
        } else {
            return true;
        }
    }

    function __toString() {
        return $this->number .
            ($this->extension ? ' x' . $this->extension : '');
    }
}
?>
```

SQL-оператор для создания таблицы `entityphone` выглядит так:

```
CREATE TABLE entityphone (
    phoneid int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    entityid int,
    snumber varchar(20),
    sextension varchar(20),
```

```

stype varchar(50),
CONSTRAINT fk_entityemail_entityid
    FOREIGN KEY (entityid) REFERENCES entity(entityid)
);

```

Класс `DataManager` используется для наполнения собственных объектов данными, создания массива `propertyTable` и определения некоторых правил проверки данных. Функция `__toString` реализована для того, чтобы упростить вывод полных номеров телефонов.

Класс `DataManager`

Рассмотрим класс `DataManager`. В нем, как и в остальных примерах кода для работы с базами данных, используется `MySQL`, хотя такой класс вполне может работать с любой другой СУРБД, например, `PostgreSQL` или `Oracle`. Основная задача класса `DataManager` заключается в том, чтобы собрать весь код для доступа к данным в одном месте, упростив таким образом возможное изменение типа базы данных или параметров подключения к ней. Все методы класса объявлены как статические, потому что класс не использует переменные экземпляра. Следует отметить использование статической переменной в функции `getConnection()`. Это сделано для того, чтобы гарантировать, что во время запроса одной страницы будет открыто только одно подключение к базе данных. С установкой подключения к базе данных связаны большие издержки, поэтому устранение ненужных подключений способствует повышению производительности. Создайте файл `class.DataManager.php` и введите в него следующий код:

```

<?php
require_once('class.Entity.php'); //это понадобится позднее
require_once('class.Individual.php');
require_once('class.Organization.php');

class DataManager {
    private static function _getConnection() {
        static $hDB;

        if(isset($hDB)) {
            return $hDB;
        }

        $hDB = mysql_connect('localhost', 'phpuser', 'phppass')
            or die("Не удалось подключиться к базе данных!");
        $Link = mysql_select_db('sample_db') or die("Не удалось выбрать базу данных!");
        return $hDB;
    }

    public static function getAddressData($addressID) {
        $sql = "SELECT * FROM entityaddress WHERE addressid = $addressID";
        $res = mysql_query($sql, DataManager::_getConnection());
        if(! ($res && mysql_num_rows($res))) {
            die("Не удалось получить данные для адреса $addressID");
        }

        return mysql_fetch_assoc($res);
    }

    public static function getEmailData($emailID) {
        $sql = "SELECT * FROM entityemail WHERE emailid = $emailID";

```



```

$res = mysql_query($sql, DataManager::_getConnection());

if(! ($res && mysql_num_rows($res))) {
    die("Не удалось получить данные для email-адреса $emailID");
}

return mysql_fetch_assoc($res);
}
?>

```

Класс `DataManager` предоставляет структуры данных, используемые для наполнения массива `$data` в подклассах класса `PropertyObject`. Существуют отдельные функции, возвращающие данные каждого типа. Впоследствии в этот класс будут добавлены еще несколько новых функций.

Все методы класса объявлены как статические. Статические методы требуют, чтобы все переменные экземпляров были также статическими. Чтобы использовать методы статического класса, не нужно создавать экземпляр этого класса. В определенных ситуациях это имеет смысл. Например, класс `Math` предоставляет такие методы, как `squareRoot()`, `power()` и `cosine()`, и имеет свойства, среди которых есть математические константы `e` и `pi`. Все экземпляры этого класса выполняют одинаковые вычисления. Квадратный корень 2 не изменяется, число 4 в кубе всегда будет равно 64, а две упомянутые константы всегда будут оставаться константами. Нет необходимости создавать отдельные экземпляры этого класса, поскольку его состояние и свойства не изменяются никогда. Класс `Math`, реализованный таким образом, должен давать возможность вызывать все его функции статически.

Класс `DataManager` почти такой же. Все его функции самодостаточны, а все переменные экземпляра, с которыми они взаимодействуют, являются статическими. Класс не предоставляет своим пользователям никаких свойств. Таким образом, методы класса можно вызывать с помощью оператора статических методов `::`. Поскольку все созданные методы являются статическими, не нужно создавать объект с помощью оператора `$obj = new DataManager();` вместо этого можно использовать синтаксис вроде `DataManager::getEmail()`.

Следует отметить использование статической переменной функции в частном методе `_getConnection()`. (Использование статических переменных функций обсуждалось в главе 6.)

Классы `Entity`, `Individual` и `Organization`

Имея все поддерживающие классы, можно перейти к ядру приложения — к классу `Entity` и его подклассам.

Прежде всего необходимо обновить UML-диаграмму и внести изменения в иерархию объектов. Все классы кроме `DataManager`, который не является производным от какого-либо другого класса, созданы как подклассы `PropertyObject`. Класс `PropertyObject` реализует абстрактный интерфейс, который называется `Validator`. Новая UML-диаграмма показана на рис. 13.12.

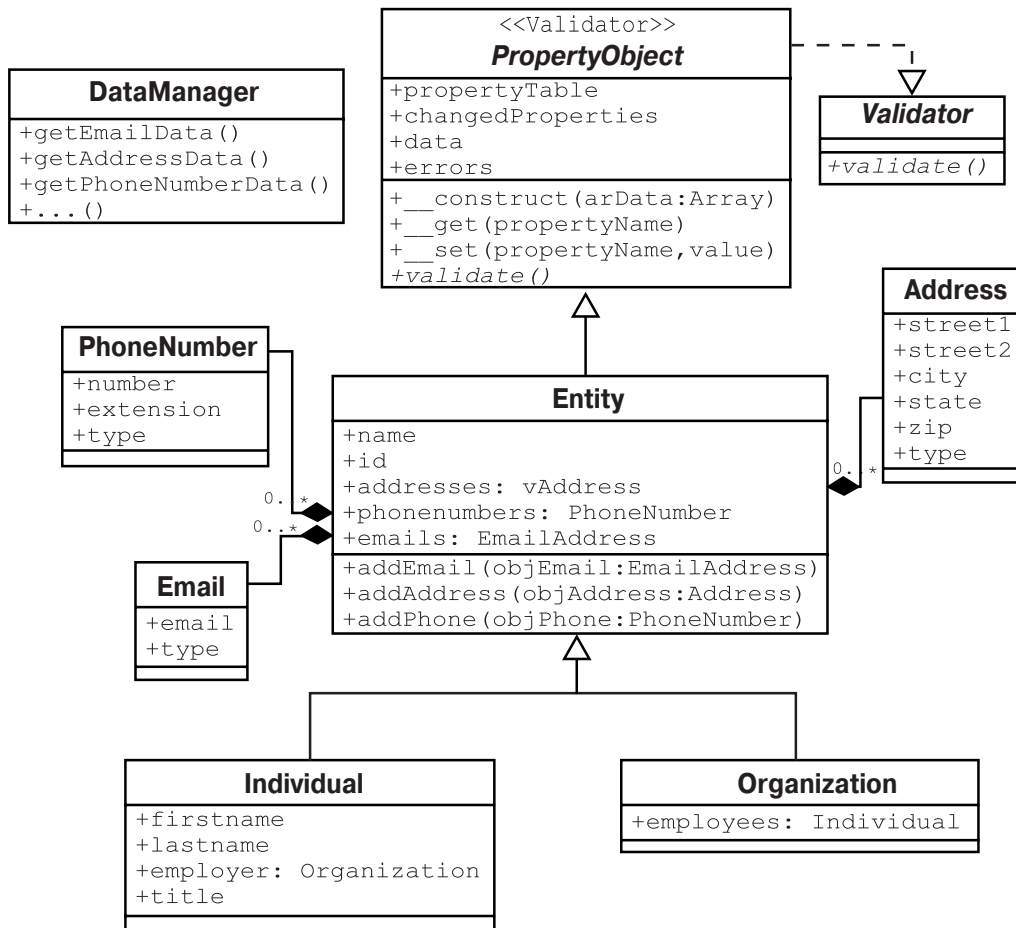


Рис. 13.12.

Теперь можно начинать разрабатывать классы Entity, Individual и Organization. Следующий код представляет собой полностью реализованный класс Entity:

```

<?php
require_once('class.PropertyObject.php');
require_once('class.PhoneNumber.php');
require_once('class.Address.php');
require_once('class.EmailAddress.php');

abstract class Entity extends PropertyObject {

    private $_emails;
    private $_addresses;
    private $_phonenumbers;

    public function __construct($entityID) {
        $arData = DataManager::getEntityData($entityID);

        parent::__construct($arData);
    }
}
  
```

```

$this->propertyTable['entityid'] = 'entityid';
$this->propertyTable['id'] = 'entityid';
$this->propertyTable['name1'] = 'sname1';
$this->propertyTable['name2'] = 'sname2';
$this->propertyTable['type'] = 'ctype';

$this->_emails = DataManager::getEmailObjectsForEntity($entityID);
$this->_addresses = DataManager::getAddressObjectsForEntity($entityID);
$this->_phonenumbers =
DataManager::getPhoneNumberObjectsForEntity($entityID);
}

function setID($val) {
    throw new Exception('Изменение значения поля id не допускается!');
}

function setEntityID($val) {
    $this->setID($val);
}

function phonenumbers($index) {
    if(!isset($this->_phonenumbers[$index])) {
        throw new Exception('Указан некорректный номер телефона!');
    } else {
        return $this->_phonenumbers[$index];
    }
}

function getNumberOfPhoneNumbers() {
    return sizeof($this->_phonenumbers);
}

function addPhoneNumber(PhoneNumber $phone) {
    $this->_phonenumbers[] = $phone;
}

function addresses($index) {
    if(!isset($this->_addresses[$index])) {
        throw new Exception('Указан неправильный адрес!');
    } else {
        return $this->_addresses[$index];
    }
}

function getNumberOfAddresses() {
    return sizeof($this->_addresses);
}

function addAddress(Address $address) {
    $this->_addresses[] = $address;
}

function emails($index) {
    if(!isset($this->_emails[$index])) {
        throw new Exception('Указан неправильный email-адрес!');
    } else {
        return $this->_emails[$index];
    }
}

function getNumberOfEmails() {
    return sizeof($this->_emails);
}

```

```

function addEmail(Email $email) {
    $this->_emails[] = $email;
}

public function validate() {
    //Общие процедуры проверки корректности данных
}

}
?>

```

Перемещение всех функций для получения и установки свойств в родительский класс `PropertyObject` упрощает класс `Entity` и гарантирует, что в нем присутствует только код, необходимый для реализации объекта.

Класс `Entity` объявлен как абстрактный, потому что сам по себе он бесполезен. Все объекты являются объектами либо класса `Individual`, либо `Organization`. Создавать объекты класса `Entity` бессмысленно. Объявление класса как абстрактного препятствует созданию объектов этого класса.

В код были добавлены обращения к нескольким новым функциям класса `DataManager`: `getEntityData()` и `get[x]ObjectsForEntity`. Функция `getEntityData()` возвращает данные, необходимые для создания объекта, так же как рассмотренные выше функции для обработки типов контактной информации. Ниже приведен код новых функций в файле `class.DataManager.php`:

```

// для краткости начало файла здесь не показывается
...
    die("Невозможно получить данные для телефона $phoneID");
}

return pg_fetch_assoc($res);
}

```

```

public static function getEntityData($entityID) {
    $sql = "SELECT * FROM entities WHERE entityid = $entityID";
    $res = mysql_query($sql, DataManager::_getConnection());
    if(! ($res && mysql_num_rows($res))) {
        die("Failed getting entity $entityID");
    }
    return mysql_fetch_assoc($res);
}
?>

```

Чтобы добавить функции `get[x]ObjectsForEntity`, необходимо поместить следующий код в конец файла `class.DataManager.php` сразу после объявления функции `getEntityData`:

```

public static function getAddressObjectsForEntity($entityID) {
    $sql = "SELECT addressid from entityaddress WHERE entityid = $entityID";
    $res = mysql_query($sql, DataManager::_getConnection());
    if(!$res) {
        die("Невозможно получить адресные данные для записи $entityID");
    }

    if(mysql_num_rows($res)) {
        $objs = array();
        while($rec = mysql_fetch_assoc($res)) {
            $objs[] = new Address($rec['addressid']);
        }
    }
}

```

```

        return $objs;
    } else {
        return array();
    }
}

public static function getEmailObjectsForEntity($entityID) {

    $sql = "SELECT emailid from entityemail WHERE entityid = $entityID";
    $res = mysql_query($sql, DataManager::_getConnection());
    if(!$res) {
        die("Невозможно получить email-адрес для записи $entityID");
    }

    if(mysql_num_rows($res)) {
        $objs = array();
        while($rec = mysql_fetch_assoc($res)) {
            $objs[] = new EmailAddress($rec['emailid']);
        }
        return $objs;
    } else {
        return array();
    }
}

public static function getPhoneNumberObjectsForEntity($entityID) {
    $sql = "SELECT phoneid from entityphone WHERE entityid = $entityID";
    $res = mysql_query($sql, DataManager::_getConnection());

    if(!$res) {
        die("Невозможно получить номер телефона для записи $entityID");
    }

    if(mysql_num_rows($res)) {
        $objs = array();
        while($rec = mysql_fetch_assoc($res)) {
            $objs[] = new PhoneNumber($rec['phoneid']);
        }

        return $objs;
    } else {
        return array();
    }
}
}

```

Все эти функции принимают в качестве параметра идентификатор записи. Они выполняют запрос к базе данных, чтобы определить, существуют ли для указанной записи номера телефонов, а также e-mail- и почтовые адреса. Если это так, функции создают массив объектов `EmailAddress`, `Address` или `PhoneNumber`, передавая идентификатор соответствующему конструктору. Затем созданный массив возвращается объекту `Entity`, где он сохраняется в соответствующей частной переменной.

Класс `Entity` выполняет всю основную работу, поэтому остается только реализовать классы `Individual` и `Organization`. Создайте файл `class.Individual.php` и введите в него следующий код:

```

<?php
require_once('class.Entity.php');
require_once('class.Organization.php');

class Individual extends Entity {

    public function __construct($userID) {

```

```

        parent::__construct($userID);

        $this->propertyTable['firstname'] = 'name1';
        $this->propertyTable['lastname'] = 'name2';
    }

    public function __toString() {
        return $this->firstname . ' ' . $this->lastname;
    }

    public function getEmployer() {
        return DataManager::getEmployer($this->id);
    }

    public function validate() {
        parent::validate();

        //специальная проверка данных частного лица
    }
}
?>

```

Коротко и ясно. Класс `Individual` устанавливает несколько новых свойств, которые упрощают доступ к имени и фамилии частного лица и позволяют не использовать довольно неудобные свойства `name1` и `name2`, определенные в классе `Entity`. Кроме того, в классе также определяется новый метод, `getEmployer()`, которому требуется новая функция в классе `DataManager`. Эта функция будет рассматриваться при описании класса `Organization`, код которого представлен ниже. Создайте файл с именем `class.Organization.php` и введите в него следующий код:

```

<?php
require_once('class.Entity.php');
require_once('class.Individual.php');

class Organization extends Entity {

    public function __construct($userID) {
        parent::__construct($userID);

        $this->propertyTable['name'] = 'name1';
    }

    public function __toString() {
        return $this->name;
    }

    public function getEmployees() {
        return DataManager::getEmployees($this->id);
    }

    public function validate() {
        parent::validate();
        //специальная проверка данных организации
    }

}

?>

```

Этот класс благодаря эффективности наследования также довольно прост. Объявляется свойство `name`, которое упрощает получение названия организации (свойство `sname2` для организаций не используется).

Чтобы добавить в класс `DataManager` функции `getEmployer()` и `getEmployee()`, необходимо добавить следующий код в конец файла `class.DataManager.php`:

```
public static function getEmployer($individualID) {
    $sql = "SELECT organizationid FROM entityemployee " .
        "WHERE individualid = $individualID";
    $res = mysql_query($sql, DataManager::_getConnection());
    if(! ($res && mysql_num_rows($res))) {
        die("Невозможно получить данные о работодателе для $individualID");
    }

    $row = mysql_fetch_assoc($res);

    if($row) {
        return new Organization($row['organizationid']);
    } else {
        return null;
    }
}

public static function getEmployees($orgID) {
    $sql = "SELECT individualid FROM entityemployee " .
        "WHERE organizationid = $orgID";
    $res = mysql_query($sql, DataManager::_getConnection());
    if(! ($res && mysql_num_rows($res))) {
        die("Невозможно получить данные о работнике для $orgID");
    }

    if(pg_num_rows($res)) {
        $objs = array();
        while($row = mysql_fetch_assoc($res)) {
            $objs[] = new Individual($row['individualid']);
        }
        return $objs;
    } else {
        return array();
    }
}
```

Две эти функции зависят от присутствия таблицы `entityemployee`, которая показана ниже:

```
CREATE TABLE entityemployee (
    individualid int NOT NULL,
    organizationid int NOT NULL,
    CONSTRAINT fk_entityemployee_individualid
        FOREIGN KEY (individualid) REFERENCES entity(entityid),
    CONSTRAINT fk_entityemployee_organizationid
        FOREIGN KEY (organizationid ) REFERENCES entity(entityid)
);
```

Эта таблица связывает частных лиц с организациями, в которых они работают. Соответствующие функции класса `DataManager` аналогичны уже описанным.

Чтобы заставить систему работать, необходима еще одна последняя функция — `getAllEntitiesAsObjects()`, метод класса `DataManager`, который перечисляет все записи в базе. Этот метод завершает всю работу, которую необходимо выполнять в объектах:

```

public static function getAllEntitiesAsObjects() {
    $sql = "SELECT entityid, type from entities";
    $res = mysql_query($sql, DataManager::_getConnection());

    if(!$res) {
        die("Невозможно получить все записи");
    }

    if(mysql_num_rows($res)) {
        $objs = array();
        while($row = mysql_fetch_assoc($res)) {
            if($row['type'] == 'I') {
                $objs[] = new Individual($row['entityid']);
            } elseif ($row['type'] == 'O') {
                $objs[] = new Organization($row['entityid']);
            } else {
                die("Неизвестный тип записи {$row['type']}!");
            }
        }
        return $objs;
    } else {
        return array();
    }
}

```

Класс `DataManager` позволяет перечислить все записи с контактной информацией в системе. Он проверяет значение поля `stуре` в таблице `entity`, чтобы определить, к какому типу относится запись — частное лицо или организация, а затем создаст объект соответствующего типа и добавляет его в массив, который возвращает описываемая функция.

Использование системы

К этому моменту уже становится очевидной реальная эффективность объектно-ориентированного подхода. В результате выполнения следующего кода (`test.php`) отображается подробный перечень всех контактных сведений в базе данных:

```

<?php
require_once('class.DataManager.php'); //включаем весь
//необходимый код

function println($data) {
    print $data . "<br>\n";
}

$arContacts = DataManager::getAllEntitiesAsObjects();
foreach($arContacts as $objEntity) {
    if(get_class($objEntity) == 'Individual') {
        print "<h1>Частное лицо - {$objEntity->__toString()}</h1>";
    } else {
        print "<h1>Организация - {$objEntity->__toString()}</h1>";
    }
    if($objEntity->getNumberOfEmails()) {
        //Есть e-mail-адреса. Выводим заголовок
        print "<h2>Email-адреса</h2>";
        for($x=0; $x < $objEntity->getNumberOfEmails(); $x++) {
            println($objEntity->emails($x)->__toString());
        }
    }
    if($objEntity->getNumberOfAddresses()) {
        //Есть почтовые адреса.
        print "<h2>Адреса</h2>";
        for($x=0; $x < $objEntity->getNumberOfAddresses(); $x++) {

```



```

        println($objEntity->addresses($x)->__toString());
    }
}
if($objEntity->getNumberOfPhoneNumbers()) {
    //Есть номера телефонов.
    print "<h2>Телефоны</h2>";
    for($x=0; $x < $objEntity->getNumberOfPhoneNumbers(); $x++) {
        println($objEntity->phonenumbers($x)->__toString());
    }
}
print "<hr>\n";
}
?>

```

В браузере страница должна выглядеть так, как показано на рис. 13.13 (естественно, данные в таблицах будут отличаться).

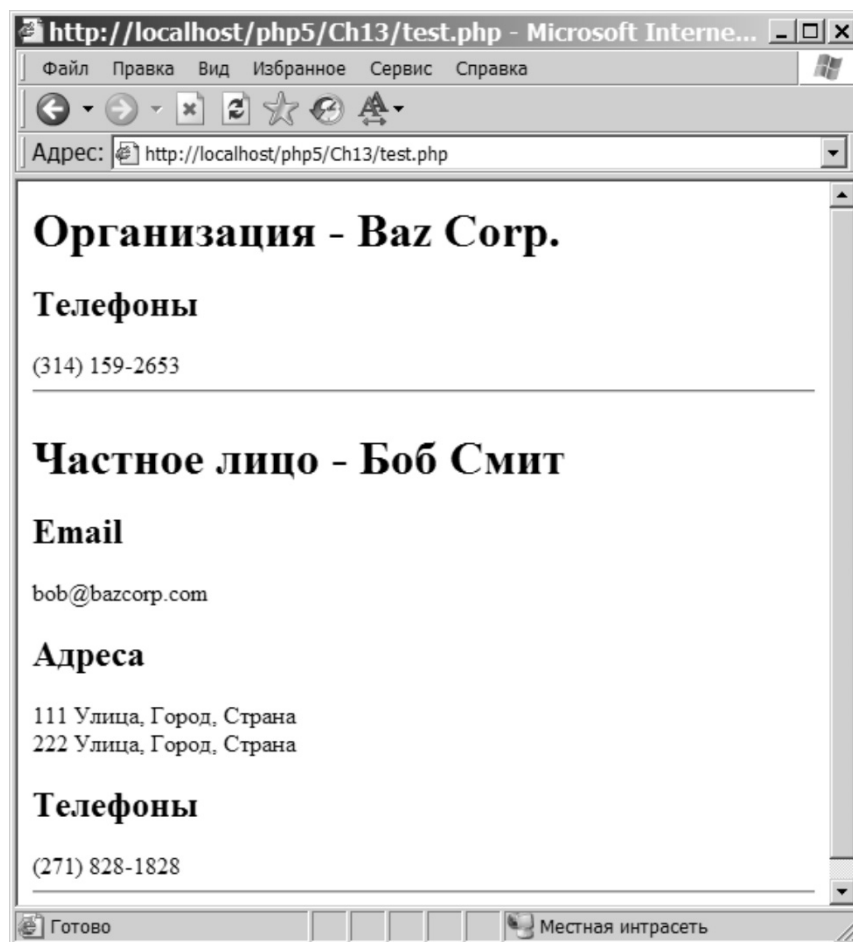


Рис. 13.13.

Используя всего лишь 36 строк кода, можно отобразить почти всю информацию по записям в системе. Здесь не показаны сведения о работодателях и работниках, это задание остается в качестве упражнения читателю. Строка, в которой вызывается функция `get_class`, должна подсказать, какой класс используется, т.е. какая функция вызывается — `getEmployer()` в классе `Individual` или `getEmployees()` в классе `Organization`.

Резюме

UML-диаграммы — важнейший инструмент для планирования сложных (и простых) приложений. Правильно сформированные диаграммы позволяют документировать сложные системы более наглядным способом, чем это можно сделать с помощью текстового описания. Использование диаграмм классов как части рутинного процесса разработки программного обеспечения значительно облегчает понимание того, как должны быть организованы классы и таблицы баз данных.

Используя все преимущества ОО-возможностей PHP5, можно быстро разрабатывать приложения и создавать легко поддерживаемый код, который будет обладать высокой степенью гибкости и расширяемости, а также сократить общее количество кода, необходимого для реализации бизнес-требований.

Разделяя программную архитектуру на уровень бизнес-логики (класс `Individual`) и уровень доступа к данным (класс `DataManager`), можно легко изменять основной источник данных, структуру таблиц и запросов, не нарушая при этом остальную часть приложения. Объекты, ответственные за реализацию бизнес-логики, не загромождены механизмом доступа к данным, присутствие которого может запутать и усложнить бизнес-правила.

14

PEAR

Иногда программисту, начинающему новый проект, может показаться, что поставленная перед ним задача слишком сложна и почти невыполнима.

Даже лучшие программисты часто испытывают трудности в поиске правильной начальной точки. В такой ситуации полезно помнить о том, что половина необходимого для проекта кода, скорее всего, уже написана.

По мере того как проекты разработчика становятся более зрелыми, он понимает, что может повторно использовать большую часть кода из предыдущих разработок (это еще одна причина использовать модульный, объектно-ориентированный подход). Разработка по типу “найти и заменить” за годы сэкономила тысячи часов работы программистов.

Конечно, возможны ситуации, когда воспользоваться нечем. Разработка некоторых компонентов может занимать чрезвычайно много времени, не внося фактически никакого полезного вклада в ядро проекта. В такой ситуации может возникнуть синдром “чистого листа” и появиться желание выпить еще одну чашку кофе, прежде чем браться за работу.

К счастью, самая трудная задача выполнима: к вашим услугам репозиторий, в котором хранятся тысячи полезных, хорошо протестированных, повторно используемых РНР-компонентов, каждый из которых можно бесплатно использовать и включать в разрабатываемые приложения.

В этой главе рассматривается репозиторий PEAR, а также объясняются причины его возникновения и описываются ситуации, в которых он может помочь. Прочтя материал главы, читатель узнает, когда следует, а когда не следует использовать репозиторий. Самое главное — научиться идентифицировать наиболее подходящие к разрабатываемому проекту компоненты и интегрировать их в приложение.

Что такое PEAR?

Репозиторий PEAR (PHP Extension and Application Repository — репозиторий PHP-расширений и приложений) был основан в 1999 году одним из самых уважаемых и продуктивных участников проекта PHP норвежцем Стигом С. Бэккенем (Stig S. Bakken). Главной целью проекта PEAR является обеспечение официально утвержденной библиотеки компонентов с открытым исходным кодом, в разработке которых принимают участие PHP-архитекторы всего мира, а также создание интегрированного средства для инсталляции этих компонентов в PHP.

Разработчики, имеющие опыт работы с Perl, могут провести параллель с архивом CPAN (Comprehensive Perl Archive Network). PEAR заимствует многие полезные функции, предложенные CPAN, включая определение и автоматическую установку связанных компонентов во время инсталляции модулей.

Компоненты или *пакеты* в сети PEAR обеспечивают несметное число функций, включая соединение с базами данных, поддержку чтения и записи необычных форматов файлов и компонентов для генерации сложного HTML-кода. Многие компоненты обладают поддержкой XML, т.е. в работе с ними можно применить навыки, полученные в ходе изучения главы 11.

В дополнение к этому, PEAR обеспечивает ряд стандартов написания кода, которых должны придерживаться все, кто вносит свой вклад в развитие PEAR. Стандарты написания кода описываются далее.

Структура PEAR

Проект PEAR разделяется на несколько блоков:

- ❑ библиотека пакетов, каждый из которых представляет отдельную область функциональности;
- ❑ диспетчер пакетов PEAR (PEAR Package Manager), используемый для установки и удаления пакетов из инсталляции PHP;
- ❑ PFC (PHP Foundation Classes — базовые классы PHP-кода) — подмножество классов в репозитории, состоящих из модулей в высшей степени стабильного и всесторонне протестированного кода, которые включаются в стандартную инсталляцию PHP;
- ❑ PECL (PHP Extension Community Library — общественная библиотека PHP-расширений) — библиотека написанных на C (а не на PHP) компонентов, которая недавно стала самостоятельным проектом (более подробная информация приведена на сайте <http://pear.php.net>);
- ❑ PEAR-стандарты написания кода, которые будут рассматриваться в этой главе далее.

Пакеты в репозитории PEAR почти всегда поставляются в виде отдельных классов (понятие классов рассматривалось в предыдущих двух главах). Поскольку пакеты предназначены для повторного использования, объектно-ориентированная методика имеет особый смысл.

Это означает, что если читатель еще не проработал материал глав 12 и 13, то сейчас это стоит сделать.

Базовые классы PHP-кода

Базовые классы PHP-кода или PFC (PHP Foundation Classes), которые включаются в обычную установку PHP, предоставляют множество важных подпрограмм, функции которых выходят за рамки внутренней функциональности PHP. Хотя эти классы, строго говоря, являются неофициальными, их широкое применение в PHP-приложениях считается стандартным и утвердившимся.

Пакеты, входящие в состав PFC, как правило, находят широкий спектр применения и используются не только для узкоспециальных задач, например, для работы с малопонятными СУБД-платформами или форматами файлов. Более того, эти пакеты всегда считаются стабильными. На момент написания книги некоторые из базовых классов распространялись в составе PHP5, включая класс Mail и класс OS Guess для определения используемой операционной системы, а также классы для создания соединений с другими сетевыми узлами. Некоторые из этих классов сами по себе могут показаться несколько непонятными, но PEAR-классы часто основательно их используют. Так как эти классы распространяются с PHP, их можно не устанавливать при установке зависимых от них PEAR-классов. PFC-классы всегда оптимизированы для той версии PHP, с которой они поставляются; там, где диспетчер пакетов может сам загружать PFC-классы, возможны проблемы, связанные с малопонятной несовместимостью между версиями PHP и PFC. Явное связывание PFC-классов с дистрибутивами PHP обеспечивает общий интерфейс к ядру PHP, и не-PFC-классы могут не зависеть от каких-либо различий между версиями PHP.

Чтобы подробнее узнать, какие модули поставляются как часть базовых классов, следует просмотреть каталог `/usr/local/lib/php` (на Unix-системах) или `C:\PHP\PEAR` (в Windows) свежей установки PHP. Документация по использованию этих классов (как и других PEAR-классов) расположена на Web-сайте PEAR <http://pear.php.net>.

Поскольку эти пакеты включаются в стандартную поставку PHP, нет необходимости устанавливать их вручную, если только они не были явно отключены при установке среды разработки.

Теоретически их можно обновлять с помощью диспетчера пакетов PEAR (о котором речь пойдет далее), но на практике они обновляются, когда появляются новые версии PHP.

Общественная библиотека PHP-расширений

Общественная библиотека PHP-расширений, или PECL (произносится *пекл*), первоначально была подмножеством репозитория PEAR, содержащим компоненты, написанные не на PHP, а на C. Логическое обоснование подобного шага состоит в том, что такие компоненты обычно должны выполняться очень быстро; скомпилированный C-код работает намного быстрее, чем PHP-код, особенно это касается операций, в которых интенсивно используются математические вычисления. PECL позволяет разработчикам воспользоваться скоростью и эффективностью C и одновременно предоставляет интерфейс к таким компонентам через стандартный PHP-синтаксис.

Теперь PECL-библиотека выделена в отдельный проект, полностью независимый от PEAR, хотя в PECL используется тот же формат и политика, что и в PEAR.

Рассмотрение библиотеки PECL выходит за рамки данной главы, однако осведомленность о PECL, несомненно, окажется полезной читателю. Хорошая отправная точка в поиске информации о PECL — Web-сайт проекта <http://pecl.php.net>.

Диспетчер пакетов PEAR

Многие из пакетов в репозитории PEAR связаны *зависимостями*. Зависимость означает необходимость инсталляции одного класса для того, чтобы другой класс успешно функционировал. Например, PEAR-компонент, предоставляющий возможность выполнять POST HTTP-запросы, зависит от существования пакета общих HTTP-функций, а этот пакет, в свою очередь, зависит от пакета TCP/IP-абстракций.

Очевидно, что, используя свежую инсталляцию PHP, разработчик может столкнуться с трудностями удовлетворения вручную всех зависимостей для определенного класса. Этот процесс значительно упрощается с помощью диспетчера пакетов PEAR (PEAR Package Manager).

Диспетчер пакетов предоставляет развитые функции для управления PEAR-пакетами в инсталляции PHP, но основная его цель заключается в том, чтобы облегчить пользователю установку и удаление PEAR-пакетов.

Диспетчер “знает” обо всех зависимостях, т.е. при установке определенного пакета, который зависит от одного или нескольких других пакетов, диспетчер также устанавливает и эти связанные пакеты. Кроме того, диспетчер “знает” об иерархии зависимостей, поэтому он гарантирует, что пакеты будут установлены в корректном порядке.

Установка пакетов с помощью диспетчера пакетов PEAR описана в этой главе далее.

Обзор PEAR-стандартов

Стандарты написания кода обсуждались в главе 6, здесь рассматриваются стилистические соглашения, используемые в этой книге в примерах кода.

“Корректный” способ написания кода обсуждается уже около 30 лет, но единого гласного решения этого вопроса все еще не найдено. PHP отчасти является удачным потому, что с момента его создания прошло сравнительно немного времени и стилистические разногласия, которые наблюдаются в других языках, еще не успели появиться. Авторы этой книги приняли решение, что примеры кода будут придерживаться стандартов, определенных проектом PEAR. Эти стандарты известны как PEAR-стандарты написания кода (PEAR coding standards, PCS). Таким образом, в книге используются такие же стандарты, какие проект PEAR требует от авторов пакетов.

Более того, если разработчик когда-либо захочет передать свои наработки проекту PEAR, с тем чтобы другие могли воспользоваться результатами его труда, то ему также придется придерживаться названных стандартов, в таком случае привычка использовать их окажется очень полезной.

На Web-сайте PEAR очень подробно рассматривается тема стандартов написания кода, поэтому здесь обсуждаются лишь некоторые аспекты этих стандартов.

Управляющие структуры, комментарии и отступы

PEAR требует, чтобы весь код оформлялся соответствующими контексту отступами с использованием подходящего числа пробелов (а не символов табуляции). В случае PEAR один уровень отступа равен четырем пробелам.

Некоторые редакторы допускают такие настройки, при которых одно нажатие клавиши <Tab> автоматически вставляет в код вместо символа табуляции четыре пробела.

Кроме отступов PEAR требует выразительного выделения конструкций. Например, если круглые скобки использовать не обязательно, но их присутствие улучшает понимание кода, то лучше все-таки их использовать. Пустые пространства должны

использоваться только для обособления отдельных строк кода и комментариев, а там, где они включаются для пояснения кода, их количество по возможности должно быть минимальным.

Фигурные скобки должны всегда использоваться в управляющих структурах (операторы `while/if/switch/for/foreach`), даже когда в блоке содержится всего одна строка и РНР-интерпретатор не требует применения скобок. Это значительно повышает читабельность кода. Открывающая скобка должна всегда находиться в той же строке, что и управляющий блок. Закрывающая скобка должна находиться в новой строке. PEAR никак не комментирует, следует ли ставить точку с запятой перед закрывающей фигурной скобкой.

С целью обеспечения четкости кода начало блока РНР-кода (т.е. `<?php`) всегда должно содержать буквы `php` (хотя формально это не обязательно).

Комментарии поощряются всегда, когда они повышают читабельность кода. Следует избегать использования предиката `#`, несмотря на то, что РНР допускает его использование. Предпочтительнее использовать С-стиль — `//` (для однострочных комментариев) или символы `/*` и `*/` (для начала и конца многострочных комментариев).

Ниже приведен пример соответствующего PEAR-стандартам кода для управляющей структуры:

```
<?php
if (($x == 14) && ($y == 19)) || $z = 24) {
    print($strMyVal. "\n");

    // Это выразительный комментарий
};
?>
```

Вызовы и определения функций

При вызове функций не должно быть пробелов между именем функции и открывающей круглой скобкой, а также между скобкой и первым аргументом функции. Если аргументы не передаются, то за открывающей скобкой сразу должна ставиться закрывающая:

```
<?php
$strMyVar = myFunction($strA, $strB, $strC);
$strAnotherVar = anotherFunctionWithNoParameters();
?>
```

При объявлении функций правила использования фигурных скобок в управляющих структурах несколько изменяются: начальная скобка блока должна находиться в новой строке сразу после имени функции и определения списка аргументов, например:

```
<?php
function myFunction($strArg1, $strArg2)
{
    return($strMyResult);
};
?>
```

Необязательные аргументы должны объявляться с использованием методики `$strArg = ''`. Необязательные аргументы всегда должны быть последними в списке аргументов.

Наконец, функция должна всегда возвращать значение, если это значение поддается интерпретации. По крайней мере, функция должна возвращать признак своего успешного или безуспешного выполнения.

Соглашения по именованию

Именованние переменных, классов и функций в языках, имеющих слабую типизацию, таких как PHP, имеет особую важность.

В этой связи PEAR-соглашения выдвигают следующие требования.

- ❑ Имена классов должны начинаться с прописной буквы, например, Publisher. Если класс имеет иерархическое расположение внутри PEAR, то это должно быть отражено в его имени, например, HTML_TreeMenu, где TreeMenu — компонент внутри HTML-иерархии PEAR.
- ❑ Имена функций должны начинаться с имени родительского пакета, за которым следует символ подчеркивания (если он применим), а затем описательное имя без символов подчеркивания. Все буквы первого слова или слога описательного имени должны быть строчными, а первая буква каждого последующего слова или слога должна быть прописной (если это применимо), например, XML_RPC_serializeData. Имена частных функций экземпляра или переменных класса, значения которых присваиваются только внутри других методов этого класса, должны начинаться с префикса `_` (например, `this->_currentStatus`).
- ❑ Имена констант должны состоять из прописных букв, для разделения слов следует использовать символ подчеркивания `$HOME_DIRECTORY`.

PEAR не определяет соглашений для именования рабочих переменных. Среди многих авторов PEAR-пакетов принято использовать венгерскую нотацию, с помощью которой в имени переменной указывается тип хранимых в ней данных; например, переменная `$strMyVariable` предназначена для хранения строки, а `$intMyNumber` — для хранения численного значения.

Авторы книги настоятельно рекомендуют принять этот метод кроме случаев, где он оказывается громоздким; например, переменные счетчиков циклов разумнее называть короткими именами `$x`, `$y` и `$z`, но такие имена не следует использовать для переменных, в которых хранятся значащие целые числа.

Установка PEAR-пакетов

Устанавливать PEAR-пакеты удивительно просто. Гораздо сложнее найти необходимый для разрабатываемого проекта пакет. Часто простое ознакомление с интерфейсом пакета характеризуется крутой кривой изучения. По мере зрелости PEAR пакеты и поставляемая с ними документация значительно улучшаются.

Поиск PEAR-пакетов на сайте pear.php.net

Хорошей отправной точкой для поиска PEAR-пакетов является начальная страница проекта PEAR, <http://pear.php.net>.

Если вы знаете, какой пакет необходимо установить, можете смело переходить к разделу “Использование диспетчера пакетов”, но если известна лишь некоторая функциональность, можно просмотреть доступные пакеты на сайте PEAR или воспользоваться поиском по ключевому слову.

Для просмотра пакетов щелкните по ссылке “Packages” (пакеты) в правом верхнем углу страницы. Посетителям сайта Yahoo или любого другого иерархического каталога интерфейс должен показаться весьма знакомым. Выберите интересующую вас категорию и просмотрите перечень всех входящих в нее пакетов.

В отличие от сайта Yahoo структура сайта PEAR не столь многоуровневая; например, если щелкнуть по ссылке “HTML”, то будут показаны все пакеты, входящие в категорию HTML; здесь нет такого понятия, как подкатегория. Ссылки на последующие страницы сводят к минимуму переполнение экрана.

К сожалению, поисковая функция сайта слабо развита — на момент написания книги поиск ключевого слова осуществлялся только в именах пакетов, а не в их описаниях. Можно ли таким способом найти необходимый пакет, решайте сами. В то же время на каждой странице сайта имеется поисковая форма, которая позволяет выполнять поиск по ключевым словам по всему сайту.

Изучение PEAR-классов и приложений

Как только необходимый пакет найден, можно щелкнуть по его имени и перейти на начальную страницу пакета. Здесь предлагается перечень версий, существующие зависимости (как родительские, так и дочерние), а также некоторые статистические данные по пакету.

Важно то, что многие пакеты имеют ссылки на страницы с подробной справочной информацией о классах, методах и функциях пакета. Посетителям сайта `www.php.net` это должно показаться удобным и знакомым.

Единственное, чего здесь не следует делать, — загружать сам пакет. Для этого лучше использовать диспетчер пакетов PEAR. Установка пакетов вручную возможна, но эта тема выходит за рамки данной главы (установка пакета вручную подробно описана на Web-сайте PEAR).

Очень важно знать, когда действительно следует прибегать к PEAR-пакетам. Хотя в репозитории содержится множество великолепного программного обеспечения, некоторые пакеты являются незавершенными или нестабильными, другие могут просто не подойти для разрабатываемого проекта. В ходе планирования PHP-проекта необходимо заблаговременно определить, в чем могут помочь репозитории, подобные PEAR, и когда они помочь неспособны. Поэтому прежде чем установить сроки проекта, следует основательно поэкспериментировать с найденными компонентами в безопасной среде. Если вы обнаружите, что создавать новый компонент придется с нуля, то почему бы не предложить этот компонент для включения в репозиторий PEAR? Это не только позволит тысячам других программистов воспользоваться результатами вашего труда, но может оказаться весьма полезным и для вас.

Установка и использование диспетчера пакетов PEAR

Диспетчер пакетов PEAR включается во все современные версии PHP и определенно во все версии PHP5.

В инсталляциях PHP в Linux и других Unix-системах диспетчер пакетов должен быть уже установлен, если только он не был отключен во время первоначальной установки PHP.

В то же время в Windows требуется еще одна дополнительная операция. (Пользователи Linux могут пропустить этот раздел.)

Помните, что при развертывании завершенного приложения на реальном сервере, например, на Web-сервере провайдера Internet-услуг (ISP), для правильной работы приложения все необходимые ему PEAR-пакеты уже должны быть установлены

провайдером. Если ISP не устанавливает PEAR-пакеты, то их можно установить вручную, хотя в этом случае автоматическое определение неудовлетворенных зависимостей будет недоступно. Более подробная информация об установке PEAR-пакетов вручную приведена на Web-сайте PEAR.

Установка диспетчера PEAR-пакетов для Windows

В рассматриваемом ниже примере предполагается, что PHP установлен в каталог C:\PHP (по умолчанию). Если это не так, необходимо исправить команды так, чтобы они соответствовали используемому каталогу инсталляции PHP.

Вызовите оболочку командной строки и введите следующую команду:

```
C:\>cd php\pear  
  
C:\php\PEAR>..\php go-pear.php  
Welcome to go-pear!
```

Сообщения сценария помогут установить диспетчер пакетов. Основное назначение сценария заключается в том, чтобы создать исполняемый файл pear, который можно будет запустить как в Linux/Unix-дистрибутивах.

Сценарий запросит параметры прокси-сервера, так как они понадобятся для загрузки и обновления базовых классов PEAR из Internet. Эти параметры записываются и используются при последующих загрузках PEAR-пакетов, поэтому очень важно не ошибиться во время их ввода. Большинству пользователей не требуется указывать прокси-сервер; в случае сомнений спросите об этих параметрах у провайдера или системного администратора.

Предлагаемые по умолчанию каталоги для установки приемлемы для большинства пользователей, поэтому в ответ на следующий запрос достаточно нажать клавишу Enter.

На вопрос об установке PEAR-пакетов, встроенных в PHP, следует ответить утвердительно.

После этого пакеты будут загружены, распакованы и, если не будет проблем с Internet-соединением, установлены в обычный каталог библиотек PHP. В каталоге C:\PHP создается исполняемый файл pear; чтобы начать работу с диспетчером пакетов, необходимо запустить этот файл. Затем сохраняются введенные параметры прокси-сервера.

Теперь можно использовать диспетчер пакетов.

Использование диспетчера пакетов

Чтобы запустить диспетчер пакетов, необходимо открыть окно командной строки (в Windows) или запустить оболочку (в Unix).

Для использования диспетчера пакетов желательно иметь права администратора или пользователя root (если такое возможно), в противном случае может выясниться, что отсутствуют права на запись файлов в каталог с библиотеками PHP.

Диспетчер пакетов предоставляет пользователю множество удобных команд, самыми полезными из них являются install, uninstall и upgrade. Ниже приведены соответствующие примеры:

```
pear install имя_компонента  
  
pear uninstall имя_компонента  
  
pear upgrade имя_компонента
```

Необходимо убедиться, что имя компонента введено правильно с соблюдением регистра символов, пробелов и символов подчеркивания. Если все было сделано правильно, то на экране должны появиться следующие строки:

```
root@genesis:~# pear install HTML_TreeMenu
downloading HTML_TreeMenu-1.1.9.tgz ...
...done: 49,213 bytes
install ok: HTML_TreeMenu 1.1.9
root@genesis:~#
```

Появление этих строк указывает на то, что все прошло нормально — диспетчер пакетов подключился к репозиторию, загрузил и установил необходимый пакет (а также все возможные пакеты, от которых он зависит).

В процессе работы диспетчера могут появляться сообщения со словом “deprecated” (устаревший). Дело в том, что некоторые пакеты в PEAR написаны для PHP4, а версия 5 иногда выводит предупреждения, если синтаксис устанавливаемого пакета не согласуется с требованиями PHP5. В большинстве случаев эти предупреждения можно просто проигнорировать; если пакет установился успешно, то все в порядке. На Web-сайте PEAR можно найти примечания о совместимости установленного пакета.

Вообще при установке PEAR соответствующий модуль должен быть помещен в каталог библиотек PHP, который обычно определяется для сервера, а не для пользователя, отсюда и необходимость иметь административные привилегии.

Могут понадобиться (как в описанном ниже примере с компонентом HTML_TreeMenu) дополнительные действия, прежде чем пакет будет интегрирован в разрабатываемый код. Например, придется скопировать файлы данных (такие как GIF, JPEG или XML-файлы конфигурации) в более удобный для проекта каталог. Подобные действия обычно описываются в документации к пакету.

Однако в девяти случаях из десяти использование диспетчера пакетов является самым простым способом добавления PEAR-пакетов в инсталляцию PHP. Диспетчер пакетов действительно удобный помощник.

Практика Проверка пригодности PEAR-компонента

Один из лучших способов изучить репозиторий PEAR и оценить пригодность какого-либо компонента заключается в установке и испытании этого компонента.

Предположим, например, что разрабатывается интранет-сайт для небольшой компании. Множество различных документов — страховки, больничные листы и сценарии праздничных мероприятий — должны быть доступны пользователям корпоративной сети. Разработчик полагает, что лучше всего реализовать на сайте сворачивающееся, иерархическое навигационное меню на основе DHTML (Dynamic HTML), похожее на меню справочного сайта Microsoft MSDN.

Мудро полагая, что путь к просветлению лежит через повторное использование кода, PHP-разработчик в поисках подходящего компонента обращается к Web-сайту PEAR, используя описанные ранее этапы. В конце концов, он находит компонент HTML_TreeMenu.

Подробная информация о пакете HTML_TreeMenu приведена на сайте PEAR http://pear.php.net/package/HTML_TreeMenu.

Используя диспетчер PEAR-пакетов, установите HTML_TreeMenu. Команды в следующем примере будут в основном одинаковыми как для Unix-, так и для Windows-

инсталляции PHP. Просто введите эти команды в командной строке (в Unix это следует сделать от имени пользователя root):

```
pear install HTML_TreeMenu
downloading HTML_TreeMenu-1.1.9.tgz ...
...done: 49,213 bytes
install ok: HTML_TreeMenu 1.1.9
```

Если все выглядит нормально, то можно двигаться дальше и испытывать только что установленный пакет.

Чтобы генерировать дерево навигации исключительно с помощью PHP-операторов, класс `HTML_TreeMenu` можно использовать без каких-либо изменений. Однако гораздо полезнее было бы заставить его генерировать дерево навигации из XML-файла. Это позволило бы часто обновлять навигационные ссылки, вообще не меняя PHP-кода. В более крупных проектах это также означает, что не знакомые с PHP сотрудники компании смогут обновлять навигационное меню, не беспокоя PHP-программистов.

Класс `HTML_TreeMenu` не имеет собственной поддержки генерации навигационных меню из XML-файлов. В версии для PHP4 рекомендуется устанавливать дополнительный класс (не входящий в репозиторий PEAR), который предоставляет такую поддержку. Однако этот дополнительный класс не обновляется и не поддерживает радикально измененную в PHP5 поддержку XML. Поэтому, чтобы добавить соответствующую функциональность, придется написать собственный класс поддержки (он описывается в этой главе далее). Не стоит опасаться адаптации или расширения PEAR-классов, если это позволяет лучше реализовать собственные замыслы; нет предела совершенству.

Очевидно, что первым этапом является чтение документации. Просматривая страницу пакета на сайте PEAR, нетрудно убедиться, что на ней отсутствует важная ссылка на документацию (пользовательская документация имеется не во всех пакетах). Отчаиваться не стоит — не все потеряно.

Рассмотрим содержимое каталога `/usr/local/lib/php/docs/HTML_TreeMenu` в Unix или `C:\PHP\PEAR\pear\DOCS\HTML_TreeMenu` в Windows. Здесь есть весьма полезный файл `HTML_TreeMenu.pdf`, который можно открыть с помощью Adobe Acrobat. В этом документе описаны некоторые подробности о различных свойствах и методах класса, т.е. API-интерфейс класса. Кроме того, в документе есть примеры использования класса. Ввиду упомянутой выше несовместимости следует игнорировать рекомендации о разборе XML-файлов.

Чтобы продолжить работу с пакетом, создайте файл `treemenutest.xml`. Для простоты используйте XML DTD-определение (определение типа документа), упомянутое в документации к `HTML_TreeMenu`. Убедитесь, что созданный файл сохранен в том каталоге, к которому PHP имеет доступ, например, `C:\inetpub\wwwroot\xml` в Windows или `~/public_html/xml` в Unix. Если необходимо обеспечить безопасность, то хранить этот файл там, где посетители сайта могут получить к нему доступ посредством браузера, не следует.

```
<?xml version="1.0" encoding="windows-1251"?>
<treemenu>
  <node text="В начало" icon="folder.gif" link="treemenutest.php" />
  <node text="Здоровье и страховка" icon="folder.gif">
    <node text="О больничных и страховке" icon="document.gif" link="#" />
    <node text="Несчастные случаи" icon="document.gif" link="#" />
    <node text="Болезнь" icon="document.gif" link="#" />
    <node text="Длительная болезнь" icon="document.gif" link="#" />
  </node>
</treemenu>
```

Поскольку все, что в данном случае требуется сделать, это оценить пригодность компонента, в настоящий момент не обязательно включать большое количество элементов в XML-файл — приведенный здесь код вполне достаточен для целей тестирования.

Следует отметить ссылки на несколько GIF-файлов. Класс `HTML_TreeMenu` для создания знаков “плюс”, “минус” и т.д. использует еще несколько таких файлов, которые явно в XML-файле не упоминаются. Эти файлы включены в дистрибутив, и их можно скопировать в каталог для изображений, расположенный ниже корневого каталога, в котором сохранен следующий код.

Теперь следует сформировать навигационную страницу. Создайте файл `tree-menutest.php` и введите в него следующий код:

```
<HTML>
<HEAD>
  <SCRIPT LANGUAGE="Javascript" SRC="TreeMenu.js"></SCRIPT>
</HEAD>
<BODY>
  <?php
    require_once('HTML/TreeMenu.php');
    require_once('xmlhtmltree.phpm');
    $objXMLTree = new XMLHTMLTree("treemenutest.xml");
    $objXMLTree->GenerateHandOffs();
    $objXMLTree->ParseXML();
    $objTreeMenu = $objXMLTree->GetTreeHandoff();           ?>
    <H1>Тестирование древовидного меню</H1>
    <HR>
    <?php
      $objTreeMenu->printMenu();
    ?>
  </BODY>
</HTML>
```

Здесь кроме `PEAR`-пакета `HTML_TreeMenu` подключается новый пакет, `xmlhtmltree.phpm`. Это упомянутый ранее вспомогательный класс; он конвертирует XML-код в PHP-операторы, позволяя, таким образом, использовать в классе `HTML_TreeMenu` XML-код.

Ниже представлен код вспомогательного компонента. Сохраните этот код в файле `xmlhtmltree.phpm` в том же каталоге, что и `treemenutest.php`. Пока разбираться в его работе не стоит, это можно сделать позднее.

```
<?
class XMLHTMLTree {
  private $xml_content;
  private $hanTreeHandoff;
  private $objHTMLTreeMenu;

  function __construct($strXMLSourceFile = "", $strXMLSource = "") {
    if ($strXMLSourceFile) {
      $this->xml_content = implode(" ", @file($strXMLSourceFile));
    } else {
      $this->xml_content = $strXMLSource;
    };
    $this->objHTMLTreeMenu = new HTML_TreeMenu();
    $this->depth = 0;
  }

  function ParseXML() {
    $strXML = $this->xml_content;
    $objDOM = simplexml_load_string($strXML);
    foreach ($objDOM->node as $thisNode) {
```

```

        $this->_ParseNode($thisNode);
    };
}

private function _ParseNode(&$objNode, &$arPoint = "") {
    // Добавляем узел
    if (!$arPoint) {
        $objNewNode = new HTML_TreeNode(array('text' => $objNode['text'],
        'link' => $objNode['link'], 'icon' => 'folder.gif', 'expandedIcon' =>
        'folder-expanded.gif', 'expanded' => false));
        $newArPoint = &$objNewNode;
    } else {
        $newArPoint = &$arPoint->addItem(new HTML_TreeNode(array('text' =>
        $objNode['text'], 'link' => $objNode['link'], 'icon' => 'folder.gif',
        'expandedIcon' => 'folder-expanded.gif')));
    };
    // Проверяем, есть ли в оригинале дочерние узлы
    foreach ($objNode->node as $thisNode) {

        if ($thisNode['text']) {
            $this->_ParseNode($thisNode, $newArPoint);
        };
    };
    if (!empty($objNewNode)) {
        $this->objHTMLTreeMenu->addItem($objNewNode);
    };
}

function GenerateHandOffs() {
    // Создаем класс представления
    $this->hanTreeHandoff = &new HTML_TreeMenu_DHTML($this->objHTMLTreeMenu,
    array('defaultClass' => 'treeMenuDefault'));
    $this->hanTreeHandoff->images = 'images'; }

function GetTreeHandoff() {
    return($this->hanTreeHandoff);
}

}
?>

```

Единственное что в этом коде можно изменить — расположение графических файлов для визуализации сворачивающихся меню. Если желательно (или требуется) сохранять эти файлы в другом каталоге, то следует изменить строку:

```
$this->hanTreeHandoff->images = 'images';
```

и указать в ней необходимый путь к изображениям. Изображения, показанные на рис. 14.1 ниже, включены в дистрибутив пакета HTML_TreeMenu и находятся в подкаталоге /usr/local/lib/php/data/HTML_TreeMenu (Unix) или C:\PHP\PEAR\PEAR\DATA\HTML_TreeMenu (Windows).

Необходимо пройти последний этап, прежде чем класс можно будет посмотреть в действии. В дистрибутив включен файл TreeMenu.js, содержащий библиотеку JavaScript-функций. Она располагается в том каталоге, который доступен PHP, но недоступен Web-браузеру (JavaScript-файлы просто передаются Web-браузеру, а PHP-файлы предварительно обрабатываются). Чтобы обойти это препятствие, можно скопировать JavaScript-файл из каталога /usr/local/lib/php/data/HTML_TreeMenu (или его Windows-эквивалента) в тот каталог, в котором находится только что созданный PHP-файл treemenutest.php.

Запустите Web-браузер и вызовите в нем страницу `treemenutest.php`. Результат показан на рис. 14.1.

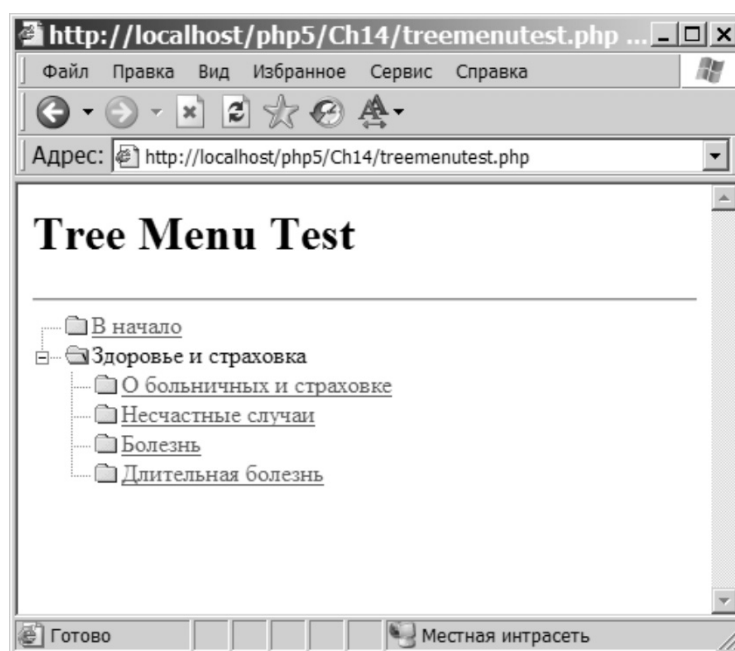


Рис. 14.1.

Можно поэкспериментировать с меню. Разверните и сверните раздел `Здоровье и страховка`. Если все работает, примите поздравления! Вы только что воспользовались первым PEAR-модулем.

В случае возникновения проблем сначала необходимо просмотреть протокол ошибок PHP. В зависимости от того, как настроен PHP-интерпретатор, ошибки могут отображаться либо непосредственно в браузере, либо в журнале ошибок Web-сервера. В любом случае следует просмотреть все пункты, которые могут помочь в диагностике проблемы. Проблема, вероятно, заключается в простой опечатке (в таком случае ее легко устранить) либо PHP не может найти класс `HTML_TreeMenu`. В последнем случае, скорее всего, в конфигурации PHP есть какая-то ошибка, и придется проверить этапы, описанные ранее в этой главе.

Как это работает

Рассмотрим работу пакета `HTML_TreeMenu`, связанную с преобразованием простого XML-файла в работоспособное навигационное меню.

Подключение пакета

В начале используется функция `require_once`, заставляющая PHP подключить код из PEAR-пакета `HTML_TreeMenu`.

Кроме того, необходимо подключить вспомогательный класс `xmlhtmltree.php`, который позволит передавать XML-код классу `HTML_TreeMenu`.

Есть две причины использования `require_once` вместо других PHP-функций подключения файлов:

- ❑ если пакет не будет найден, то PHP сгенерирует неисправимое исключение и остановит выполнение сценария;
- ❑ если пакет уже был подключен другим блоком кода выполняемого сценария, то во второй раз он не будет подключаться, что значительно повышает производительность.

Работа с классами

После подключения пакета необходимо создать экземпляры классов.

Сначала создается объект класса `HTML_TreeMenu`:

```
$objXMLTree = new XMLHTMLTree("treemenutest.xml");
```

Если вместо `require` использовать `include`-функцию для подключения пакета, то в случае отсутствия соответствующего пакета PHP не остановит выполнение сценария до тех пор, пока не достигнет этой строки. В данном примере, где пакет используется сразу после подключения, это не большая проблема. Однако в реальных приложениях, где между подключением и использованием пакета создаются записи в базе данных, это действительно очень важно. Примером такого приложения может быть Web-магазин, в котором в результате одного заказа создается несколько записей в базе данных. Если подключенный PEAR-класс не используется, например, в течение времени, необходимого для создания половины записей в базе данных, то после сбоя приложения будет создана только половина записей. Это не только неаккуратно в техническом смысле, но и может создать для использующей сайт компании реальные административные проблемы при определении того, насколько полно был оформлен заказ, оплатил ли клиент товар и т.д. На практике для предотвращения таких проблем можно использовать транзакции баз данных, но гораздо лучше гарантировать то, что в случае отсутствия необходимого компонента на этапе подключения генерируется неисправимая ошибка и остальная часть сценария не выполняется понапрасну.

Следует подчеркнуть, что здесь PEAR-класс не используется непосредственно — создается экземпляр вспомогательного класса, которому сообщается путь к входному XML-документу.

Далее, вспомогательный класс должен создать необходимый `DHTML_HTML_TreeMenu`-объект, т.е. — “переключатель”. Этот объект имеет сложную связь с объектом класса `HTML_TreeMenu`, поэтому, когда в объект дерева добавляется узел, этот узел также добавляется в переключатель.

```
$objXMLTree->GenerateHandOffs();
```

Внутренняя работа этого метода будет рассмотрена далее при подробном изучении вспомогательного класса.

Затем вспомогательный класс фактически выполняет разбор XML-кода во входном документе. Для этого используется встроенное расширение `SimpleXML`, удобно преобразующее XML-код в иерархию объектов, которую впоследствии можно будет легко обрабатывать. Хотя эту иерархию обрабатывает вспомогательный класс, для добавления узлов (по сути пунктов в списке) его собственному объекту класса `HTML_TreeMenu` он использует методы, предоставленные исходным PEAR-классом (`HTML_TreeMenu`).

Теперь вспомогательный класс имеет объект, готовый к созданию динамического HTML-кода, отображающего привлекательное навигационное меню. Можно было бы

заставить класс делать это непосредственно, но в таком случае он ждал бы своей очереди в течение всего времени жизни сценария. Используемый подход позволяет генерировать сначала остальную часть страницы. В примере имеются только некоторые основные HTML-заголовки. Часто в реальных приложениях выполняется большая работа до того, как потребуются навигационное меню.

Сначала необходимо получить DHTML HTML_TreeMenu-объект от вспомогательного класса:

```
$objTreeMenu = $objXMLTree->GetTreeHandoff();
```

Чтобы этот объект создал требуемый HTML-код, нужно вызвать один метод:

```
$objTreeMenu->printMenu();
```

который сгенерирует необходимый для меню код.

Еще любопытнее просмотреть исходный код меню из Web-браузера. Фактически PHP не генерирует HTML-код вообще — генерируется только JavaScript-код. PHP в данном случае используется для наполнения JavaScript-массива — его собственного объекта древовидного меню, который будет использоваться Web-браузером для отображения меню. Фактическая логика для генерации DHTML-кода содержится в файле `treeMenu.js`. Использование динамической серверной логики для генерации статической клиентской JavaScript-логики в наши дни становится все более популярным, а рассматриваемый здесь пакет предоставляет наглядный пример, объясняющий такую популярность. Это позволяет не только воспользоваться преимуществами невероятно мощных PHP-методик синтаксического анализа XML-кода для определения того, какая информация будет отображаться в Web-браузерах, это также позволяет полностью использовать преимущества сегодняшнего поколения Web-браузеров при определении того, *как* эта информация должна отображаться. Такое разделение уровней подталкивает более современную методику программирования, которая называется MVC (Model, View, Controller — модель, отображение, контроллер). В книге эта методика применяется мало, но очень полезно выработать привычку ее использовать.

Работа DHTML выходит за рамки данной книги, но заинтересованным читателям рекомендуется посетить Web-сайт Дена Стейнмана (Dan Steinman) Dynamic Duo www.dansteinman.com/dynduo/.

Вспомогательный класс

Рассмотрим вспомогательный класс `xmlhtmltree.php` более подробно. Как отмечалось ранее, этот класс позволяет использовать в качестве источника данных для создания пунктов древовидного меню XML-код вместо PHP-кода.

Следовательно, задача класса заключается в том, чтобы принять правильно сформированный XML-документ, создать объект класса HTML_TreeMenu, а затем обойти все XML-элементы, одновременно вызывая предоставленные этим объектом методы, и заполнить, таким образом, объект исключительно на основании XML-документа.

Вспомогательный класс используется в этой главе везде, где упоминается HTML_TreeMenu, поэтому следует рассмотреть работу каждого его метода.

Сначала рассмотрим переменные экземпляра:

```
private $xml_content;  
private $hanTreeHandoff;  
private $objHTMLTreeMenu;
```

Первая переменная содержит XML-содержимое, которое хранится в виде строки, вторая представляет собой переключатель — ссылку на DHTML-версию HTML_TreeMenu-объекта, а третья — сам HTML_TreeMenu-объект.

Конструктор принимает один из двух возможных параметров: либо путь к XML-файлу, либо строку, содержащую XML-код. Используя оператор `= ""`, можно сообщить PHP о том, что оба параметра являются необязательными. Если указывается исходный XML-файл, сценарий извлекает его содержимое. В любом случае исходный XML-код сохраняется впоследствии в переменной `xml_content` и подготавливается, таким образом, к разбору.

```
function __construct($strXMLSourceFile = "", $strXMLSource = "") {
    if ($strXMLSourceFile) {
        $this->xml_content = implode("", @file($strXMLSourceFile));
    } else {
        $this->xml_content = $strXMLSource;
    };
    $this->objHTMLTreeMenu = new HTML_TreeMenu();
}
```

Кроме того, в конструкторе создается объект класса `HTML_TreeMenu`, который сохраняется как переменная экземпляра класса (`objHTMLTreeMenu`).

Метод `GenerateHandOffs` создает отображаемый `HTML_TreeMenu`-объект (DHTML-код), к которому затем можно получить доступ с помощью метода `GetTreeHandoff`.

```
function GenerateHandOffs() {
    // Создаем класс отображения
    $this->hanTreeHandoff = &new HTML_TreeMenu_DHTML($this->objHTMLTreeMenu,
array('defaultClass' => 'treeMenuDefault'));
    $this->hanTreeHandoff->images = 'images';
}
```

Синтаксис для отображения DHTML-кода объекта меню взят непосредственно из документации `HTML_TreeMenu`. Чтобы программа могла находить изображения, применяемые для генерации навигационного меню, здесь также присваивается значение свойству `images`.

Метод `ParseXML` начинает разбор XML-содержимого вспомогательного класса.

```
function ParseXML() {
    $strXML = $this->xml_content;
    $objDOM = simplexml_load_string($strXML);
    foreach ($objDOM->node as $thisNode) {
        $this->_ParseNode($thisNode);
    };
}
```

Сначала этому методу передается XML-код. Это не обязательно, но ясность кода такой прием повышает.

Затем используется SimpleXML-метод `simplexml_load_string` для загрузки XML-структуры и ее представления в виде псевдообъектной структуры, которая сохраняется в переменной `$objDom`.

Далее вызывается метод `foreach` для обхода семейства узловых “объектов”, которые доступны благодаря SimpleXML. Ссылка на каждый из этих объектов передается частному методу `_ParseNode`, который недоступен для кода вне вспомогательного класса.

```
private function _ParseNode($objNode, &$arPoint = "") {
    // Добавляем узел
    if (!$arPoint) {
        $objNewNode = new HTML_TreeNode(array('text' => $objNode['text'], 'link' =>
            $objNode['link'], 'icon' => 'folder.gif', 'expandedIcon' =>
            'folder-expanded.gif', 'expanded' => false));
        $newArPoint = &$objNewNode;
    } else {
        $newArPoint = &$arPoint->addItem(new HTML_TreeNode(array('text' =>
```

```

        $objNode['text'], 'link' => $objNode['link'], 'icon' =>
        'folder.gif', 'expandedIcon' => 'folder-expanded.gif')));
    };
    // Проверяем, есть ли в оригинале дочерние узлы
    foreach ($objNode->node as $thisNode) {
        if ($thisNode['text']) {
            $this->_ParseNode($thisNode, $newArPoint);
        }
    };
    if ($objNewNode) {
        $this->objHTMLTreeMenu->addItem($objNewNode);
    };
}

```

Метод `_ParseNode` рекурсивно вызывает сам себя. Рекурсия лучше всего описывается как средство определенного метода выполнять выборочные вызовы самого себя с целью обхода структуры данных с неизвестной степенью вложенности. Например, если есть двумерный массив чисел, то чтобы обойти его, понадобится два `for`-цикла — вероятно, в одном из них будет использоваться переменная счетчика *x*, а в другом *y*. Аналогично, если есть трехмерный массив чисел, придется использовать три вложенных цикла со счетчиками *x*, *y* и *z*. Однако в этих двух примерах число измерений постоянно и известно. В XML структура может иметь любое число дочерних узлов, каждый из которых в свою очередь тоже может иметь любое количество дочерних узлов и т.д. Использование рекурсии гарантирует, что каждый отдельный узел обрабатывается новым экземпляром метода каждый раз, когда находится новое множество дочерних узлов; как только множество дочерних узлов обработано, родительский экземпляр продолжает работу с той точки, в которой прекратилась обработка дочерних узлов.

Эта методика весьма четко и просто работает в примере с генерацией меню. Для каждого узла в XML-коде в `HTML_TreeMenu`-объекте с помощью методов, описанных в документации к данному пакету, создается соответствующий узел. Затем выполняется проверка существования в генерируемом в текущий момент времени узле дочерних узлов. Если дочерние узлы есть, каждый из них в свою очередь передается новому экземпляру метода `_ParseNode`, выполняющему точно такой же процесс с дочерним узлом. Ссылка на родительский узел (если он есть) также передается методу, поэтому метод знает, к какому узлу в создаваемой иерархии следует добавить дочерний объект.

Наконец, рассмотрим метод `GetTreeHandoff()`.

```

function GetTreeHandoff() {
    return($this->hanTreeHandoff);
}

```

Все очень просто. Этот метод возвращает ссылку на DHTML-код, сгенерированный `HTML_TreeMenu`, доступный благодаря использованному выше методу `GenerateHandOffs()`. Структура, возвращаемая этим методом, может содержать собственный метод `printMenu`, который при необходимости вызывается для генерации HTML- и JavaScript-кода, делающего меню видимым в окне браузера.

Использование вспомогательных классов иногда бывает неизбежным. Если разработчик многократно использует собственноручно написанный класс, то он может передать этот класс в репозиторий PEAR, указав в качестве зависимости исходный PEAR-класс, на котором базируется его собственная разработка. О том, как это делается, подробно рассказано на Web-сайте проекта PEAR.

Использование PEAR-пакетов

Итак, вы уже знаете, как найти необходимый PEAR-пакет для использования в разрабатываемом проекте, как установить этот пакет и все пакеты, от которых он зависит, а также как интегрировать PEAR-пакеты в собственный код. Теперь можно применить полученные знания на практике и создать работоспособное приложение, используя один PEAR-компонент и несколько более распространенных встроенных в PHP подпрограмм.

Практика Создание приложения с использованием одного PEAR-компонента

Разрабатываемое в этом примере приложение предназначено для использования на персональном Web-сайте, демонстрирующем посетителям перечень недавно прослушанных музыкальных композиций. Приложение работает, проверяя коллекцию MP3-файлов. В настоящее время коллекционирование MP3-музыки весьма популярно, во многих таких коллекциях содержатся тысячи песен. MP3 — самый популярный формат кодирования музыки для прослушивания на компьютере.

Это PHP-приложение предназначено для работы на настольном компьютере, поскольку для его работы требуется доступ к хранилищу ежедневно прослушиваемых MP3-файлов. В силу многих причин такое приложение на практике может оказаться бесполезным, но оно вполне удовлетворительно подходит на роль рабочего примера автономного приложения.

Предположим, пользователь хранит MP3-файлы в каталоге C:\MP3. Если это не так (или приложение должно работать на Unix-системе), то можно изменить в коде значение переменной \$MY_MP3_DIR.

PEAR-пакет: MP3_ID

Пакет MP3_ID используется для считывания из MP3-файла информации, которая называется IDv3-тег. В IDv3-теге содержится информация о жанре данной композиции, дате, стране, где она была написана, а также многие другие сведения. IDv3-тег присутствует в каждом MP3-файле, но может быть пустым — это в первую очередь зависит от программы, которая использовалась для создания MP3-файла. Для целей данного проекта предположим, что все MP3-файлы имеют правильно сформированные IDv3-теги.

Более подробную информацию о пакете MP3_ID можно получить на странице http://pear.php.net/package/MP3_ID.

Пакет предоставляет методы как для чтения, так и для записи IDv3-тегов, но в рассматриваемом здесь примере используются только методы чтения, позволяющие получить определенную информацию о MP3-файлах в коллекции, а точнее, имя исполнителя и название песни.

Пакет устанавливается как обычно:

```
root@genesis:~# pear install MP3_ID
downloading MP3_Id-1.0.tgz ...
...done: 7,517 bytes
install ok: MP3_Id 1.0
root@genesis:~#
```

Приложение

Рассматриваемое приложение создается в виде одной PHP-страницы, на которой генерируется перечень из пяти верхних MP3-файлов в каталоге, отсортированном по времени последнего доступа к файлу (что теоретически должно совпадать с порядком

их проигрывания). Если в каталоге найдено больше пяти файлов, то отображаются только первые пять из них.

Создайте файл с именем `mp3id.php` и поместите его в каталог, доступный Web-серверу. Введите в файл следующий код. Не забудьте при этом изменить значение переменной `$MY_MP3_DIR` так, чтобы оно указывало на каталог, в котором действительно хранятся MP3-файлы (убедитесь, что в этом каталоге есть несколько MP3-файлов). Не беспокойтесь, если не все понятно — позднее работа сценария будет рассмотрена подробно.

```
<?php
require_once 'MP3/Id.php';

static $MY_MP3_DIR = "/home/ed/mp3"; # Измените это значение!
$objDir = dir($MY_MP3_DIR);
$numFiles = 0;
$fileTimeHash = Array();
# Пройти по всем найденным файлам в списке
while (false !== ($strEntry = $objDir->read())) {
    # Проверить, является ли текущий файл MP3-файлом,
    # а не каталогом или файлом другого типа!
    if (eregi("\.mp3$", $strEntry)) {
        $fileTimeHash[$strEntry] = filetime($MY_MP3_DIR . "/" . $strEntry);
        $numFiles++;
    }
};

# Отсортировать список файлов по дате последнего доступа и
# поместить список в обычный массив для последующего отображения
arsort($fileTimeHash);
$fileList = Array();
$thisArrayIndex = 0;
foreach ($fileTimeHash as $strFilename => $intAccessTime) {
    $fileList[$thisArrayIndex]["FILENAME"] = $strFilename;
    $fileList[$thisArrayIndex]["ACCESSED"] = $intAccessTime;
    $thisArrayIndex++;
};

# Если найдено более 5 MP3-файлов, показать только первые 5
if ($numFiles > 5) {
    $numFiles = 5;
};

?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Моя MP3-коллекция</title>
</head>
<body>
<H1>Моя MP3-коллекция </H1>
<HR>
Нижe перечислены пять верхних песен, которые я недавно прослушал
<BR><BR>
<table border="1" cellpadding="3" cellspacing="3">
<tr>
<td>Место</td>
<td>Исполнитель</td>
<td>Название композиции</td>
<td>Дата последнего прослушивания:</td>
</tr>
</tr>
<?php
$objMP3ID = new MP3_Id();
for ($i = 0; $i<=($numFiles)-1; $i++) {
?>
```

```

<tr>
<?php
    $strThisFile = $arFileList[$i]["FILENAME"];
    $strPath = $MY_MP3_DIR . "/" . $strThisFile;
    $intResult = $objMP3ID->read ($strPath);
    $strArtist = $objMP3ID->getTag ("artists", "Unknown Artist");
    $strName = $objMP3ID->getTag ("name", "Unknown Track");
    $intAccessTime = $arFileList[$i]["ACCESSED"];
?>
    <td><?=$i+1?></td>
    <td><?=$strArtist?></td>
    <td><?=$strName?></td>
    <td><?=$date("m/d/Y H:i", $intAccessTime)?></td>
</tr>
<?php
    }
?>
</body>
</html>

```

Запустите Web-браузер и откройте в нем только что созданную PHP-страницу. Результат должен быть похожим на рис. 14.2.

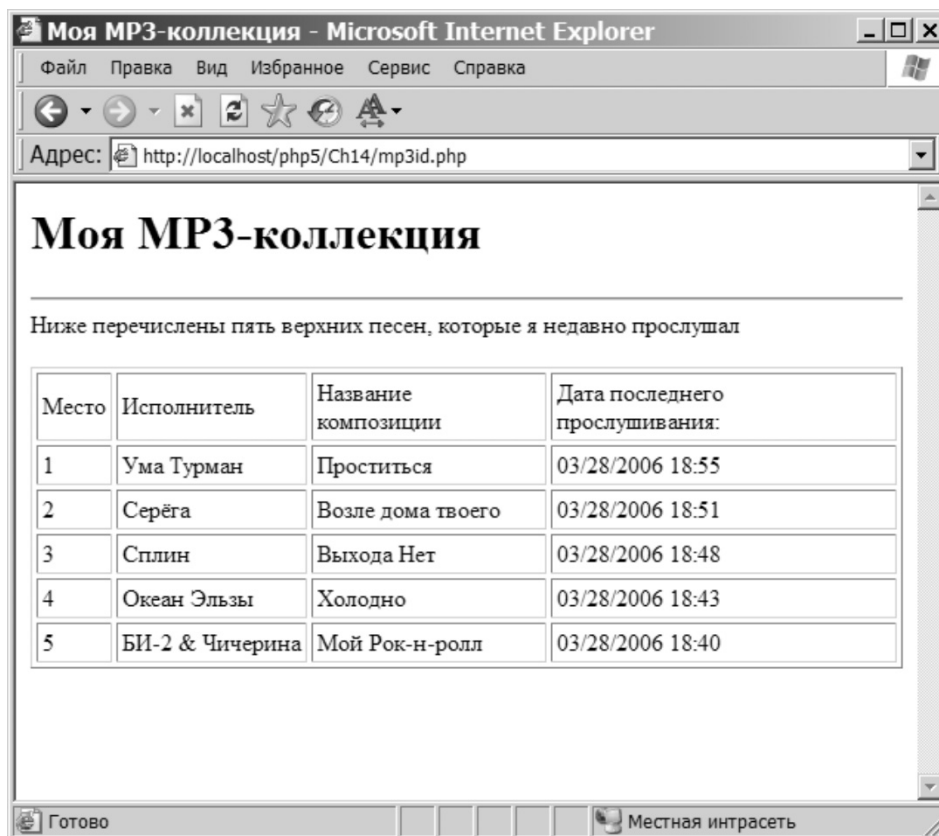


Рис. 14.2.

Как это работает

Как и в примере с пакетом `HTML_TreeMenu`, всю основную работу приложение выполняет в начале, а затем в случае необходимости отображает информацию, аккуратно вставленную в HTML-код. Код пакета `MP3_ID` лаконичен благодаря тому, что в нем нет зависимостей и его можно использовать сразу после установки.

В первой строке `require`-функция загружает модуль:

```
require_once 'MP3/Id.php';
```

Объект класса пока не создается, потому что в этом нет необходимости.

Сначала нужно просмотреть каталог MP3-файлов и создать ассоциативный массив, в котором будет храниться время доступа к файлам:

```
static $MY_MP3_DIR = "/home/ed/mp3"; # Измените это значение!
$objDir = dir($MY_MP3_DIR);
$numFiles = 0;
$arFileTimeHash = Array();
# Пройти по всем найденным файлам в списке
while (false !== ($strEntry = $objDir->read())) {
    # Проверить, является ли текущий файл MP3-файлом,
    # а не каталогом или файлом другого типа!
    if (eregi("\.mp3$", $strEntry)) {
        $arFileTimeHash[$strEntry] = filetime($MY_MP3_DIR . "/" . $strEntry);
        $numFiles++;
    }
};
```

Полезный `Dir`-объект позволяет просмотреть необходимый каталог. К сожалению, он не дает возможности фильтровать вывод. При создании листинга каталога существует вероятность, что в результате будет выведено большое количество ненужной информации.

Чтобы обойти эту проблему, можно пропускать все возвращаемые `Dir`-объектом результаты через регулярное выражение. Описание регулярных выражений — большая тема, которая вкратце рассматривалась в главе 5. Применяемое здесь регулярное выражение, `\.mp3$`, передается нечувствительному к регистру символов методу сравнения, `eregi`, который проверяет, заканчивается ли запись строкой `.mp3`. Если это так, то, скорее всего, данный файл является MP3-файлом и его следует включить в ассоциативный массив.

Ключами ассоциативного массива являются имена файлов (без путей), а значениями — время последнего доступа к соответствующим файлам, которое можно получить, используя функцию `filetime`. Значение переменной `$numFiles`, которое будет использоваться позднее, увеличивается на единицу после нахождения каждого файла.

Не нарушая связь значений и ключей ассоциативного массива, можно заставить PHP-функцию `arsort` отсортировать его по значениям в обратном порядке:

```
# Отсортировать список файлов по дате последнего доступа и
# поместить список в обычный массив для последующего отображения
arsort($arFileTimeHash);
```

В этом примере ключом элемента массива является имя файла, а значением — время доступа, т.е. после сортировки массива файлы, к которым недавно осуществлялся доступ, окажутся первыми.

После этого отсортированный массив преобразуется в обычный индексированный массив, каждый элемент которого является ассоциативным массивом с двумя ключами: `FILENAME` (имя файла) и `ACCESSED` (время доступа к файлу):

```
$intThisArrayIndex = 0;
foreach ($arFileTimeHash as $strFilename => $intAccessTime)
```

```
{
    $arFileList[$intThisArrayIndex]["FILENAME"] = $strFilename;
    $arFileList[$intThisArrayIndex]["ACCESSED"] = $intAccessTime;
    $intThisArrayIndex++;
};
```

Этот этап не обязателен — он лишь проясняет логику, которая необходима для отображения перечня файлов.

Наконец, добавляется простое правило для ограничения вывода — если в массиве больше пяти MP3-файлов, то в списке показываются только первые пять из них, а если меньше, то показываются все файлы:

```
# Если найдено более 5 MP3-файлов, показать только первые 5
if ($intNumFiles > 5) {
    $intNumFiles = 5;
};
```

Теперь можно начинать формирование HTML-кода. Для создания таблицы, в которой отображаются результаты, используется цикл по массиву от 0 (первый MP3-файл) до значения переменной `$intNumFiles` (определенное ранее количество файлов). Естественно, так как отсчет начинается с нуля, из общего количества файлов необходимо вычесть единицу.

В каждой итерации цикла используется объект класса `MP3_ID` — его методу `read` передается имя и путь к файлу, который необходимо обработать, а метод `getTag` позволяет извлечь теги “artist” и “name”:

```
<?php
    $objMP3ID = new MP3_Id();
    for ($i = 0; $i<=($intNumFiles)-1; $i++)
    {
?>
        <tr>
<?php
        $strThisFile = $arFileList[$i]["FILENAME"];
        $strPath = $strMyMP3Directory . "/" . $strThisFile;
        $intResult = $objMP3ID->read ($strPath);
        $strArtist = $objMP3ID->getTag ("artists", "Unknown Artist");
        $strName = $objMP3ID->getTag ("name", "Unknown Track");
        $intAccessTime = $arFileList[$i]["ACCESSED"];
?>
        <td><?=$i+1?></td>
        <td><?=$strArtist?></td>
        <td><?=$strName?></td></st>
        <td><?=date("m/d/Y H:i", $intAccessTime)?></td>
    </tr>
<?php
    };
?>
```

Следует отметить, что здесь также выводится время доступа к файлу. Эта информация извлекается не из IDv3-тега файла, а из самого массива с перечнем файлов; информация извлекается и записывается фактически во время создания листинга каталога, поэтому к ней можно запросто вернуться.

Время доступа, которое представлено в виде временной метки Unix (количество истекших секунд с первого января 1970 года), с помощью PHP-функции `date` отображается в более читабельном формате.

Итак, в несколько простых шагов была создана оригинальная реклама для начальной страницы (которая будет гораздо интереснее, чем ссылка вроде “Заполните мою гостевую книгу”).

Что делать, если найдена проблема?

При использовании этого интересного приложения возможно появление небольшой проблемы. Если запустить сценарий один раз, то все будет работать нормально, но если запустить его дважды (нажать кнопку “Обновить” в Web-браузере), то колонка “время последнего прослушивания” будет выглядеть необычно.

Это не ошибка, а скорее случайная функция. Чтобы считать ID-тег MP3-файла, фактически необходимо осуществить доступ к этому файлу, в результате чего изменится время последнего доступа к нему. Поэтому при повторном запуске приложения время доступа к MP3-файлу равно времени запуска сценария, а не времени последнего прослушивания файла.

Есть простой способ обойти эту проблему — использование PHP-функции touch. Сразу после доступа к файлу и получения IDv3-сведений следует, используя функцию touch, установить время доступа к файлу, равное его предыдущему значению. К сожалению, данный метод работает, только если владельцем MP3-файлов является тот же пользователь, от имени которого работает процесс Web-сервера.

Более сложное решение предполагает периодическое индексирование MP3-файлов в базе данных, чтобы не нужно было считывать IDv3-теги каждый раз, когда пользователь посещает Web-страницу; затем можно обращаться к базе данных, оставляя время доступа к файлу нетронутым. Реализация этого способа — хорошее упражнение для читателей.

Время доступа к файлу — свойство, которое операционная система предоставляет PHP. В зависимости от используемой операционной системы и от файловой системы сервера это свойство может быть недоступно в PHP. В таком случае вместо времени доступа можно использовать время последнего изменения файла. Это означает, что данные, возвращаемые сценарием, не будут абсолютно точными. Поэтому если планируется развернуть такое приложение в реальной среде, прежде чем предоставлять к нему общий доступ, желательно обеспечить проверку возвращаемых результатов.

Создание приложения с использованием двух PEAR-компонентов

Объединим полученные навыки работы с PEAR и создадим новое приложение с нуля. Ниже рассматривается пример создания полезного приложения с использованием двух уже описанных PEAR-компонентов — HTML_TreeMenu и MP3_ID.

Приложение

В наши дни некоторые из самых передовых Internet-радиостанций, а также обычные радиостанции, имеющие свои Web-сайты, предлагают своим слушателям возможность заказывать на этих сайтах любимые песни.

Использование ограниченного списка композиций с запросами в свободной форме было бы неразумным, поскольку многие запросы слушателей оставались бы невыполненными. Обычно станции предлагают списки всех музыкальных композиций, находящихся в их архивах, и позволяют слушателям выбирать из этих списков интересные их песни.

Постоянные поступления в архив, которые обычно состоят из MP3-файлов, могут сильно усложнять поддержку запросной части Web-сайта, поэтому желательно иметь способ динамической генерации и удобного форматирования списка запросов.

Разрабатываемое приложение обладает указанными функциями. Оно предоставляет все MP3-файлы в каталоге сервера в виде иерархического списка (используя

компонент HTML_TreeMenu), отсортированного по имени исполнителя. Пользователь заказывает песню, щелкнув на ее названии, затем приложение генерирует и отправляет диджею радиостанции e-mail-письмо, в котором указан соответствующий файл.

Для отправки почты в приложении используется встроенная PHP-функция mail(). Это очень простой способ отправки e-mail и в большинстве реальных приложений требуется нечто более сложное. В следующей главе описывается поиск PEAR-классов, которые могут выручить разработчика.

Для создания вывода компонента HTML_TreeMenu также используется XML. Чтобы преобразовать XML-код в PHP-методы, понятные HTML_TreeMenu, также придется использовать вспомогательный класс.

Архитектура

С целью аккуратной организации компонентов приложения его можно разделить на два отдельных PHP-файла. Первый файл radiogeneratexml.php генерирует XML-представление каталога MP3-файлов на сервере, отсортированное по имени исполнителя. Второй сценарий radiorequest.php фактически отображает этот список и обрабатывает переданные пользователем данные.

XML-код передается от одного сценария другому посредством HTTP-запросов. Сценарий будет заставлять сервер делать запросы от своего имени. Для тех, кто знаком с идеей Web-служб на базе XML, такой подход не покажется бессмысленным. В данном случае идея заключается в том, чтобы реализовать возможность получать XML-данные практически из любого источника. Например, реальная радиостанция может использовать один сервер для обслуживания общедоступного Web-сайта и полностью обособленный сервер для хранения MP3-файлов. Второй сервер может быть недоступен из внешнего мира. Вместо того чтобы пытаться считывать информацию о файлах через какие-либо сетевые ресурсы, что весьма неэффективно, можно поместить файл radiogeneratexml.php на сервер с MP3-файлами, а сценарий radiorequest.php — на общедоступный Web-сервер, и заставить последний запрашивать XML-код с другого сервера.

Кроме всего прочего, это позволит в случае необходимости сделать XML-код общедоступным и, таким образом, предоставить Web-сайтам третьих лиц доступ к данным.

XML-код в целях разрабатываемого приложения соответствует стандартам, которых требует компонент HTML_TreeMenu. Если учесть возможность вторичного использования XML, то впоследствии можно обновить XML-код так, чтобы приложение было более ориентированным на службу, а не на формат отображения. Это позволит использовать простую таблицу стилей XSL для преобразования служебного XML-кода в XML-код, понятный PEAR-компоненту.

Генерация XML-кода

Сценарий radiogeneratexml.php отвечает не только за создание XML-кода, который может использоваться страницей заказов, но и за чтение MP3-каталога, извлечение полезных данных об исполнителе и названии песни из каждого MP3-файла, а также за необходимую группировку и сортировку списка композиций.

Рассмотрим кратко XML-код, который нужно генерировать для компонента HTML_TreeMenu:

```
<?xml version="1.0"?>
<treemenu>
  <node text="Extreme Metal Grinding" icon="folder.gif">
    <node text="Extreme Noises" icon="document.gif"
link="radiorequest.php?requestfile=9991885931034.mp3" />
  </node>
</treemenu>
```

```

<node text="Extreme Temper Loss" icon="document.gif"
link="radiorequest.php?requestfile=9991885931035.mp3" />
</node>
<node text="Massive Pitch Correction" icon="folder.gif">
<node text="How long ago" icon="document.gif"
link="radiorequest.php?requestfile=9991885931036.mp3" />
<node text="Ten minutes to Sunrise" icon="document.gif"
link="radiorequest.php?requestfile=9991885931037.mp3" />
</node>
</treemenu>

```

В данном случае в коллекции MP3-записей имеется два исполнителя: “Extreme Metal Grinding” и “Massive Pitch Correction” (это не самая большая радиостанция в мире). Названия песен обоих исполнителей размещаются в соответствующих узлах иерархии.

Название каждой песни представляет собой ссылку на страницу `radiorequest.php`; в ссылке передается GET-параметр `requestFile`. Имя MP3-файла передается потому, что диджею удобнее использовать имя файла, а не IDv3-тег. PHP-функция `urlencode()` гарантирует, что имя файла перед передачей будет преобразовано в необходимую для URL форму.

Чтобы проверить, как страница выглядит после обработки `HTML_TreeMenu`, можно использовать код из примера в первом разделе “Практика” этой главы и вставить XML-код в файл `treemenutest2.xml`. Затем необходимо модифицировать файл `treemenutest.php`, указав в нем корректное имя XML-файла:

```
$objXMLTree = new XMLHTMLTree("treemenutest2.xml");
```

Запустите Web-браузер и снова откройте в нем страницу `treemenutest.php`. Результат работы сценария показан на рис. 14.3.

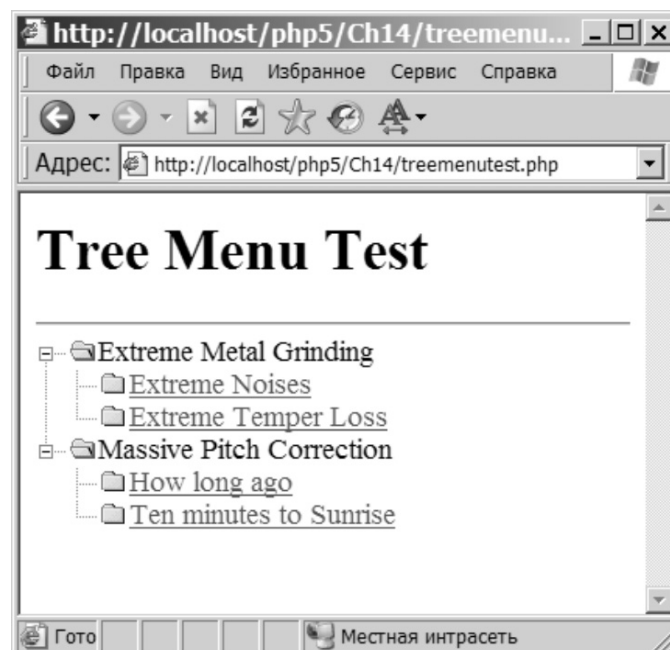


Рис. 14.3.

Теперь страница заказов выглядит так, как нужно (хотя HTML-код можно немного исправить). Необходимо только определить, каким образом подобный код будет генерироваться динамически на основании содержимого каталога MP3-файлов.

Практика Создание двухуровневого, двухкомпонентного PEAR-приложения

Работа кода для обоих компонентов подробнее описывается после листингов.

Сценарий radiogeneratexml.php

Создайте новый файл, вставьте в него следующий код и сохраните его под именем radiogeneratexml.php. Именно этот уровень автоматически генерирует XML-код для дерева навигации на основании содержимого каталога MP3-файлов. Поместите файл в каталог, доступный Web-серверу.

```
<?php
header("Content-Type: text/xml\n\n");

require_once 'MP3/Id.php';

static $strMyMP3Directory = "/home/ed/mp3";
$objDir = dir($strMyMP3Directory);
$intNumFiles = 0;
$arMP3Files = Array();

// Пройти в цикле по всем файлам и поместить их имена в массив
while (false != ($strEntry = $objDir->read())) {
    // Проверить, является ли этот файл MP3-файлом
    if (ereg("\.mp3$", $strEntry)) {
        $arMP3Files[] = $strMyMP3Directory . "/" . $strEntry;
    }
};

// Создать объект класса MP3_ID
$objMP3ID = new MP3_Id();

// Создать массив уникальных имен исполнителей
$arArtists = Array();
$arTestArtists = Array();
for ($i=0; $i<=sizeof($arMP3Files)-1; $i++) {
    $strPath = $arMP3Files[$i];
    $intResult = $objMP3ID->read($strPath);
    $strArtist = $objMP3ID->getTag("artists", "Unknown Artist");
    $strTestArtist = strtoupper(preg_replace("/[{}^{}A-Za-z0-9]/",
"", $strArtist));
    // Проверить, находится ли имя данного исполнителя (после того
    // как необычные символы удалены и все буквы
    // переведены в верхний регистр) в массиве - если нет, то
    // добавить исполнителя в массив
    if (in_array($strTestArtist, $arTestArtists) == false) {
        array_push($arArtists, $strArtist);
        // Обратите внимание, для добавления имен в массив $arArtists
        // используются исходное количество пробелов и регистр символов...
        array_push($arTestArtists, $strTestArtist);
    }
};

// Для каждого исполнителя создается массив, содержащий все
// имена файлов его песен, а также названия песен
$arTracks = Array();
// Кроме того, чтобы предотвратить повторное считывание информации
// о песнях, создается массив всех уже вычисленных индексов песен
$arAlreadyAccountedForSongIndices = Array();
for ($i=0; $i<=sizeof($arArtists)-1; $i++) {
```

```

    $strArtistName = $arArtists[$i];
    $strTestArtistName = $arTestArtists[$i];
    $arTracks[$strArtistName] = Array();
    // Определяем, какие песни написаны данным исполнителем
    for ($j=0; $j<=sizeof($arMP3Files)-1; $j++) {
        if (in_array($j, $arAlreadyAccountedForSongIndices) == false) {
            $strPath = $arMP3Files[$j];
            $intResult = $objMP3ID->read ($strPath);
            $strThisArtist = $objMP3ID->getTag ("artists", "Unknown Artist");
            $strThisTestArtist = strtolower(preg_replace("/[^A-Za-z0-9]/",
                "", $strThisArtist));
            if ($strThisTestArtist == $strTestArtistName) {
                // Эта песня действительно принадлежит проверяемому
                // исполнителю, поэтому вводим ее индекс в массив
                // $arAlreadyAccountedForSongIndices, чтобы не
                // проверять ее снова
                array_push($arAlreadyAccountedForSongIndices, $j);
                // Получаем название песни и ссылку для ее заказа и
                // помещаем их во временный массив
                $arSongHash["TITLE"] = $objMP3ID->getTag ("name", "Unknown Title");
                $strSongFilename = str_replace("$strMyMP3Directory" . "/", "",
                    $arMP3Files[$j]);
                // Создаем ссылку по имени файла, обработав имя файла
                // с помощью функции urlencode
                $arSongHash["LINK"] = "radiorequest.php?requestfile=" .
                    urlencode($strSongFilename);

                // Помещаем в массив следующий доступный индекс
                // $arTracks[имя исполнителя]
                array_push($arTracks[$strArtistName], $arSongHash);
            }
        }
    }
    // Сортируем массив по именам исполнителей по
    //возрастанию (от А до Z), используя функцию ksort
    ksort($arTracks);

    // Выводим подходящий XML-код
    ?>
    <treemenu>
    <?php foreach ($arTracks as $artist_name => $arHash) { ?>
        <node text="<?=$artist_name?>" icon="folder.gif">
    <?php for ($i=0; $i<=sizeof($arHash)-1; $i++) { ?>
        <node text="<?=htmlentities($arHash[$i] ["TITLE"])?>" icon="document.gif"
            link="<?=htmlentities($arHash[$i] ["LINK"])?>" />

    <?php }; ?>
    </node>
    <?php }; ?>
    </treemenu>

```

Сценарий radiorequest.php

Создайте новый файл и введите в него следующий код. Сохраните файл с именем `radiorequest.php` и поместите его в тот же каталог сервера, в котором находится файл `radiogeneratexml.php`. Сценарий отображает дерево навигации по доступным для заказа MP3-записям, сгенерированное сценарием `radiogeneratexml.php`, и обрабатывает сделанные пользователями заказы.

```

<?php
    static $strMyMP3Directory = "/home/ed/mp3";
    static $strMyDJsEmailAddress = "ed@example.com";

```

```

// Подключаем необходимые PEAR-объекты
require_once('HTML/TreeMenu.php');
require_once('xmlhtmltree.phpm');

// Определяем URL для получения XML и получаем его. Этот
// фрагмент можно модифицировать, если XML-генератор
// расположен на другом сервере.
$strXMLURL = "http://" . $_SERVER["SERVER_NAME"] . str_replace("request",
    "generatexml", $_SERVER["SCRIPT_NAME"]);

$strXML = implode (false, file($strXMLURL));

$objXMLTree = new XMLHTMLTree("", $strXML);
$objXMLTree->GenerateHandOffs();
$objXMLTree->ParseXML();
$objTreeMenu = $objXMLTree->GetTreeHandoff();

// Проверим, сделан ли запрос - если да, то следует слегка
// изменить вывод

$requestMade = false;
$requestSuccessful = false;

if (!empty($_GET['requestfile'])) {
    $requestMade = true;
    // Получаем имя файла
    $strRequestFilename = $_GET['requestfile'];
    // Проверяем существование файла
    $strFullPath = $strMyMP3Directory . "/" . $strRequestFilename;
}
if (@filesize($strFullPath) > 0) {
    $requestSuccessful = true;
    // Все сработало, отправляем диджею e-mail
    mail($strMyDJsEmailAddress, "Заказ новой песни", "Сделан заказ песни: " .
        $strFullPath);
} else {
    $requestSuccessful = false;
}

?>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript" SRC="TreeMenu.js"></SCRIPT>
</HEAD>
<BODY>
<H1>Радио PHP</H1>
<?php
    if ($requestMade) {
?>
<B>Спасибо за заказ!</B>
<BR><BR>
<?php
    if ($requestSuccessful) {
?>
        Ваш заказ принят, мы постараемся как можно
        быстрее поставить заказанную вами песню.

    <?php
    } else {
?>
        К сожалению, мы не можем поставить заказанную
        вами песню. Возможно, она была недавно удалена из
        нашей коллекции. Пожалуйста, попробуйте заказать
        другую песню.
    <?php

```

```

    };
    ?>
    <BR><BR>
    <?php
    };
    ?>
    <?php
    if (!($requestSuccessful)) {
    ?>

    Закажите песню из приведенного ниже списка. Заказ будет отправлен
    нашему диджею. Если она давно не проигрывалась, то мы включим ее
    как можно быстрее!
    <HR>
    <?php
    $objTreeMenu->printMenu();
    ?>
    <?php
    };
    ?>
    </BODY>
</HTML>

```

Откройте в браузере страницу `radiorequest.php`. Должна появиться страница, похожая на рис. 14.4.

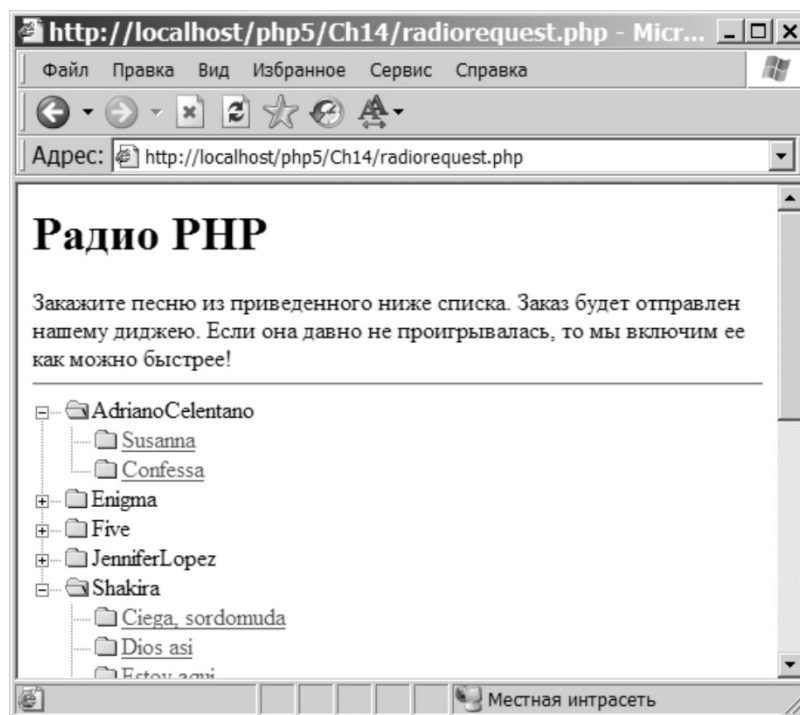


Рис. 14.4.

Разверните один из узлов и щелкните на какой-либо песне. В результате этого должно появиться сообщение, показанное на рис. 14.5 (предполагается, что параметр `sendmail_path` в файле `php.ini` задан правильно).

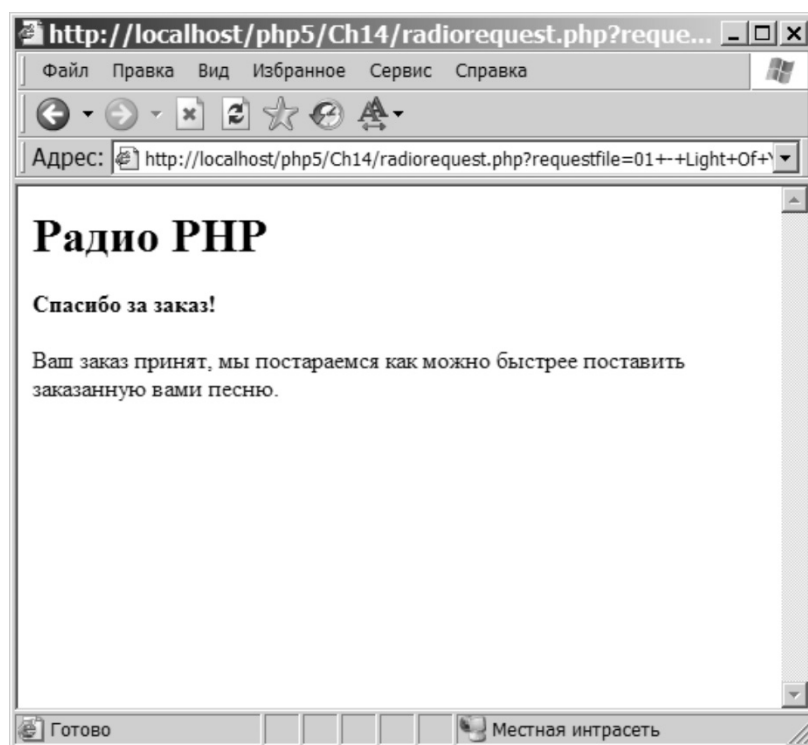


Рис. 14.5.

Как это работает: radiogeneratexml.php

Рассмотрим работу сценария `radiogeneratexml.php` шаг за шагом.

Строго говоря, клиенту отправляется XML-код, а не HTML, поэтому следует сообщить об этом клиенту (встроенный в РНР объект запросов не делает этого), используя правильно сформированный HTTP-заголовок Content Type:

```
header("Content-Type: text/xml\n\n");
```

Очень важно, чтобы перед этим заголовком в коде не было пустот вне разделителей `<?php` и `?>`, потому что, встретив любой не-РНР-код — даже пробел или перевод строки, — РНР отправляет заголовок типа содержимого `text/html`, а это в данном случае нежелательно.

Далее следует подключить PEAR-класс (MP3_ID):

```
require_once 'MP3/Id.php';
```

В этом файле выводится обычный XML-код, поэтому нет необходимости подключать класс `HTML_TreeMenu`.

Рассмотрим создание перечня MP3-файлов в каталоге. Измените значение соответствующей переменной, чтобы указать каталог, в котором находятся MP3-файлы (для работы этого примера в каталоге должно быть хотя бы три MP3-файла). Затем с помощью модифицированной версии кода из предыдущего MP3_ID-проекта создается одномерный массив всех MP3-файлов (упорядоченных по имени) в этом каталоге:


```
static $strMyMP3Directory = "/home/ed/mp3"; # Укажите реальный каталог с MP3-файлами
$objDir = dir($strMyMP3Directory);
$intNumFiles = 0;
$arMP3Files = Array();

while (false != ($strEntry = $objDir->read())) {
    if (eregi("\.mp3$", $strEntry)) {
        $arMP3Files[] = $strMyMP3Directory . "/" . $strEntry;
    };
};
```

Теперь можно использовать линейную коллекцию MP3-файлов. Так как физическая структура создаваемого приложения в своей основе упорядочена по именам исполнителей, имеет смысл создать один массив с именами всех исполнителей в коллекции MP3-файлов. Возможно, коллекция состоит из тысяч файлов, но, скорее всего, отдельных исполнителей в ней будет только несколько сотен.

Получение имени исполнителя из каждого файла — первая цель использования класса MP3_ID:

```
$objMP3ID = new MP3_Id();
$arArtists = Array();
$arTestArtists = Array();
for ($i=0; $i<=sizeof($arMP3Files)-1; $i++) {
    $strPath = $arMP3Files[$i];
    $intResult = $objMP3ID->read ($strPath);
    $strArtist = $objMP3ID->getTag ("artists", "Unknown Artist");
    $strTestArtist = strtoupper(preg_replace("/[^A-Za-z0-9]/", "", $strArtist));
    if (in_array($strTestArtist, $arTestArtists) == false) {
        array_push($arArtists, $strArtist);
        array_push($arTestArtists, $strTestArtist);
    };
};
```

Следует отметить, что создается два массива — \$arArtists, содержащий имена исполнителей, и \$arTestArtists с измененной версией каждого имени, где все буквы переведены в верхний регистр и удалены все не алфавитно-цифровые символы (включая пробелы). Это сделано по одной причине: необходимо убедиться, что каждый исполнитель появляется в списке только один раз. Удаление пробелов и знаков препинания, а также перевод всех символов в верхний регистр позволяет с большой долей уверенности сравнить имя исполнителя, извлеченное из IDv3-тега, с тем именем, которое уже встречалось в течение времени жизни цикла.

В качестве примера рассмотрим вымышленного исполнителя “Extreme Metal Grinding”. Предположим, что в коллекции имеется три композиции этого исполнителя, созданных в разное время тремя различными программами. Имя исполнителя в IDv3-теге вполне может быть представлено тремя незначительно отличающимися способами, например, “Extreme Metal Grinding”, “EXTREME ‘Metal’ Grinding” и “Extreme Metalgrinding”. Используемое в операторе \$strTestArtist = strtoupper (preg_replace (“/[^A-Za-z0-9]/”, “”, \$strArtist)) регулярное выражение и преобразование к верхнему регистру для каждого из этих тегов позволяет всегда получать один и тот же результат: EXTREMOMETALGRINDING. Вероятно, на экране такое имя будет выглядеть некрасиво, но оно хорошо подходит для сравнения и создания учетной записи для этой группы исполнителей.

Итак, в цикле просматривается каждый MP3-файл и из него извлекается имя исполнителя, к которому применяется регулярное выражение. Затем нужно проверить, присутствует ли измененное имя исполнителя в тестовом массиве. Если это не так, то имя исполнителя записывается в оба массива — в тестовый массив (сюда записывается

измененное имя) и в реальный массив имен исполнителей (записываются исходные имена). Если имя записано в тестовом массиве, то нет необходимости добавлять его снова.

Теперь следует поместить каждую песню в один массив. Необходимо создать массив, упорядоченный по уникальным именам исполнителей, поэтому есть смысл создать ассоциативный массив, в котором ключами будут имена исполнителей. Так как каждому исполнителю, скорее всего, принадлежит несколько песен, значением каждого элемента массива будет другой одномерный массив, в котором каждый элемент будет представлять одну песню. Поскольку желательно хранить как фактическое название каждой песни, так и имя соответствующего файла, следует представить каждый элемент этого массива тоже в виде массива. Получающаяся в результате структура данных весьма сложна (при желании ее можно просмотреть с помощью функции `var_dump()`), но она идеально подходит для создания XML-кода и действительно очень упрощает код сценария `radiorequest.php`.

Имена всех исполнителей в только что созданном массиве обрабатываются с помощью следующего цикла:

```
$arTracks = Array();
$arAlreadyAccountedForSongIndices = Array();
for ($i=0; $i<=sizeof($arArtists)-1; $i++) {
    $strArtistName = $arArtists[$i];
    $strTestArtistName = $arTestArtists[$i];
    $arTracks[$strArtistName] = Array();
    for ($j=0; $j<=sizeof($arMP3Files)-1; $j++) {
        if (in_array($j, $arAlreadyAccountedForSongIndices) == false) {
            $strPath = $arMP3Files[$j];
            $intResult = $objMP3ID->read ($strPath);
            $strThisArtist = $objMP3ID->getTag ("artists", "Unknown Artist");
            $strThisTestArtist = strtoupper(preg_replace("/[^A-Za-z0-9]/", "",
                $strThisArtist));
            if ($strThisTestArtist == $strTestArtistName) {
                array_push($arAlreadyAccountedForSongIndices, $j);
                $arSongHash["TITLE"] = $objMP3ID->getTag ("name", "Unknown Title");
                $strSongFilename = str_replace("$strMyMP3Directory" . "/" . "",
                    $arMP3Files[$j]);
                $arSongHash["LINK"] = "radiorequest.php?requestfile=" .
                    urlencode($strSongFilename);
                array_push($arTracks[$strArtistName], $arSongHash);
            }
        }
    }
};
```

По сути, в цикле обрабатывается список исполнителей (реальный список, а не тестовый), а в массиве создается ключ на основе имен исполнителей.

Затем для каждого ключа просматривается весь список MP3-файлов и проверяется, принадлежит ли каждая композиция данному исполнителю. Имя исполнителя, хранящееся под соответствующим индексом в тестовом массиве, сверяется с именем исполнителя в MP3-файле, которое предварительно обрабатывается регулярным выражением. Это, как уже было сказано, позволяет игнорировать различия в количестве пробелов, знаков препинания и регистре символов.

Чтобы определить, какие песни принадлежат тому или иному исполнителю, всю коллекцию приходится просматривать в цикле. Вследствие этого рассматриваемый сценарий может потреблять значительные ресурсы и работать медленно. Поэтому для повышения производительности необходимо связать каждый индекс массива MP3-файлов (`$arMP3Files`) с исполнителем только один раз — композиция, для которой

создан MP3-файл, может быть написана только одним исполнителем, по крайней мере, если рассматривать IDv3-теги. Выполняется проверка по искомому индексу, и если этот индекс уже связан с исполнителем, то нет смысла тратить на него время. Как только принадлежность заданного MP3-файла определена (индекс *j* в исходном массиве *\$arMP3Files*), этот индекс добавляется в другой массив *\$arAlreadyAccountedForSongIndices*. В результате можно не проверять этот MP3-файл при последующих итерациях, если его принадлежность данному исполнителю уже установлена. Это значительно повышает производительность, поскольку ограничивается количество необходимых операций чтения IDv3-тегов.

Если имя исполнителя проверяемой записи совпадает с именем исполнителя, чье авторство проверяется в текущий момент, то в массив с индексом текущего исполнителя добавляется соответствующий массив, представляющий эту запись. Добавляемый массив содержит название песни и ссылку (содержащую имя файла без пути) на страницу *radiorequest.php*. Идея заключается в том, что когда пользователь щелкает по этой ссылке, он фактически подтверждает, что хочет прослушать эту песню. Чтобы гарантировать, что создаваемый URL будет корректно обрабатываться Web-браузером, используется функция *urlencode*.

Наконец, для сортировки массива по ключам в алфавитном порядке используется функция *ksort*. Почти так же как и функция *arsort*, которая использовалась в предыдущем MP3_ID-приложении, *ksort* сохраняет в ассоциативном массиве внутреннюю связь между каждым ключом и его значением:

```
ksort($arTracks);
```

Используя созданную структуру данных, можно выводить XML-код.

Так как генерируемый XML-код крайне прост, преимущества встроенной PHP5-библиотеки *libxml* для поддержки XML не используются. Вместо этого XML-вывод создается так же, как если бы это был HTML-код. Однако вполне можно переписать соответствующую подпрограмму, так чтобы она использовала *libxml2* (см. главу 8).

Весьма простая рекурсия позволяет обработать структуру данных и отправить вывод непосредственно в браузер. Обоснование для использования в предыдущем блоке кода внешне громоздкой структуры данных теперь должно быть очевидным — можно конвертировать структуры данных в XML, используя всего несколько строк кода. В результате программист немного усложняет себе работу, но вместе с тем упрощает генерирование XML-кода. Это пригодится, если впоследствии придется по какой-либо причине подправить XML-код — это можно будет легко сделать, не затрагивая многочисленных строк сложной логики.

Следующий фрагмент выводит XML-код в браузер:

```
<treemenu>
<? foreach ($arTracks as $artist_name => $arHash) { ?>
  <node text="<?=$artist_name?>" icon="folder.gif">
<? for ($i=0; $i<=sizeof($arHash)-1; $i++) { ?>
  <node text="<?=htmlentities($arHash[$i]["TITLE"])?>" icon="document.gif"
    link="<?=htmlentities($arHash[$i]["LINK"])?>" />
<? }; ?>
  </node>
<? }; ?>
</treemenu>
```

При генерации обоих значений (названия записи и ссылки) используется функция *htmlentities*, гарантирующая, что вывод будет полностью соответствовать XML-стандарту. Это может в буквальном смысле исказить URL, но синтаксический анализатор компонента *HTMLTree_Menu* исправит это неудобство.

Теперь можно открыть страницу `radiogeneratexml.php` в Web-браузере. Чтобы заставить Internet Explorer отображать страницу как XML (что делает ее более удобочитаемой и позволит разворачивать и сворачивать отдельные узлы), можно использовать следующий трюк с URL:

`http://your_server_name/radiogeneratexml.php/.xml`

Web-сервер игнорирует лишние символы `/.xml` в конце, но Internet Explorer “полагает”, что пользователь определенно запрашивает XML-страницу и поэтому отображает ее как следует. (Непонятно, почему этот браузер не учитывает MIME-заголовок, для создания которого пришлось приложить определенные усилия в начале сценария.)

Чтобы полностью убедиться, что все работает, можно скопировать XML-код, вставить его в файл `treemenutest.xml`, а затем открыть страницу `treemenutest.php` снова. Чтобы сделать это, просто откройте в браузере предыдущий URL. В окне должен появиться XML-код, в котором можно найти MP3-файлы. Если все выглядит нормально, то можно считать, что большая часть работы сделана.

Как это работает: сценарий `radiorequest.php`

Файл `radiorequest.php` до сих пор был самым простым из двух файлов. Работа этого сценария будет понятна, если вы знаете, как работает `radiogeneratexml.php`.

Сценарий начинается с определения некоторых переменных:

```
<?php
static $strMyMP3Directory = "/home/ed/mp3"; # Укажите реальный каталог MP3-файлов
static $strMyDJsEmailAddress = "ed@example.com";
?>
```

Очевидно, следует заменить e-mail-адрес и вставить путь к каталогу MP3-файлов на Web-сервере, как это было сделано в сценарии генерирования XML-кода.

Загрузить MP3-файлы и поэкспериментировать с ними можно, посетив сайт www.iuta.com. Для редактирования IDv3-тегов можно использовать Winamp (www.winamp.com) — MP3-проигрыватель, позволяющий также редактировать IDv3-теги.

Затем подключаются PEAR-объекты. Здесь нет необходимости использовать `MP3_ID` — только `HTML_TreeMenu` и вспомогательный компонент:

```
require_once('HTML/TreeMenu.php');
require_once('xmlhtmltree.phpm');
```

Для создания URL, указывающего на XML-генератор, используется несложный трюк. В рассматриваемом примере можно предположить, что сценарий расположен на собственном сервере в том же каталоге, что и сценарий генератор XML-кода. URL сценария заказа известен, потому что он вызывается пользователем. Корректный URL для сценария-генератора идентичен данному URL, кроме слова `request`, которое заменено `generatexml`. Для этого в сценарии используется поиск и замена, в результате которой формируется URL сценария-генератора.

Как только URL создан, содержимое соответствующей страницы извлекается так же, как содержимое файла локальной файловой системы — PHP достаточно развит, чтобы выполнить такой запрос. Затем полученный XML-вывод передается `HTML_TreeMenu`-объекту так же, как в начале главы. Ниже приведен соответствующий фрагмент кода:

```
$strXMLURL = "http://" . $SERVER["SERVER_NAME"] . str_replace("request",
    "generatexml", $_SERVER["SCRIPT_NAME"]);
```

```

$strXML = implode ('', file($strXMLURL));

$objXMLTree = new XMLHTMLTree("", $strXML);
$objXMLTree->GenerateHandOffs();
$objXMLTree->ParseXML();
$objTreeMenu = $objXMLTree->GetTreeHandoff();

```

Затем подключается условная логика, определяющая поведение сценария при получении песни, заказанной пользователем. В XML-выводе генерируются ссылки для дерева навигации, которые выглядят следующим образом:

```
/radiorequest.php?requestfile=9991885931037.mp3
```

Сценарий `radiorequest.php` фактически выполняет две функции: он отображает дерево MP3-файлов, а также сообщение о том, что заказ выполнен успешно. В этом РНР-сценарии необходимо принимать во внимание запросы, в которые включен параметр `requestfile`, определяющий имя запрашиваемого MP3-файла. Желательно не только знать, что заказ был сделан, но и убедиться в том, что заказ был достоверным. Для этого применяется проверка существования файла:

```

$requestMade = false;
$requestSuccessful = false;
if (!empty($_GET['requestfile'])) {
    $requestMade = true;
    $strRequestFilename = $_GET["requestfile"];
    $strFullPath = $strMyMP3Directory . "/" . $strRequestFilename;
    if (@filesize($strFullPath) > 0) {

```

Следует отметить, что проверка существования файла выполняется путем определения полного пути к нему, после чего определяется его размер с помощью функции `filesize`. Размер, превышающий единицу, указывает на то, что файл действительно существует. Оператор `@` используется потому, что попытка вызывать РНР-функцию `filesize` для несуществующего файла приводит к генерации неисправимой ошибки, в результате которой выполнение сценария прекращается, а это абсолютно нежелательно.

Если файл существует, то заказ песни считается успешным и сценарий отправляет диджею e-mail-сообщение, используя определенную в начале сценария переменную `$strMyDJsEmailAddress`, в которой записан необходимый адрес:

```

        $requestSuccessful = true;
        // Все работает. Можно отправлять письмо диджею.
        mail($strMyDJsEmailAddress, "Новый заказ", "Заказана песня. Файл: " .
            $strFullPath);
    } else {
        $requestSuccessful = false;
    };
};

```

Функция `mail()` позволяет определить необходимый e-mail-адрес, тему и тело сообщения, а также другие параметры, которые подробно описаны в главе 15.

Теперь можно переходить к логике отображения, т.е. к блоку сценария, генерирующему HTML-код, который фактически отправляется браузеру пользователя.

Комбинация переменных `$requestMade` и `$requestSuccessful` используется для определения того, какую информацию отображать. Если заказа не было, то сценарий приглашает пользователя сделать заказ и выводит меню, созданное `HTML_TreeMenu`-компонентом. Если заказ сделан, но выполнить его не удастся, выводится поясняющее сообщение и приглашение сделать другой заказ; при этом также выводится `HTML_TreeMenu`-меню. Если заказ сделан и его можно выполнить, то выводится сообщение с благодарностью, а меню не выводится, потому что в целях данного упражнения предполагается, что пользователь не будет заказывать вторую песню.

На практике, чтобы ограничить количество заказов от одного пользователя за один день, можно использовать какую-либо специальную методику, например, cookie-файлы.

Ниже приведен код отображения:

```
<HTML>
<HEAD>
  <SCRIPT LANGUAGE="Javascript" SRC="TreeMenu.js"></SCRIPT>
</HEAD>
<BODY>
  <H1>Радио PHP</H1>
  <?php
    if ($requestMade) {
  ?>
  <B>Спасибо за заказ!</B>
  <BR><BR>
  <?php
    if ($requestSuccessful) {
  ?>
      Ваш заказ принят, мы постараемся как можно
      быстрее поставить заказанную вами песню.

  <?php
    } else {
  ?>
      К сожалению, мы не можем поставить заказанную
      вами песню. Возможно, она была недавно удалена из
      нашей коллекции. Пожалуйста, попробуйте заказать
      другую песню.
  <?php
    };
  ?>
  <BR><BR>
  <?php
    };
  ?>
  <?php
    if (!($requestSuccessful)) {
  ?>

      Закажите песню из приведенного ниже списка. Заказ
      будет отправлен нашему диджею.
      Если песня давно не проигрывалась, то мы включим ее
      как можно быстрее!
  <HR>
  <?php
    $objTreeMenu->printMenu();
  ?>
  <?php
    };
  ?>
</BODY>
</HTML>
```

Меню генерируется в точности так же, как в первом примере этой главы.

Наконец, следует включить упомянутый в начале HTML-кода JavaScript-файл, необходимый для работы HTML_TreeMenu-компонента.

Резюме

Эта глава знакомит читателя с PEAR — репозиторием PHP-расширений и приложений.

В главе рассматривались темы поиска подходящих пакетов на Web-сайте проекта PEAR, а также методика их загрузки и установки; была описана структура проекта PEAR —

знание организации репозитория впоследствии должно облегчить поиск необходимых компонентов.

Читатель познакомился с методикой изучения интерфейсов новых PEAR-компонентов и оценки их пригодности для разрабатываемого проекта, отмечались некоторые характерные особенности PEAR-пакетов, описаны способы их преодоления.

В этой главе вновь рассматривались PEAR-стандарты написания кода, которых придерживались авторы книги при написании примеров, а также логическое обоснование, лежащее в основе этих стандартов.

Наконец, в главе был представлен пример создания функционального, полезного приложения с помощью нескольких PEAR-пакетов.

В следующей главе рассматриваются встроенные в PHP почтовые функции и примеры использования нескольких PEAR-пакетов, предоставляющих некоторые развитые функции обработки и диспетчеризации почты.

RНР5 и электронная почта

Одним из самых простых и наиболее распространенных требований к РНР-приложению является возможность отправлять e-mail-сообщения. Электронная почта за очень короткое время стала одним из самых популярных средств коммуникации и РНР, естественно, предоставляет функции для реализации этого средства в приложениях. Детали работы почты в Internet могут быть весьма сложными, и в основном эта сложность исходит не от РНР-программ, а скорее от различных технологий, платформ и протоколов, которые используются для доставки почты из одной точки в другую. И все же начать использовать РНР и e-mail можно с помощью нескольких простых функций.

Прежде чем рассматривать разработку сценариев для отправки e-mail в Windows и Linux-платформах, следует изучить некоторые технические принципы, лежащие в основе отправки e-mail. Конечно, понимание того, как РНР5 взаимодействует с почтовым сервером, весьма важно, и чтобы читателю было проще понять это, в главе приводится пример создания несложного приложения для отправки почты, в котором используется функция `mail()`. Наконец, в этой главе рассматривается метод создания различных типов содержимого e-mail-писем с использованием формата MIME (Multipurpose Internet Mail Extension — многоцелевое расширение почты в Internet).

Основы e-mail

Чтобы использовать самые распространенные РНР-функции, РНР должен работать в операционной системе (например, Windows или Linux) и иметь доступ к службам Web-сервера. Для использования e-mail-функций РНР также должен иметь доступ к МТА (Mail Transport Agent — почтовый транспортный агент), например, к Sendmail, серверному программному обеспечению, которое отправляет и получает e-mail-сообщения.

Несмотря на то что в РНР существует функция `mail()`, которая отправляет e-mail, в этом языке нет (на момент написания книги) официальных версий каких-либо РНР-функций или расширений, получающих почту. Расширение `mailparse` все еще находится в разработке.

Чаще всего в PHP используется MTA Sendmail; программист может даже не знать, что на используемом им сервере работает Sendmail — в сценарии работает функция `mail()`. В последующем разделе описываются некоторые технические вопросы, связанные с почтовыми протоколами Internet. Однако если читателя главным образом интересует отправка почты и работа почтовой функции, то можно сразу перейти к разделу “Создание простого PHP-приложения для работы с e-mail”.

Почтовые протоколы Internet

Как и в случае любого другого вида связи в Internet, в транспортировке почты задействован ряд протоколов. Internet-протоколы основываются на документах RFC (Requests For Comment — запросы на комментарии), которые в случае принятия формируют используемый по обоюдному согласию язык для осуществления связи. Отправка e-mail в Internet основывается на протоколе SMTP (Simple Mail Transfer Protocol — простой протокол отправки почты), RFC 821, разработанном Джонатаном Постелом (Jonathan Postel). В 2001 году рабочей группой по информационно-вычислительным сетям (Network Working Group) была опубликована новая версия RFC 821, RFC 2821 под редакцией Дж. Кленсена (J. Klenssen), AT&T Laboratories.

Коммуникационная модель (концептуальная структура информационного обмена) заключается в том, что пользователь (и файловая система пользователя) использует клиентскую программу (например, Outlook или Eudora) для подключения (в технических терминах — для установки канала двухсторонней передачи) с SMTP-сервером. Затем сервер передает сообщение соответствующему получающему SMTP-серверу (основываясь на доменном имени завершающей части e-mail-адреса). Получающий SMTP-сервер отвечает за помещение данного сообщения в соответствующее место (которое обычно называется почтовым ящиком) или применение иных соответствующих функций.

В результате этого процесса почта перемещается от одного сервера к другому. Как получить почту от сервера? Для этого предназначены два протокола — POP3 и IMAP. Прежде чем доставить почту от удаленного сервера, они выполняют проверку регистрационной информации пользователя. Это можно представить себе как обычный почтовый ящик, ключ к которому вы передаете другому человеку. Если ключ подходит, то этот человек сможет доставить вам почту.

Структура e-mail-сообщения

E-mail-сообщение с правильной структурой имеет заголовки, за которыми следует тело сообщения. Кроме того, в сообщение также могут включаться отдельные файлы-вложения. Заголовки включают в себя e-mail-адрес получателя, e-mail-адрес отправителя, тему сообщения, список получателей копий данного сообщения и идентификатор сообщения. Не все эти заголовки обязательны, а некоторые могут быть определены пользователем, что упрощает их фальсификацию.

В RFC 2822 представлено определение составных частей e-mail-адреса в Internet. (Некоторые RFC можно найти на странице www.faqs.org/rfcs/.) Этот стандарт определяет сообщение как состоящее из US-ASCII-символов, разделенных на строки, каждая из которых завершается символом перевода строки и возврата каретки (CRLF). Поля заголовков представляют собой строки символов, разделенные с помощью специального синтаксиса. За заголовками следует тело сообщения, отделенное от заголовков пустой строкой (строка, в которой нет ничего кроме CRLF).

Поле заголовка представляет собой строку символов, содержащую имя поля, двоеточие и тело поля. Поля заголовков не обязательно должны следовать в каком-либо определенном порядке. Единственным обязательным полем является поле From, содержащее дату создания и адрес отправителя сообщения. В следующей таблице представлены возможные заголовки.

Поле заголовка	Описание
Trace	Включает в себя дату повторной отправки, адрес отправителя, адрес получателя и т.д. Используется, когда сообщение отправляется повторно (например, когда первая попытка доставки не удалась, сервер может попытаться отправить письмо позднее)
From	Обязательное поле. Содержит адрес отправителя. Следует, однако, учесть, что даже если это поле в сообщении присутствует, его точность не гарантируется. Если поле содержит несколько адресов, то в поле Sender должен присутствовать только один адрес. Дата создания включается в данные этого поля. Обратите внимание на заголовок From — это не то же самое поле, что необязательное поле “From:”
Sender	Содержит единственный адрес, с которого отправлено данное письмо
Reply-to	Содержит необязательный адрес для ответа
To	Содержит разделенный запятыми список адресатов
Cc	Содержит разделенный запятыми список адресатов, которым отправляются копии данного сообщения
Bcc	Содержит разделенный запятыми список адресатов, которым отправляются копии данного сообщения, при этом получатели не могут ни увидеть, ни узнать, каким Bcc-получателям отправлялось письмо
Message-id	Необязательное поле, но каждое сообщение должно его содержать. Содержит уникальный идентификатор сообщения
In-reply-to	Необязательное. Один или несколько идентификаторов сообщения
References	Необязательное. Один или несколько идентификаторов сообщения
Subject	Необязательное. Содержит короткую строку, определяющую тему сообщения
Comments	Необязательное. Комментарии о теле сообщения
Keywords	Необязательное. Разделенные запятыми ключевые слова, которые могут оказаться важными для пользователя
Optional	Необязательное. Поле должно лишь соответствовать инструкциям о форматировании, описанным в RFC 2822. Содержимое поля не уточняется

Ниже приводится пример сообщения с заголовками и телом (тело усечено в целях экономии места) в текстовом формате. Это сообщение было получено и отображено как Web-страница, в виде HTML, хотя здесь показан только текст. В заголовке “From” указано имя пользователя почтовой программы (в данном случае mailer@mailers.lindows.com) и дата создания письма. В строке “Received:” указан IP-адрес и “родословная” e-mail-сообщения. Кроме этого в письме есть строки “To:”, “Date:”, “From:” и “Subject:” (заголовок “From:” с двоеточием — не то же самое, что “From” в начале сообщения).

```
From mailer@mailers.lindows.com Thu Jan 22 17:52:58 2004
Received: from [130.94.123.236] (helo=mailer.lindows.com)
by mail1.servata.com with esmtp (Exim 3.35 #1 (Debian))
```

```
id 1AjqVG-0005LN-00
for <info@servata.com>; Thu, 22 Jan 2004 17:52:58 -0800
Received: (qmail 30557 invoked by uid 99); 23 Jan 2004 01:23:04 -0000
To: info@servata.com
Date: Thu, 22 Jan 2004 16:23:43 -0800
From: Michael Robertson <mailer@mailer.lindows.com>
Subject: Michael's Minute: Disagree with Linus MIME-Version: 1.0
Content-Type: multipart/alternative;
    boundary="==_LINDOWS_34a8357a5a341381f6a48042673dcb9c"
Message-Id: <E1AjqVG-0005LN-00@mail1.servata.com>

Lindows.com Michael's MinuteFrom: Michael Robertson [mailer@mailer.lindows.com]
Sent: Thursday, January 22, 2004 4:24 PM
To: info@servata.com
Subject: Michael's Minute: Disagree with Linus      If this message is not
    displaying properly, click here to launch it in your browser.

    Michael's Minute: Disagree with Linus

I've never personally met Linus Torvalds, the much heralded man behind Linux. ...
some computer manufacturers are now shipping computers preinstalled with
    desktop Linux through major retailers, making it practical
    for average or even new computer users to embrace Linux.
...
```

Отправка e-mail с помощью PHP

Прежде чем приступить к практическому рассмотрению отправки e-mail-сообщений, необходимо проверить некоторые параметры как в Windows, так и в Linux-инсталляции PHP.

В Windows должен быть доступ к SMTP-серверу — это может быть сервер ISP, если разработка ведется на персональной машине. Откройте файл `php.ini` и найдите раздел `[mail function]`. Задайте соответствующее значение для параметра SMTP; например:

```
SMTP = smtp.my.server.net
```

Затем задайте такое значение параметра `sendmail_from`, которое отражало бы ваш e-mail-адрес, например:

```
sendmail_from = davidm@doggiestouch.co.za
```

Linux-пользователям придется только установить параметр `sendmail_path`:

```
sendmail_path = smtp.my.server.net
```

Это все. Теперь можно использовать функцию `mail()` для отправки электронной почты.

Использование функции mail()

Функция `mail()` возвращает булево значение, т.е. можно использовать оператор `if` для проверки успешности ее выполнения. В качестве аргументов данная функция принимает адрес получателя, тему сообщения, тело сообщения и любые дополнительные заголовки, которые желательно отправить. Для отправки писем подходит код, подобный приведенному ниже. В данном случае письмо отправляется по заданному адресу и имеет тему и тело, состоящее из трех строк, разделенных символами новой строки (`\n`):

```
If (mail("joeblow@example.com", "Тема", "Первая строка\nВторая строка\nТретья строка")) {
    //сообщить пользователю, что письмо отправлено
} else {
    //сообщить пользователю об ошибке
}
```

Практика Отправка e-mail-сообщения

Вызванная в PHP-приложении функция `mail()` отправляет письма через стандартный MTA-агент. Логично предположить, что функция требует, как минимум, адреса получателя, тему сообщения и само сообщение (которое обычно называется телом). Чтобы добавить адрес отправителя (From) или любую другую дополнительную информацию, при формировании e-mail можно включить дополнительные заголовки. Рассмотрим небольшой сценарий `mail.php` (не забудьте заменить e-mail-адрес в переменной `$to` — предположим, что письмо необходимо отправить самому себе):

```
<?php
$to = "edit@contechst.com";
$subject = "Ваше письмо отправлено";
$body = "Тестовое письмо";
if (mail($to, $subject, $body)) {
    echo "<b>PHP отправил Ваше e-mail-письмо<b>";
}
?>
```

Если сетевое соединение установлено и SMTP-сервер работает правильно, то в окне браузера будет выведено подтверждающее сообщение. Следует, однако, отметить, что успешная отправка письма не означает, что это письмо будет принято получателем. Существует множество причин, по которым почта не доходит до получателя — указанный в письме адрес получателя может не соответствовать адресу на принимающем сервере, принимающий сервер может быть временно недоступен и т.д.

Как это работает

PHP-функция `mail()` позволяет пользователю отправлять e-mail-сообщения, указав несколько аргументов (таких как адрес получателя). Это булева функция, т.е. она возвращает `TRUE` при успешном завершении (почта была отправлена) и `FALSE` в противном случае. Аргументы можно указывать в виде текстовых строк или в виде переменных, содержащих соответствующие данные.

Предположим, что письмо дошло до адресата. Результат показан на рис. 15.1.

Рассмотрим каждый аргумент, принятый функцией `mail()`.

Заголовок “To:” автоматически добавляется функцией, а данные для него считываются из первого аргумента. Эти данные можно жестко запрограммировать, принять как переменную, сгенерированную в результате отправки пользователем формы (этот случай иллюстрируется далее) либо получить их из записей в базе данных или иного источника информации. В этом поле можно указать несколько получателей, разделяя e-mail-адреса запятыми (без пробелов). Запятые должны находиться между кавычками, окружающими e-mail-адреса, иначе они будут интерпретированы функцией как следующий аргумент.

Тема e-mail-сообщения автоматически считывается из второго аргумента функции. Тему также можно жестко запрограммировать, принять как значение переменной, полученное в результате отправки формы, или вставить из любого источника данных. Конечно, было бы немного странно отправлять всем получателям разные темы (если вы не спамер), но это определенно возможно.

Тело сообщения состоит из текста, и в него можно добавить слова приветствия и подпись. Тело сообщения — просто текст, и любые начальные и конечные фрагменты или другие компоненты добавляются с помощью оператора конкатенации (`.`).

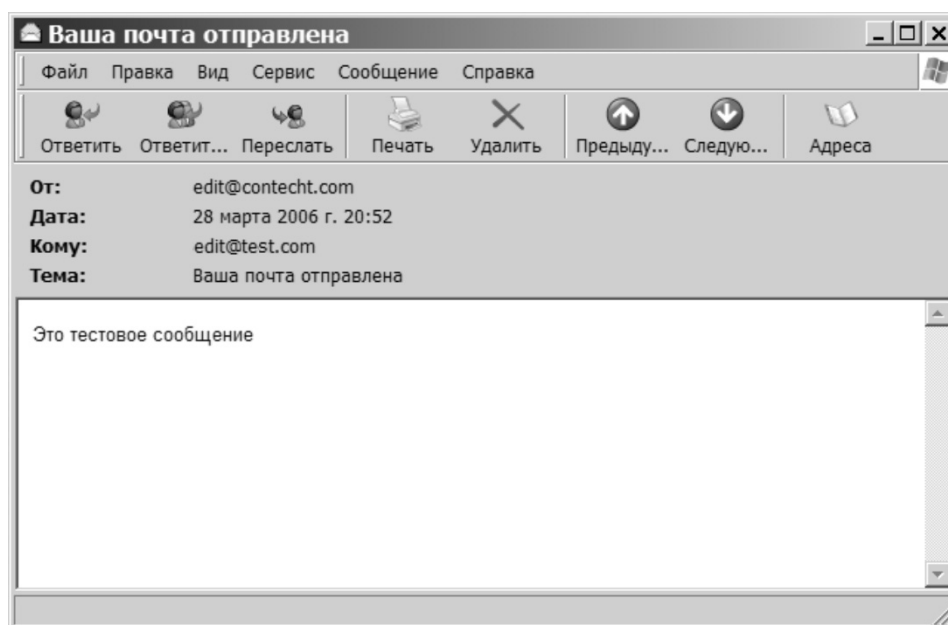


Рис. 15.1.

Дополнительные заголовки, такие как “From:”, можно добавлять как текстовые строки в качестве четвертого аргумента функции `mail()`. Каждый аргумент принимает форму заголовка, начинающегося с соответствующего ключевого слова с последующим двоеточием, например, `From:`. После ключевого слова следует строка, содержащая имя и e-mail-адрес (в угловых скобках) отправителя. Эта строка завершается символами возврата каретки и перевода строки. Следующий пример показывает, как формируется заголовок “From:”:

```
"From: $sender_name <$from>\r\n"
```

Если добавляются другие заголовки, то они могут быть собраны в переменной `$headers` при условии, что каждый из них завершается символами перевода строки и возврата каретки.

Некоторые МТА в Unix могут работать только с символом новой строки (`\n`).

В простом приложении, которое разрабатывается далее в настоящей главе, используются e-mail-адреса, вставляемые по одному за раз, но в более сложных приложениях (например, в диспетчерах рассылки новостей) для обработки множества e-mail-адресов могут использоваться циклы, такие как `for` или `do while`. Источниками e-mail-адресов для приложения могут быть записи базы данных, .xml-файлы или даже простые текстовые файлы.

До сих пор в книге рассматривались только простые текстовые сообщения. А что если в e-mail-письмо необходимо добавить вложение или отправить вместо простого текста данные другого формата? Далее описаны способы, позволяющие решать эти проблемы.

Многоцелевые расширения почты в Internet (MIME)

Формат MIME расширяет e-mail за рамки простого текста, позволяя указывать тип содержимого и кодировку сообщения. Формат MIME определяется стандартом RFC 2045, который опубликован на странице www.faqs.org/rfcs/rfc2045.html.

Поля MIME-заголовков

Как базовый текстовый e-mail-протокол, MIME использует поля заголовков для определения содержимого и структуры e-mail-сообщений, содержащих нестандартное текстовое наполнение, а также присоединяемые файлы. MIME-заголовки описываются ниже.

- ❑ **MIME Version** (версия MIME): указывает на то, что сообщение в целом соответствует RFC, если используется версия 1.0.
- ❑ **Content-Type** (тип содержимого): указывает на тип тела сообщения или вложенного файла. Отсутствие этого заголовка свидетельствует о том, что используется простой текстовый формат. Например, для отправки e-mail-сообщений в формате HTML (это очень популярно в наши дни) необходимо установить заголовки версии MIME и типа содержимого подобным образом:

```
$headers .= "MIME-Version: 1.0 \n";
```

```
$headers .= "Content-type: text/html; charset=iso-8859-1 \n";
```

- ❑ **Content-Transfer-Encoding** (транспортная кодировка содержимого): указывает кодировку данных, например, base64 для файла изображения.
- ❑ **Content-Id** (идентификатор содержимого; необязательный заголовок с исключениями): уникально обозначает MIME-блоки в сообщении, позволяя одному телу идентифицировать другое.
- ❑ **Content-Description** (необязательный заголовок): текстовое описание содержимого.
- ❑ **Content-Disposition** (расположение содержимого): рекомендует клиентскому приложению способ интерпретации части сообщения, к которой применяются текущие заголовки. Значением может быть `attachment` (вложение) или `inline` (непосредственно). Как и следует ожидать, часть сообщения присутствует либо в виде вложенного файла, либо отображается непосредственно. Непосредственное отображение удобно для HTML-писем, в которых используются изображения.

Многоэлементный MIME-формат

Формат MIME предусматривает отправку e-mail-сообщений с вложениями. Такие сообщения называются *многоэлементными MIME-сообщениями* (multipart MIME messages), потому что MIME-секция сообщения разделяется на несколько частей, каждая из которых имеет собственные MIME-заголовки. Общий тип содержимого (Content-Type) для таких сообщений — `multipart/mixed`, но в каждой подсекции может быть установлен ее собственный тип, например, `text/html` или другой.

Маркеры границ

Частью многоэлементного MIME-сообщения является конструкция маркера границ. При создании сообщения устанавливается параметр (который называется *граница* (*boundary*)), равный произвольно определенной текстовой строке. Это произвольная строка, но она не должна встречаться в теле сообщения (т.е. не должна быть

распространенным словом). Для создания маркеров границ можно задействовать функцию `md5()`, используя при этом значение текущего времени и какой-либо произвольный текст. В этом случае крайне маловероятно, что маркер границы встретится в составе тела сообщения.

Рассмотрим секции многоэлементного MIME-сообщения, которое представлено ниже. Следует отметить, как определяется, а затем используется граница для разделения разных частей целого сообщения.

```
//начало заголовка
```

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="0000_PHP5_0000";
Content-Transfer-Encoding: 7bit
```

Эта часть e-mail не должна быть видимой для того, кто читает письмо, при условии, что он использует MIME-совместимый e-mail-клиент.

```
--0000_PHP5_0000
```

```
//текстовый заголовок
```

```
Content-Type: text/plain; charset="iso-8859-1"
Content-Transfer-Encoding: 7bit
```

Это тело сообщения и оно должно быть простым читабельным текстом.

```
--0000_PHP5_0000
```

```
//заголовок для вложенного изображения
```

```
Content-Type: image/jpeg; name="myimage.jpg";
Content-Transfer-Encoding: base64
Content-Disposition: attachment
```

<данные файла myimage.jpg в кодировке base64>

```
--0000_PHP5_0000
```

```
//заголовок непосредственно вставленного изображения
```

```
Content-Type: image/jpeg; name="myimage.jpg";
Content-Transfer-Encoding: base64
Content-Disposition: inline
```

<данные файла myimage.jpg в кодировке base64>

```
--0000_PHP5_0000
```

HTML-письма

Хотя обычные e-mail-сообщения состоят из простого текста, можно так установить MIME-заголовки, чтобы тело сообщения передавалось в HTML-формате. В приложении, разработка которого рассматривается ниже, используется переключатель, для того чтобы пользователь мог выбрать формат письма — простой текст или HTML. Если пользователь выбрал HTML, то приложение создает MIME-заголовки, в которых определяется тип содержимого (`Content-Type`) — `text/html`.

В дополнение к правильной установке заголовков для формата HTML, в приложении используется весьма удобный инструмент, позволяющий пользователю создавать HTML-код изображениями и текстом. Речь идет о JavaScript-сценарии `htmlArea`, соз-

данным и бесплатно распространяемым компанией Interactivetools.com, Inc. Небольшой фрагмент кода для вставки редактора `htmlArea` показан в коде Web-страницы приложения, хотя его исходный код не приводится. Весь код и изображения, составляющие редактор, можно загрузить с сайта www.interactivetools.com. Он не связан с какими-либо почтовыми функциями PHP, но очень удобен, так как позволяет визуально создать тело сообщения для HTML-письма.

Библиотеки почтовых функций в PEAR

Репозиторий PEAR содержит несколько классов, которые можно использовать для работы с электронной почтой. Чтобы заставить их работать в PHP-сценариях, понадобится установить соответствующие PEAR-библиотеки (более подробная информация о PEAR приведена в главе 14).

После установки можно поэкспериментировать с новыми экземплярами почтовых PEAR-объектов. В состав почтовых классов PEAR входят следующие пакеты:

- ☐ `Mail`: отправляет e-mail-сообщения с помощью встроенной в PHP функции `mail()`;
- ☐ `Mail_IMAP`: предоставляет функции библиотеки c-client для Web-почты;
- ☐ `Mail_Mbox`: PHP-класс для разбора и использования Unix MBOX;
- ☐ `Mail_Mime`: предоставляет классы для создания и декодирования MIME-сообщений;
- ☐ `Mail_Queue`: позволяет накапливать почту в очередях, а затем отправлять ее в фоновом режиме.

Создание простого PHP-приложения для работы с e-mail

Разрабатываемое здесь приложение способно создавать и отправлять e-mail-сообщения, используя MIME-формат и PHP-функцию `mail()`; здесь предполагается, что установлена программа Sendmail. Чтобы приложение могло отправлять письма с несколькими вложенными файлами, эти файлы должны находиться на локальном сервере. (В главе 7 рассматривалось создание простого приложения, позволяющего загружать файлы на сервер. Будем считать, что блок загрузки файлов уже выполнил свои функции и что необходимые файлы были помещены в каталог `atts`.)

Приложение считывает файлы и предоставляет пользователю возможность выбрать (из списка) один или несколько файлов для отправки их в письме. При отправке HTML-писем пользователь может указать имена файлов, присутствующих в каталоге `atts`, и отправить их вместе с письмом для непосредственного отображения в сообщении.

Когда пользователь открывает приложение в браузере, появляется HTML-страница. JavaScript-сценарий добавляется в дескрипторе `<head>` этой страницы и создает визуальный HTML-редактор (редактируемую текстовую область). После тега `</head>` открывается дескриптор `<body>` и таблица, заключающая в себе всю форму. Позднее другой JavaScript-блок в элементе формы фактически вызывает первый JavaScript-сценарий для создания текстовой области; следует отметить, что это происходит, если используется браузер Internet Explorer версии 5.5 или выше. Итак, запустите HTML-редактор и начинайте вводить следующий код. По мере рассмотрения функций различных частей создаваемого приложения здесь будут поясняться новые фрагменты кода.


```

<html>
<head>
  <title>Почтовые функции PHP</title>
  <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
  <script language="Javascript1.2"><!-- // Загрузить htmlarea
    _editor_url = "http://www.bigtip.com/php5/Chapter15/htmlarea/";
    // URL к файлам htmlarea
  var win_ie_ver = parseFloat(navigator.appVersion.split("MSIE")[1]);
  if (navigator.userAgent.indexOf('Mac') >= 0) { win_ie_ver = 0; }
  if (navigator.userAgent.indexOf('Windows CE') >= 0) { win_ie_ver = 0; }
  if (navigator.userAgent.indexOf('Opera') >= 0) { win_ie_ver = 0; }
  if (win_ie_ver >= 5.5) {
    document.write('<scr' + 'ipt src="' + _editor_url + 'editor.js">');
    document.write(' language="Javascript1.2"></scr' + 'ipt>'); } else {
    document.write('<scr'+ 'ipt>function editor_generate(){
      return false; }</scr'+ 'ipt>'); }
  // --></script>
</head>

<body bgcolor="#FFFFFF">
<table width="100%" border="0" cellpadding="10">
  <tr>
    <td>
      <h2>Использование PHP для отправки e-mail</h2>

```

В этом месте HTML-кода логично разместить форму. Так как форма должна отображать перечень файлов в каталоге `atts`, готовых к отправке в виде вложений, необходимо проинициализировать переменную `$default_directory` так, чтобы она содержала имя этого каталога:

```

<?php
//Каталог для присоединяемых файлов
$default_dir = "./atts";

```

Если пользователь уже видел форму, заполнил и отправил ее, то запускать главный PHP-код нет необходимости. Поэтому в условии оператора `if` используется переменная `$_POST['posted']` для проверки того, была ли отправлена форма. Если форма была отправлена, то PHP-код может начинать формирование e-mail-сообщения для отправки.

Если форма не была отправлена, то условие оператора `if` ложно (переменная `$_POST['posted']` не установлена), весь PHP-код обработки данных формы пропускается и отображается HTML-форма. Очевидно что когда пользователь впервые открывает страницу или когда обновляет ее, не нажимая кнопку Отправить, PHP-код обработки данных формы не выполняется — вместо этого выводится сама форма.

Следующий фрагмент кода проверяет, была ли отправлена форма, и если да, то записывает имя отправителя в переменную:

```

if (!empty($_POST)) {
  $sender_name = $_POST['first_name'] . " " . $_POST['last_name'];
}

```

Далее с помощью значения, сформированного переключателем `html_or_text`, выясняется тип отправляемого сообщения (HTML-сообщение: письмо с вложением или письмо в стандартном, простом текстовом формате):

```

if ($_POST['html_or_text'] == "html") {

```

Если значение переключателя равно `html`, то необходимо определить наличие присоединяемых файлов, проверив количество элементов в массиве `$attachments`:

```

if (count($_POST['attachments']) > 0) {

```

Если количество элементов больше нуля (есть файлы для отправки), то инициализируем переменную счетчика (\$cnt) со значением 0 и создаем значение маркера границы для разделения MIME-заголовков:

```
$cnt = 0;
$boundary = "0000_PHP5_0000";
```

Затем сценарий в цикле проходит по всем присоединяемым файлам, открывает каждый из них, извлекает его содержимое и создает соответствующий MIME-заголовок, разделяя одновременно с этим содержимое файла на блоки, а затем закрывает файл:

```
for ($i = 0; $i < count($_POST['attachments']); $i++) {
    $fp = fopen($default_dir . "\\" .
        $_POST['attachments'][$i], "rb");
    $file_name = basename($_POST['attachments'][$i]);
    $content[$cnt] = fread($fp, filesize($default_dir . "/" .
        $_POST['attachments'][$i]));
    $files_attached = "";
    $files_attached .= "--$boundary\n"
        . "Content-Type: image/jpeg; name=\"$file_name\"\n"
        . "Content-Transfer-Encoding: base64\n"
        . "Content-Disposition: inline; filename=\"$file_name\"\n"
        . chunk_split(base64_encode($content[$cnt])) . "\n";
    $cnt++;
    fclose($fp);
}
```

После того как вложения сформированы, создаются заголовки From:, CC:, BCC: и Reply-To; строка приветствия, тело сообщения и подпись, а затем главный MIME-заголовок (хотя HTML-сообщения могут также содержать вложения, здесь эта возможность не рассматривается; предположим, что основное сообщение имеет простой текстовый формат):

```
$from_header = "From: $sender_name <$_POST[from]>\nCC:
    $_POST[cc] \nBCC: $_POST[bcc]\nReply-To: $_POST[from]\n";
$salutation = $_POST['salutation'] . "\n\n";
$body = $salutation . $_POST['body'] . "\n\n" . $_POST['regards'];

// Создаем главный MIME-заголовок, добавляем тело сообщения и
// присоединяемые файлы
$files_attached .= "--.$boundary.\n";
$add_header = "";
$add_header .= "MIME-Version: 1.0\n" . "Content-Type:
    multipart/mixed; boundary=\"$boundary\"; Message-ID:
    <".md5($_POST['from'])."@example.com>";
$mail_content="--.$boundary.\n"
    . "Content-Type: text/plain; charset=\"windows-1251\"\n"
    . "Content-Transfer-Encoding: 8bit\n\n"
    . $body . "\n\n" . $files_attached;

$body = $mail_content;
```

Следует отметить, что переменной \$body присваивается значение переменной \$mail_content (в которой содержится тело сообщения и содержимое присоединяемых файлов). Таким образом, впоследствии можно просто передать значение переменной \$body функции mail().

Если e-mail-сообщение форматируется как HTML, то перед отправкой сообщения используется следующий код для создания нескольких заголовков:

```
} else {
    $salutation = $_POST['salutation'];
```

```

$salutation = $salutation . "<br><br>";
$body = $salutation . stripslashes($_POST['body']) . "<br><br>" .
    $_POST['regards'];

// Формируем HTML-заголовки
$from_header = "From: $sender_name <$_POST[from]>\nCC:
    $_POST[cc]\nBCC: $_POST[bcc]\nReply-To: $_POST[from]\n";
$add_header = "MIME-Version: 1.0\n";
$add_header .= "Content-type: text/html; charset=windows-1251\n";
}

```

Для простых текстовых e-mail-сообщений также формируются заголовки:

```

} else {
    // для простых текстовых сообщений без вложений
    $from_header = "From: $sender_name <$_POST[from]> \nCC: $_POST[cc]\nBCC:
        $_POST[bcc]\nReply-To: $_POST[from]\n";
    $salutation = $_POST['salutation'];
    $salutation = $salutation . "\n\n";
    $body = $_POST['body'];
    $body = $salutation . $body . "\n\n" . $_POST['regards'];
}

```

Затем адреса получателей (To:) записываются в одну переменную \$to:

```

$to = "$_POST[to]";

do{
    next($_POST);
}while (key($_POST) !== 'to');
for ($i = 1; $i <=7; $i++) {
    $next = next($_POST);
    if(!empty($next)){
        $to = $to . ", " . $next;
    }
}

```

После чего выполняется минимальная проверка e-mail-адресов и почта отправляется. Если почта не отправлена или в e-mail-адресе есть ошибка, то выводится соответствующее сообщение:

```

if (strpos($_POST['to'], "@") >= 0) {

    // Отправляем почту
    echo "<BR>To: $to<P>";
    echo "Subject: $_POST[subject]<P>";
    echo "Body: $body<P>";
    echo "$from_header<P>";
    echo "$add_header<P>";
    if(!isset($add_header)){
        if (mail($to, $_POST['subject'], $body)){
            echo "<h3>Ваше письмо отправлено</h3>";
        } else {
            echo "Возникла ошибка и письмо не было отправлено";
        }
    }else if (mail($to, $_POST['subject'], $body, "$from_header".
        "$add_header")) {

        echo "<h3>Ваше письмо отправлено</h3>";
    } else {
        echo "Возникла ошибка и письмо не было отправлено";
    }
    } else {
        echo "Обнаружен неправильный e-mail-адрес";
    }
}

```

Если пользователь не отправлял форму, то следует просто отобразить ее и дать пользователю возможность ввести свой e-mail-адрес и другие данные:

```

} else {
?>
<form method="POST" action="php_mail.php">
<input type="hidden" name="posted" value="true">
<table width="100%" border="1">
<tr>
<td width="16%" valign="top"><font face="Arial, Helvetica,
sans-serif" size="-1"><b>Ваше имя:</b></font></td>
<td width="84%"><font size="-1" face="Arial, Helvetica,
sans-serif"><b>Имя</b></font>
<input type="text" name="first_name">
<b><font size="-1" face="Arial, Helvetica, sans-
serif">Фамилия</font></b>
<input type="text" name="last_name">
</td>
</tr>
<tr>
<td width="16%" valign="top"><b><font face="Arial,
Helvetica, sans-serif" size="-1">От:</font></b></td>
<td width="84%">
<input type="text" name="from">
</td>
</tr>
<tr>
<td width="16%" valign="top"><b><font face="Arial,
Helvetica, sans-serif" size="-1">Кому:</font></b></td>
<td width="84%">
<input type="text" name="to">
<input type="text" name="to01">
<input type="text" name="to02">
<input type="text" name="to03">
<input type="text" name="to04">
<input type="text" name="to05">
<input type="text" name="to06">
<input type="text" name="to07">
</td>
</tr>
<tr>
<td width="16%" valign="top"><b><font face="Arial,
Helvetica, sans-serif" size="-1">Копия (CC):</font></b></td>
<td width="84%">
<input type="text" name="cc">
</td>
</tr>
<tr>
<td width="16%" valign="top"><b><font face="Arial,
Helvetica, sans-serif" size="-1">Скрытая копия (BCC):</font></b></td>
<td width="84%">
<input type="text" name="bcc">
</td>
</tr>
<tr>
<td width="16%" valign="top"><b><font face="Arial,
Helvetica, sans-serif" size="-1">Тема:</font></b></td>
<td width="84%">
<input type="text" name="subject">
</td>
</tr>
<tr>
<td width="16%" valign="top"><b><font face="Arial,
Helvetica, sans-serif" size="-1">Вложения:<br>

```

```

        Чтобы удалить вложение, щелкните левой клавишей мыши,
        удерживая клавишу Ctrl</font></b></td>
<TD width="84%">
    <select name="attachments[]" size="4" multiple>

<?php
// заполняем выпадающий список именами доступных файлов
if(!($dp = opendir($default_dir))) {
    die("Невозможно открыть $default_dir.");
} else {
    while($file = readdir($dp)) {
        if($file != '.' && $file != '..') {m
?>
        <OPTION value="<?php echo $file; ?>"><?php echo $file; ?></option>
<?php
    }
}
closedir($dp);
}
?>

        </select>
    </td>
</tr>
<tr>
    <td width="16%" valign="top"><b><font face="Arial, Helvetica,
    sans-serif" size="-1">Приветствие:</font></b></td>
    <td width="84%">
        <input type="text" name="salutation">
    </td>
</tr>
<tr>
    <td width="16%" valign="top"><b><font face="Arial,
    Helvetica, sans-serif" size="-1">Тело сообщения:</font></b></td>
    <td width="84%">
        <textarea name="body" cols="40" rows="10"></textarea>
<script language="javascript1.2">
editor_generate('body');
</script>
    </td>
</tr>
<tr>
    <td width="16%" valign="top"><b><font face="Arial,
    Helvetica, sans-serif" size="-1">Подпись:</font></b></td>
    <td width="84%">
        <input type="text" name="regards">
    </td>
</tr>
<tr>
    <td width="16%" valign="top">&nbsp;</td>
    <td width="84%"> <font face="Arial, Helvetica, sans-serif"><b>
<font
size="-1">HTML или
    присоединяемые файлы
    <input type="radio" name="html_or_text" value="html">
    Простой текст
    <input type="radio" name="html_or_text" value="text" checked>
    <input type="submit" name="Submit" value="Отправить e-mail">
    </font> </b> </font> </td>
</tr>
</table>
</form>
<?php
}

```

```
?>
</td>
</tr>
</table>
</body>
</html>
?>
```

Сохраните файл как `php_mail.php`, а затем откройте его в браузере. Первоначальная страница приложения показана на рис. 15.2.

Почтовые функции PHP - Microsoft Internet Explorer

Файл Правка Вид Избранное Сервис Справка

Адрес: http://localhost/php5/Ch15/php_mail.php

Использование PHP для отправки e-mail

Ваше имя: Имя Фамилия

От:

Кому:

Копия (CC):

Скрытая копия (BCC):

Тема:

Вложения: Чтобы удалить вложение, щелкните левой клавишей мыши, удерживая клавишу Ctrl

Приветствие:

Тело сообщения:

Подпись:

HTML или присоединяемые файлы ☒ Простой текст ☐

Готово Местная интрасеть

Рис. 15.2.

На рис. 15.3 показано подтверждающее сообщение, которое получает пользователь после нажатия кнопки Отправить e-mail.

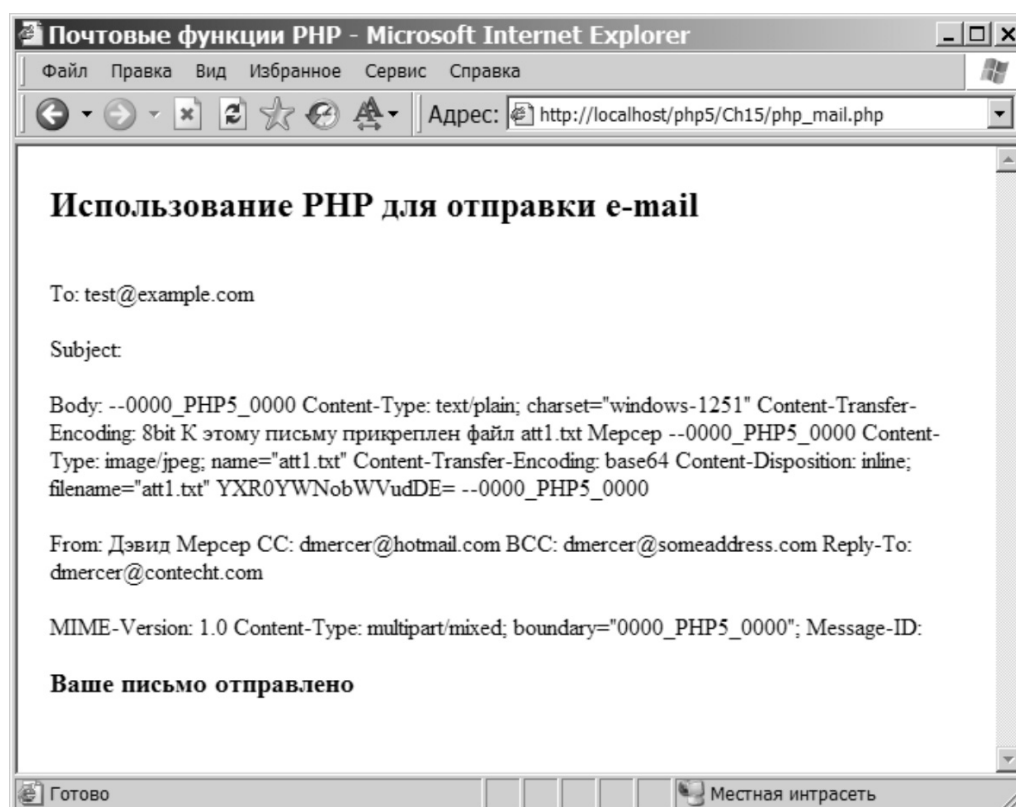


Рис. 15.3.

Однако это еще не все.

Дело в том, что если определенные поля (CC, BCC или другие) не заполнены, выводится сообщение об ошибке. В упражнении, приведенном в конце главы, необходимо модифицировать сценарий так, чтобы он мог генерировать правильно сформированные заголовки независимо от того, ввел ли пользователь соответствующие данные.

Резюме

В главе рассматривались почтовые протоколы Internet, структура e-mail-сообщений, функция `mail()` и ее аргументы, а также методика отправки многоэлементных и HTML-сообщений с помощью PHP. Представленное в главе учебное приложение может быть хорошей отправной точкой, если разработчику придется реализовывать e-mail-функциональность на Web-сайтах. Эта глава должна послужить той базой, с которой можно начинать изучение сложностей Internet-почты.

Более подробную информацию об электронной почте можно найти на сайте Международного почтового консорциума (International Mail Consortium), www.imc.org.

Упражнение

Представленное в главе приложение хорошо работает в случае создания и отправки одиночных писем. Модифицируйте это приложение так, чтобы оно позволяло отправлять сообщения списку получателей, взятому из базы данных.

16

Генерирование графики

До сих пор в книге рассматривалось использование PHP для взаимодействия с пользователем, подключения и манипулирования базами данных и отправки информации по e-mail. Другой аспект Web-технологий — добавление графики на страницы. Речь идет не о создании изображений для HTML-страниц с помощью графических редакторов, а об использовании PHP для генерирования графики во время выполнения сценария. На страницах книги уже обсуждалось использование информации, полученной из базы данных, для создания динамических страниц. Теперь изучим возможность динамического создания изображений.

В PHP имеется широкий диапазон функций, позволяющих открывать, манипулировать и выводить графику как в Web-браузер, так и на диск. В этой главе рассматривается работа этих функций и их использование для создания графики для Web-страниц. Сначала рассмотрим некоторые базовые понятия, прежде чем создавать изображения: например, теория цвета и то, как в PHP работают системы координат изображений (image coordinate systems). Затем, используя PHP-средства для рисования, читатель научится создавать изображения с нуля, чертить прямые линии, кривые и различные фигуры в изображениях. Также в главе освещаются вопросы работы с существующими изображениями, например, создание водяных знаков в изображениях, создание пиктограмм и добавление текста в изображение.

Функции обработки изображений, которые используются в PHP, основаны на библиотеке GD, разработанной компанией Boutell.Com (www.boutell.com). Начиная с версии PHP 4.3 код библиотеки был встроен в PHP-инсталляцию и включает в себя некоторые усовершенствования первоначального кода, такие как альфа-сопряжение. Встроенная версия библиотеки GD хорошо поддерживается в PHP и, таким образом, более стабильна. С PHP5 поставляется GD версии 2.0.15, и теперь, когда истекает срок действия патентов на LZW-сжатие, в библиотеку включена только поддержка операций чтения файлов формата GIF. Кроме чтения GIF библиотека позволяет также читать и записывать файлы форматов JPEG, PNG и WBMP.

Основы компьютерной графики

Сначала необходимо изучить некоторые базовые понятия, связанные с изображениями. Для работы с графикой следует понимать основы теории цвета и цветовую модель RGB, позиционирование элементов изображения с помощью библиотек GD, а также представлять себе особенности различных типов изображений.

Теория цвета

Компьютеры создают цвета, опираясь на теоретическую модель, которая называется RGB. Модель RGB (red, green, blue) означает красный, зеленый и синий — три основных цвета, различные комбинации которых позволяют создавать цвета, отображаемые на экране монитора. RGB-модель также называется *аддитивной цветовой моделью* (*additive color model*), потому что окончательный цвет создается комбинацией разного количества красного, зеленого и синего цветов. Значения красного, зеленого и синего цветов находятся в диапазоне от 0 до 255, а окончательный цвет определяется комбинацией трех значений. Например, чистый синий цвет имеет RGB-значение 0,0,255 — значения красного и зеленого цветов равны 0, а значение синего цвета установлено максимальным (255). Таким образом, максимальное количество цветов, которое доступно в модели RGB, равно $256 \times 256 \times 256 = 16,7$ миллиона.

Когда значения всех компонентов красного, зеленого и синего равны 0, получается полное отсутствие цвета или черный цвет. Аналогично, белый цвет представлен максимальными значениями всех цветов — 255.

Отдельные значения для красного, зеленого и синего цвета хранятся в 8-битовых числах — число 255 представляет собой десятичное значение восьмизначного двоичного числа 11111111. Сложение этих восьмибитовых чисел дает битовую глубину RGB-изображения — 24 бита. Обычно битовая глубина находится в диапазоне от 1 до 64 — битовая глубина равная 1 позволяет использовать только два цвета — черный и белый, битовая глубина 8 позволяет использовать 256 цветов, а битовая глубина 64 позволяет использовать в изображении миллионы цветов.

Как будет показано далее, библиотеки GD позволяют создавать как 8- так и 24-битовые изображения.

Системы координат

Чтобы рисовать внутри изображения фигуры и писать текст, должна быть возможность позиционировать эти фигуры и текст в изображении. Читателю несомненно известна система координат, в которой координаты x и y отсчитываются вправо вверх от левого нижнего угла (рис. 16.1).

В графических функциях PHP координаты отсчитываются вправо вниз из верхнего левого угла (рис. 16.2).

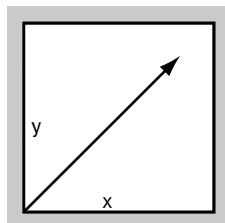


Рис. 16.1.

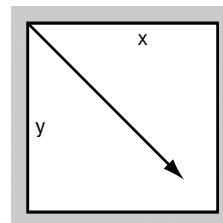


Рис. 16.2.

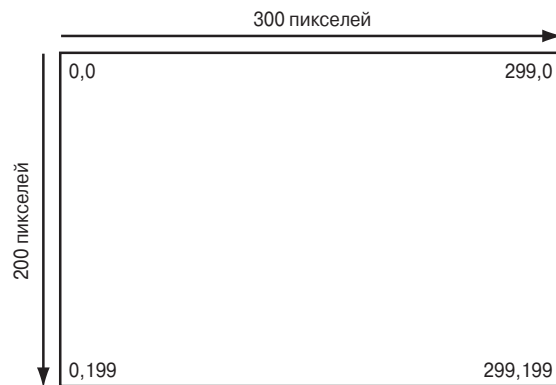


Рис. 16.3.

Первый пиксель в левом верхнем углу имеет координаты 0,0. Это означает, что последний по горизонтали пиксель в изображении имеет горизонтальную координату равную ширине изображения минус 1, а последний по вертикали пиксель имеет вертикальную координату равную высоте изображения минус 1. На рис. 16.3 показано изображение шириной 300 пикселей и высотой 200 пикселей и его угловые координаты в PHP.

Типы изображений

Как правило, встречается два типа изображений: векторные и растровые. В векторном изображении используются математические уравнения, описывающие фигуры, из которых состоит изображение. Хорошим примером векторного формата является SVG (Scalable Vector Graphics — масштабируемая векторная графика). Векторные изображения хорошо подходят для диаграмм, в которых используются прямые линии, кривые, а также разноцветные блоки, но не подходят для изображений фотографического типа. Растровые изображения состоят из пиксельных данных — изображение 20×20 пикселей составляют 400 отдельных пикселей, и каждый из этих пикселей имеет собственное RGB-значение.

Современные GD-библиотеки позволяют генерировать изображения трех основных файловых форматов Web — JPEG и PNG для Web-браузеров в настольных системах, а также WBMP для PDA-браузеров. Все указанные форматы являются растровыми. GD-функции в PHP не генерируют векторные изображения.

JPEG, PNG и WBMP — файловые форматы со сжатием. Это означает, что в них используются математические алгоритмы для сокращения количества данных, необходимых для полного описания изображения. Эти алгоритмы весьма важны, поскольку позволяют сохранить небольшие размеры файлов и сокращают время загрузки изображений.

Очень важно знать, когда следует использовать тот или иной формат. Хотя JPEG и PNG являются растровыми форматами, в них используются различные методики сжатия, и они не одинаково хорошо подходят для разных изображений.

В формате JPEG используется сжатие *с потерями*, в котором некоторые данные исходного изображения теряются в процессе сжатия. Этот формат наилучшим образом подходит для работы с такими изображениями, как фотографии, в которых есть множество едва различимых полутонов, но очень мало точных деталей. Этот формат следует применять, когда небольшая потеря качества не будет слишком заметной.

С другой стороны в формате PNG используется сжатие *без потерь*. Лучшее всего его использовать для изображений, которые содержат линии и крупные блоки цвета, на-

пример, для мультипликационных изображений. Когда изображение распаковывается, в нем содержится вся исходная информация. Это означает, что четкие границы и прямые линии (которые страдают от JPEG-сжатия) будут точно воспроизведены.

Ранние версии GD (и соответственно PHP) поддерживали формат GIF, который во многих аспектах похож на PNG. Однако компания Unisys запатентовала алгоритм, используемый для создания сжатых GIF-файлов, и в результате в версии GD 1.6 поддержка GIF была полностью заменена поддержкой формата PNG. Хотя форматы JPEG и PNG должно быть достаточно для всех потребностей разработчика в графике, в PHP также включена поддержка чтения формата GIF.

Работа с растровыми изображениями

Рассмотрим возможности PHP по созданию и улучшению изображений. В PHP создание и модификация изображений предполагает четыре этапа.

1. Создание пустого холста для изображения, с которым будет работать PHP. Это, по сути, зарезервированная область памяти, в которую графические функции PHP будут записывать данные создаваемого изображения.
2. Работа, связанная с рисованием изображения. Сюда включается настройка цветов, рисование фигур и текста внутри изображения.
3. Отправка завершеного изображения в Web-браузер или сохранение изображения на диске.
4. Удаление изображения из памяти сервера.

Создание нового изображения

Сначала необходимо создать пустой холст нового изображения, с которым будет работать PHP. Для этого можно либо использовать функцию `imagecreate()`, создающую индексированное изображение с максимальным числом цветов 256, либо функцию `imagecreatetruecolor()`, которая создает полноцветное изображение (максимум 16,7 млн цветов). Обе указанные функции принимают два параметра — ширину и высоту пустого изображения, которое необходимо создать:

```
resource imagecreate (int x_size, int y_size)
```

```
$myImage = imagecreate(200, 150);
```

Созданное в результате такого вызова изображение имеет ширину 200 и высоту 150 пикселей. Кроме того, функция возвращает значение. В данном случае переменная `$myImage` содержит идентификатор изображения, который ссылается на новое пустое изображение в памяти. Этот идентификатор используется в других графических функциях для того, чтобы однозначно идентифицировать в памяти изображение, которое необходимо обрабатывать. Идентификатор изображения аналогичен дескриптору файла или идентификатору соединения с базой данных, которые упоминались в предыдущих главах.

Распределение цветов

Прежде чем начинать создавать пустое изображение, необходимо решить, какие цвета будут использоваться, а затем вызвать функцию `imagecolorallocate()`, которая принимает четыре параметра:

```
int imagecolorallocate (resource image, int red, int green, int blue)
```

В PHP-коде это делается следующим образом:

```
$myGreen = imagecolorallocate($myImage, 51, 153, 51);
```

Первый параметр — идентификатор изображения, для которого необходимо создать цвет. Это тот идентификатор, который возвращается функцией `imagecreate()` или `imagecreatetruecolor()`. При одновременной работе с двумя изображениями для выделения в каждом из них одного и того же цвета необходимо вызвать функцию `imagecolorallocate()` дважды, по одному разу для каждого изображения.

Следующие три параметра — значения красного, зеленого и синего цвета, из которых должен быть создан желаемый цвет. Как уже отмечалось, каждое из этих значений может находиться в диапазоне от 0 до 255 и определяет цвет в модели RGB.

Функция `imagecolorallocate()` возвращает идентификатор только что созданного цвета в пределах палитры изображения, либо значение `-1`, если в палитру невозможно добавить цвет. Такое случается, когда в палитре уже содержится 256 цветов и для нового цвета просто нет места. В таком случае можно использовать функцию `imagecolorresolve()`, которая всегда возвращает корректный идентификатор цвета.

Функция `imagecolorresolve()` принимает те же параметры, что и `imagecolorallocate()` (идентификатор изображения и значения красного, зеленого и синего для формирования необходимого цвета), но в отличие от последней функции, которая просто пытается поместить необходимый цвет в палитру изображения, функция `imagecolorresolve()` сначала пытается выяснить, имеется ли этот цвет уже в палитре. Если это так, то функция возвращает индекс цвета. Если нет, то функция пытается добавить цвет в палитру. В случае успеха она возвращает идентификатор цвета в палитре. Если добавить цвет в палитру не удалось, то функция просматривает все цвета в палитре и возвращает идентификатор того цвета, который больше всех похож на запрашиваемый цвет.

Цвет, добавляемый в индексированное изображение (созданное функцией `imagecreate()`) первым, используется в качестве фонового цвета этого изображения. Изображения, созданные с помощью функции `imagecreatetruecolor()`, получают черный фон, и затем программист сам определяет цвет фона.

Основные функции рисования

Как только в палитру помещены все необходимые цвета, их можно использовать для рисования на пустом холсте. Графическая библиотека PHP предоставляет функции для рисования точек, линий, прямоугольников, эллипсов, дуг и многоугольников.

Все функции рисования в PHP имеют аналогичные наборы принимаемых параметров. Первым параметром всегда является идентификатор изображения, которое требуется нарисовать. Количество последующих параметров может варьироваться, но всегда есть координаты x и y (в пикселях), необходимые для рисования фигур или линий. Чтобы нарисовать только один пиксель, нужно указать только одну пару координат, а для рисования линии указываются x и y -координаты для начальной и для конечной точки этой линии. В качестве последнего параметра всегда указывается цвет рисуемой линии.

Рисование отдельных пикселей

Чтобы нарисовать на холсте один пиксель, можно использовать функцию `image-setpixel()`:

```
int image-setpixel (resource image, int x, int y, int color)
```

```
imagesetpixel($myImage, 120, 60, $myBlack);
```

В данном случае функция задает цвет, определенный в палитре как `$myBlack`, пикселю, который является 121-м по горизонтали и 61-м по вертикали в изображении `$myImage`. На рис. 16.4 показано расположение этого пикселя в изображении.

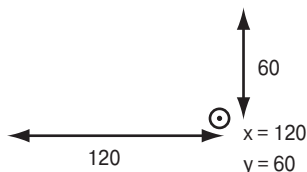


Рис. 16.4.

Стоит еще раз напомнить, что первый пиксель находится в левом верхнем углу изображения и имеет координаты 0,0, именно поэтому координата x равная 120 пикселям фактически соответствует 121 пикселю в изображении, считая слева направо.

Рисование линий

Для рисования линии в изображении используется функция `imageline()`. Так как линия имеет начальную и конечную точки, необходимо передать функции дополнительную пару координат:

```
int imageline (resource image, int x1, int y1, int x2, int y2, int color)
```

Практика Рисование линии

1. Откройте текстовый редактор и введите следующий код:

```
<?php
$myImage = imagecreate(200,100);
$myGrey = imagecolorallocate($myImage,204,204,204);
$myBlack = imagecolorallocate($myImage, 0, 0, 0);
imageline($myImage, 15, 35, 120, 60, $myBlack);
header("Content-type: image/png");
imagepng($myImage);
imagedestroy($myImage);
?>
```

2. Сохраните файл как `line.php`.
3. Откройте файл в Web-браузере. На рис. 16.5 показан результат работы сценария.

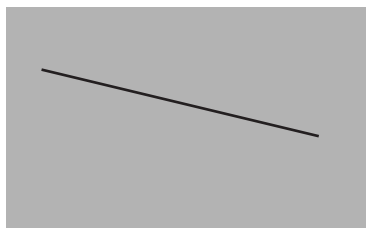


Рис. 16.5.

Если расширение GD не включено (чтобы включить его, нужно раскомментировать в файле `php.ini` строку `extension=php_gd2.dll`), то PHP сгенерирует сообщение об ошибке.

После внесения изменений в конфигурацию PHP не забудьте перезапустить сервер.

Как это работает

Сначала создается новое пустое изображение. Функция `imagecreate()` возвращает идентификатор изображения, который сохраняется в переменной `$myImage`. Затем в палитру изображения добавляются два цвета — серый и черный. Поскольку серый цвет добавляется первым, он используется в качестве фона в изображении.

Чтобы нарисовать линию, используется функция `imageline()`. Первым параметром всегда является идентификатор изображения. Два следующих параметра указывают функции `imageline()`, где начинается линия — в данном случае на расстоянии 16 пикселей вправо и 36 пикселей вниз от левого верхнего угла. Следующая пара параметров определяет конечную точку линии — 121 пиксель вправо и 61 вниз. Последний передаваемый функции параметр — цвет линии.

После того как изображение создано, его необходимо отправить в Web-браузер. Эта операция выполняется в два этапа.

1. Для отправки Web-браузеру заголовка, указывающего тип передаваемых данных, используется функция `header()`. Это гарантирует, что браузер корректно отобразит содержимое.
2. Используется функция `imagepng()`, которой в качестве параметра передается идентификатор изображения. Эта функция отправляет Web-браузеру данные изображения, которые хранятся в переменной `$myImage`, в виде PNG-файла.

Наконец, с помощью функции `imagedestroy()` изображение удаляется из памяти.

Рисование прямоугольников

Чтобы нарисовать прямоугольник, требуется указать только две точки в изображении — два противоположных угла прямоугольника. Поэтому синтаксис функции `imagerectangle()` в точности повторяет синтаксис `imageline()`. В данном случае две передаваемые пары координат используются для определения противоположных углов прямоугольника:

```
int imagerectangle (resource image, int x1, int y1, int x2, int y2, int col)
```

Откройте только что созданный файл `line.php` и сохраните его как `rectangle.php`. Замените строку:

```
imageline($myImage, 15, 35, 120, 60, $myBlack);
```

строкой:

```
imagerectangle($myImage, 15, 35, 120, 60, $myBlack);
```

Очевидно, что значения передаваемых функции `imagerectangle()` параметров те же, что и в примере с рисованием линии. Сохраните файл и откройте его в Web-браузере. Изображение, сгенерированное этим сценарием, показано на рис. 16.6.

Функция `imagerectangle()` использует первую пару координат как координаты левого верхнего угла прямоугольника, а вторую пару — как координаты правого нижнего угла. Если оставить обе функции, `imageline()` и `imagerectangle()`, то сценарий сгенерирует результат, показанный на рис. 16.7.

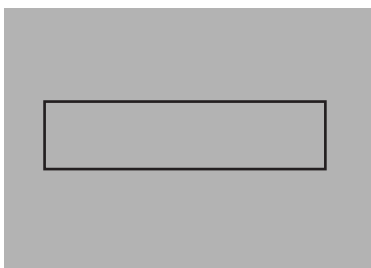


Рис. 16.6.

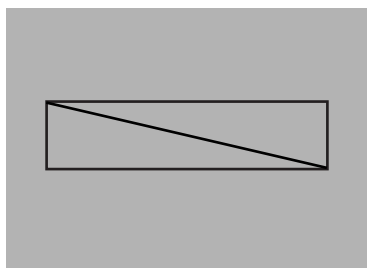


Рис. 16.7.

Рисование окружностей и эллипсов

Для рисования окружностей и эллипсов в РНР используется функция `imageellipse()`. Она отличается тем, что внешние ограничения фигуры не указываются. Эллипс описывается координатами центральной точки, а также его высотой и шириной. Рассмотрим синтаксис функции:

```
imageellipse(resource image, int x, int y, int width, int height, int col);
```

Ниже приведен пример вызова:

```
imageellipse($myImage, 90, 60, 160, 50, $myBlack);
```

Центр эллипса, показанного на рис. 16.8, имеет координаты $x = 90$ и $y = 60$ в пикселях. Ширина эллипса равна 160 пикселей, а высота 50 пикселей.

Чтобы нарисовать круг, необходимо описать эллипс, ширина которого равна высоте (рис. 16.9):

```
imageellipse($myImage, 90, 60, 70, 70, $myBlack);
```

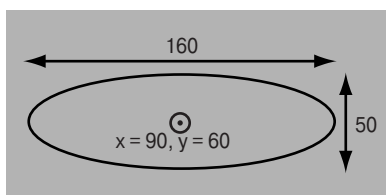


Рис. 16.8.

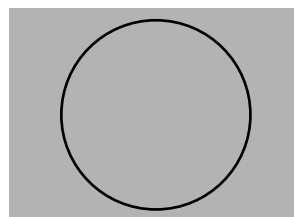


Рис. 16.9.

Рисование дуг

Дуга представляет собой часть эллипса, незамкнутую линию. Дуга описывается так же, как и эллипс за исключением того, что требуется задать дополнительные параметры, определяющие начало и конец дуги. Начальная и конечная точки задаются в градусах. В полном эллипсе 360 градусов. 0 градусов — крайняя правая точка эллипса (три часа на циферблате часов). Углы отсчитываются по часовой стрелке (рис. 16.10):

Ниже приведен синтаксис функции `imagearc()`:

```
imagearc(resource image, int x, int y, int width, int height, int start_degree, int end_degree, int col)
```

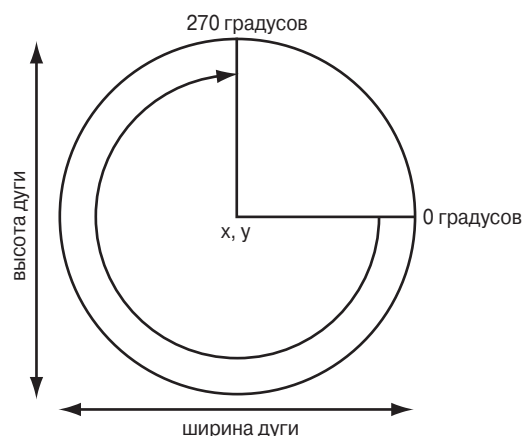



Рис. 16.10.

Рассмотрим пример рисования незамкнутого эллипса с помощью функции `imagearc()`:

```
imagearc($myImage, 90, 60, 160, 50, 45, 200, $myBlack);
```

Первый параметр (`$myImage`) идентифицирует создаваемое изображение. Два следующих параметра (90, 60) указывают центральную точку эллипса, по которой строится дуга. Параметры 160, 50 те же, что и в предыдущем примере эллипса. В действительности создают дугу два следующих параметра: значение 45 указывает функции `imagearc()`, что дуга должна начинаться в позиции с углом 45 градусов (примерно 4:22 на циферблате часов) и заканчиваться в позиции с углом 200 градусов. Следует отметить, что значение 200 — позиция в градусах от точки 0 градусов. На рис. 16.11 показана дуга 45–200 градусов.

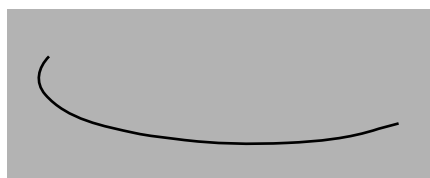


Рис. 16.11.

Дуга на рисунке может выглядеть странно, но надо помнить, что дуга рисуется по эллипсу, описанному указанной шириной и высотой. Сравните эту дугу с эллипсом, нарисованным ранее (см. рис. 16.8) с использованием тех же параметров ширины и высоты.

Рисование многоугольников

Многоугольник представляет собой фигуру, имеющую три или больше углов или вершин. Так как функция для рисования многоугольников (`imagepolygon()`) может создавать фигуры с разным количеством вершин, она принимает параметры, которые отличаются от параметров описанных выше функций. Кроме идентификатора изображения функции необходимо передать массив точек (координат x и y , определяющих

углы или вершины многоугольника). Второй параметр сообщает функции количество вершин создаваемого многоугольника.

```
int imagepolygon (resource image, array points, int num_points, int color)
```

Рассмотрим следующий код:

```
$myPoints = array(20,20,185,55,70,80);
imagepolygon($myImage,$myPoints,3,$myBlack);
```

Сначала необходимо создать массив точек. В этом примере массив содержит шесть элементов — три пары x - и y -координат. Это означает, что многоугольник имеет три угла: в точках 20, 20, 185, 55 и 70, 80.

Затем вызывается функция `imagepolygon()`, которой передаются следующие параметры:

1. Идентификатор изображения (ресурс).
2. Массив точек.
3. Количество вершин многоугольника.
4. Цвет линии, образующей создаваемую фигуру.

Полученный многоугольник показан на рис. 16.12.

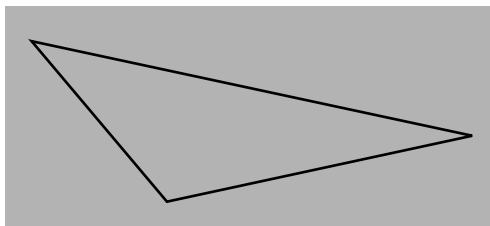


Рис. 16.12.

Практика Рисование прямоугольника с закругленными углами

1. Откройте текстовый редактор и введите следующий код:

```
<?php
function roundrect($image, $x1, $y1, $x2, $y2, $curvedepth, $color) {
    imageline($image, ($x1 + $curvedepth), $y1, ($x2 - $curvedepth), $y1, $color);
    imageline($image, ($x1 + $curvedepth), $y2, ($x2 - $curvedepth), $y2, $color);
    imageline($image, $x1, ($y1 + $curvedepth), $x1, ($y2 - $curvedepth), $color);
    imageline($image, $x2, ($y1 + $curvedepth), $x2, ($y2 - $curvedepth), $color);
    imagearc($image, ($x1 + $curvedepth), ($y1 + $curvedepth), (2 * $curvedepth),
    (2 * $curvedepth), 180, 270, $color);
    imagearc($image, ($x2 - $curvedepth), ($y1 + $curvedepth), (2 * $curvedepth),
    (2 * $curvedepth), 270, 360, $color);
    imagearc($image, ($x2 - $curvedepth), ($y2 - $curvedepth), (2 * $curvedepth),
    (2 * $curvedepth), 0, 90, $color);
    imagearc($image, ($x1 + $curvedepth), ($y2 - $curvedepth), (2 * $curvedepth),
    (2 * $curvedepth), 90, 180, $color);
}
$myImage = imagecreate(200,100);
$myGrey = imagecolorallocate($myImage,204,204,204);
$myBlack = imagecolorallocate($myImage,0,0,0);
```

```
roundrect($myImage, 20, 10, 180, 90, 20, $myBlack);
header("Content-type: image/png");
imagepng($myImage);
imagedestroy($myImage);
?>
```

2. Сохраните файл как `roundrect.php`.
3. Откройте только что созданный файл в Web-браузере. Вывод сценария показан на рис. 16.13.

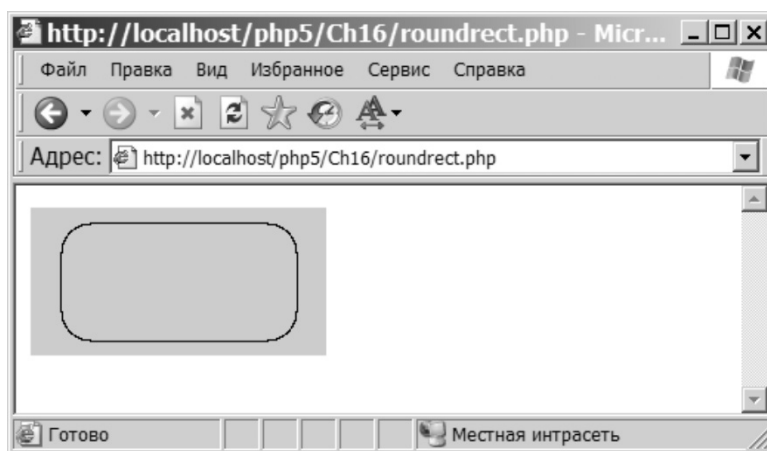


Рис. 16.13.

Как это работает

Рассмотрим код, создающий прямоугольник с закругленными углами. В сценарии нет неописанных выше функций. Нужно просто учитывать, что для создания прямоугольника с закругленными углами необходимо комбинировать прямые линии и дуги.

Сначала в сценарии создается функция для рисования прямоугольника с закругленными углами. Выделение этого кода в самостоятельную функцию позволит впоследствии использовать код повторно для рисования прямоугольников с другими параметрами. Функция принимает семь параметров:

```
<?php
function roundrect($image, $x1, $y1, $x2, $y2, $curvedepth, $color) {
```

Первый параметр — идентификатор изображения, в котором создается прямоугольник. Два следующих параметра определяют верхний левый угол прямоугольника (поскольку углы закруглены, в этой точке фактически ничего не рисуется, она используется как точка привязки для левого верхнего угла прямоугольника). Два следующих параметра определяют правый нижний угол прямоугольника. Шестой параметр, `$curvedepth`, представляет собой расстояние (в пикселях) от угла прямоугольника до начала кривой. Последний параметр — цвет линий прямоугольника. На рис. 16.14 показано, как используются переданные функции `roundrect()` параметры.

В зависимости от того, какие линии и углы рисуются, используются различные комбинации параметров `$x1`, `$x2`, `$y1`, `$y2` и `$curvedepth`.



Рис. 16.14.

Сначала рисуется верхняя горизонтальная линия прямоугольника. Эта линия не должна соединять точки с координатами $x1$ и $x2$, так как необходимо учесть закругление угла. Поэтому значение `$curvedepth` нужно прибавить к координате $x1$ и вычесть из координаты $x2$:

```
imageline($image, ($x1 + $curvedepth), $y1, ($x2 - $curvedepth), $y1, $color);
```

Поскольку эта линия горизонтальна, обе точки имеют одинаковую y -координату. Затем рисуется нижняя горизонтальная линия прямоугольника. В этом случае длину линии необходимо уменьшить на глубину кривой и использовать одинаковые y -координаты для обеих точек:

```
imageline($image, ($x1 + $curvedepth), $y2, ($x2 - $curvedepth), $y2, $color);
```

Далее рисуются две вертикальные линии, левая и правая стороны прямоугольника. На этот раз для двух точек каждой линии используется одна и та же x -координата ($x1$ для левой стороны и $x2$ для правой), а y -координаты соответствующим образом изменяются с учетом высоты кривой, образующей закругленные углы.

```
imageline($image, $x1, ($y1 + $curvedepth), $x1, ($y2 - $curvedepth), $color);
imageline($image, $x2, ($y1 + $curvedepth), $x2, ($y2 - $curvedepth), $color);
```

Первая создаваемая кривая образует верхний левый угол. Необходимо указать центральную точку дуги (рис. 16.15), поэтому значение `$curvedepth` прибавляется к значениям $x1$ и $y1$. Кроме того, указывается ширина и высота дуги. Так как глубина закругления фактически является радиусом дуги, чтобы получить ширину и высоту дуги, ее глубину необходимо удвоить.

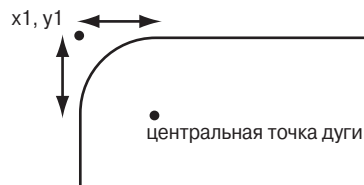


Рис. 16.15.

Дуга начинается в позиции 180 градусов (9 часов) и продолжается до позиции 270 градусов.

```
imagearc($image, ($x1 + $curvedepth), ($y1 + $curvedepth), (2 * $curvedepth),
(2 * $curvedepth), 180, 270, $color);
```

Остальные углы создаются так же, кроме того, что для получения корректных центральных точек для разных дуг нужно либо прибавлять глубину кривой, либо вычи-

тать ее из координат x и y . Не забывайте о том, что начальные и конечные позиции дуг изменяются для каждого угла.

```
imagearc($image, ($x2 - $curvedepth), ($y1 + $curvedepth), (2 * $curvedepth),  
(2 * $curvedepth), 270, 360, $color); imagearc($image, ($x2 - $curvedepth),  
($y2 - $curvedepth), (2 * $curvedepth), (2 * $curvedepth), 0, 90, $color);  
imagearc($image, ($x1 + $curvedepth), ($y2 - $curvedepth), (2 * $curvedepth),  
(2 * $curvedepth), 90, 180, $color);
```

Это все, что нужно для рисования прямоугольника с закругленными углами, поэтому далее следует закрыть функцию и можно приступить к созданию изображения.

```
}
```

Создается пустое изображение, и в его палитру добавляются два цвета. Первый добавляемый цвет будет фоновым цветом изображения:

```
$myImage = imagecreate(200,100);  
$myGrey = imagecolorallocate($myImage,204,204,204);  
$myBlack = imagecolorallocate($myImage,0,0,0);
```

Затем вызывается функция `roundrect()`, которой передаются описанные выше параметры:

```
roundrect($myImage, 20, 10, 180, 90, 20, $myBlack);
```

Чтобы отправить изображение Web-браузеру, сначала отправляется заголовок, сообщающий браузеру тип отправляемых данных. Это позволяет браузеру использовать корректную подпрограмму для отображения данных. В данном случае отправляется PNG-файл:

```
header("Content-type: image/png");
```

Затем используется функция `imagepng()`, которая отправляет данные самого изображения. Эта функция принимает один параметр, идентификатор изображения. В конце сценария используется функция `imagedestroy()` для очистки памяти.

```
imagepng($myImage);  
imagedestroy($myImage);  
?>
```

Изменение растровых изображений

Ранее в этой главе рассматривалось создание изображений с помощью функций рисования библиотеки GD и практические примеры рисования базовых фигур одного цвета. А как же быть с растровыми изображениями, состоящими не из одного простого цвета, а из миллионов точек разных цветов, например, с фотографиями? Библиотека GD позволяет не только создавать новые изображения с нуля, но и новые изображения на основе существующих JPEG-, PNG- или GIF-изображений.

Открытие существующего изображения

Для создания нового изображения используются функции `imagecreate()` и `imagecreatetruecolor()`. Чтобы создать новое изображение на основе существующего, применяют функции серии `imagecreatefrom`. Чаще всего используются функции `imagecreatefromjpeg()`, `imagecreatefromgif()` и `imagecreatefrompng()`. Существует также множество других функций, позволяющих создавать в памяти новые изображения на основе существующих, но они не так широко применяются, как три упомянутые выше функции.

Функция `imagecreatefromjpeg()` работает почти так же, как `imagecreate()`, за исключением того, что вместо передачи ширины и высоты нового изображения передается только строковый аргумент — имя файла существующего изображения. Функция возвращает идентификатор изображения, с которым затем можно работать.

Вызов

```
$myImage = imagecreatefromjpeg('moegie.jpg');
```

открывает JPEG-файл `moegie.jpg`, который находится в том же каталоге, что и исполняемый сценарий. Идентификатор изображения `$myImage` содержит данные JPEG-файла. Это можно проверить с помощью вывода данных изображения без каких-либо предварительных операций с ними.

Практика Вывод JPEG-изображения в окно браузера

1. Введите в текстовом редакторе следующий код:

```
<?php
$myImage = imagecreatefromjpeg('moegie.jpg');
header("Content-type: image/jpeg");
imagejpeg($myImage);
imagedestroy($myImage);
?>
```

2. Сохраните файл как `showjpeg.php`. Убедитесь, что имя файла, передаваемое в качестве параметра функции `imagecreatefromjpeg()`, соответствует существующему JPEG-файлу, который находится в том же каталоге, что и сценарий `showjpeg.php`.
3. Откройте страницу `showjpeg.php` в Web-браузере. Пример страницы показан на рис. 16.16.

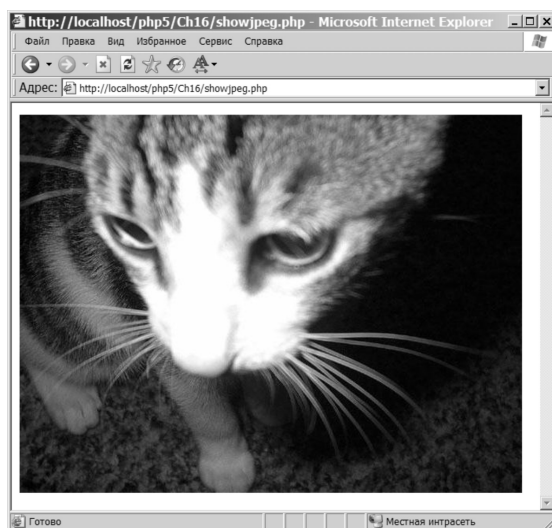


Рис. 16.16.

Как это работает

Код сравнительно прост и не должен создавать проблемы. Сначала используется функция `imagecreatefromjpeg()` для создания нового ресурса изображения из существующего файла:

```
$myImage = imagecreatefromjpeg('moegie.jpg');
```

Затем Web-браузеру отправляется заголовок, сообщающий о передаче данных изображения:

```
header("Content-type: image/jpeg");
```

Теперь все, что осталось сделать, это отправить данные изображения и удалить изображение из памяти:

```
imagejpeg($myImage);  
imagedestroy($myImage);
```

В представленном примере нет ничего, что невозможно было бы сделать с помощью простого HTML. Однако, возможность открывать существующие изображения и манипулировать ими до отправки в браузер полезна по многим причинам. Ниже перечислены некоторые возможности обработки изображений с помощью GD-функций.

- ☐ Изменение размеров изображений для создания пиктограмм или понижение качества изображений для ускорения их загрузки в браузер.
- ☐ Аннотирование изображения описательным текстом или подписью.
- ☐ Копирование части одного изображения в другое для создания водяных знаков.

Рассмотрим создание водяных знаков в изображении, а также создание пиктограмм из изображений.

Внедрение водяных знаков

При разработке Web-сайта, на котором демонстрируются оригинальные фотографии, может возникнуть необходимость защитить право интеллектуальной собственности владельцев этих фотографий. Часто для этого в изображение внедряют водяные знаки так, что после этого изображение остается прежним, но другие люди не имеют права его использовать как свое собственное. Ниже описан процесс внедрения водяных знаков в изображение.

Сначала создается простое изображение, знак авторского права (copyright-знак, рис. 16.17), которое можно затем использовать в других изображениях в качестве водяного знака. Поместим черный текст на белый фон и сохраним полученное изображение как PNG-файл с восьмицветной палитрой. (Далее будут указаны причины использования именно такого файла.)

©Allan Kent, 2004

Рис. 16.17.

Копирование copyright-знака в изображение

Теперь откроем исходное изображение, в которое нужно вставить водяной знак:

```
<?php  
$myImage = imagecreatefromjpeg('moegie.jpg');
```

Затем следует открыть copyright-знак или логотип. Так как это PNG-файл, для его открытия используется функция `imagecreatefrompng()`.

```
$myCopyright = imagecreatefrompng('copyright.png');
```

Можно разместить copyright-знак в верхнем левом углу изображения. В таком случае нет необходимости в выполнении следующих этапов. Однако если требуется расположить его справа, снизу или в центре изображения, то необходимо определить размеры обоих изображений. Функция `imagesx()` возвращает ширину изображения, а `imagesy()` — высоту. Обе функции принимают один параметр — ресурс, идентификатор изображения, размеры которого требуется определить. В рассматриваемом здесь примере copyright-знак помещается в центре изображения, поэтому необходимо определить ширину и высоту исходного изображения и copyright-знака:

```
$destWidth = imagesx($myImage);
$destHeight = imagesy($myImage);
$srcWidth = imagesx($myCopyright);
$srcHeight = imagesy($myCopyright);
```

Далее нужно вычислить координаты левого верхнего угла copyright-знака в изображении. Эта задача решается простым вычитанием ширины copyright-знака из ширины исходного изображения и последующим делением полученной разности на два. Для того чтобы получить y-координату, следует выполнить те же вычисления, но используя высоту изображений:

```
$destX = ($destWidth - $srcWidth) / 2;
$destY = ($destHeight - $srcHeight) / 2;
```

Если бы copyright-знак нужно было бы поместить около правой границы изображения, то для получения x-координаты левого верхнего угла copyright-знака пришлось бы просто отнять его ширину от ширины исходного изображения.

Получив координаты для вставки copyright-знака, можно скопировать его в исходное изображение. Для этого используется функция `imagecopy()`:

```
imagecopy($myImage, $myCopyright, $destX, $destY, 0, 0, $srcWidth, $srcHeight);
```

Функция принимает восемь параметров, сообщающих ей, что копировать из исходного изображения и куда помещать скопированный фрагмент в конечном изображении. Первым параметром является идентификатор изображения, куда будут скопированы данные, т.е. идентификатор того изображения, которое необходимо защитить водяным знаком. Вторым параметром — идентификатор изображения, из которого будут скопированы данные, т.е. copyright-знака. Третий и четвертый параметры — x и y-координаты в конечном изображении, куда требуется скопировать данные. Эти координаты обозначают левый верхний угол блока копируемых данных. Два следующих параметра (0, 0 в примере) — x и y-координаты в исходном изображении, с которых начинается копируемый блок данных. Функция использует эти два параметра, а также два следующих параметра (ширину и высоту копируемого блока) для определения блока копируемых данных в исходном изображении. В данном случае копируется все изображение, поэтому используются координаты 0, 0, а ширина и высота равны ширине и высоте copyright-знака.

После того как данные скопированы, можно как обычно выводить изображение в браузер. Затем следует очистить память, занимаемую обоими изображениями:

```
header("Content-type: image/jpeg");
imagejpeg($myImage);
imagedestroy($myImage);
```



```
imagedestroy($myCopyright);  
?>
```

На рис. 16.18 показан результат создания водяного знака в изображении.



Рис. 16.18.

Одно из требований к водяному знаку состоит в том, что первоначальное изображение должно оставаться узнаваемым. На рисунке часть изображения теперь закрыта. Необходимо улучшить сценарий так, чтобы copyright-знак закрывал как можно меньшую часть изображения.

Использование прозрачности

Можно скопировать не все изображение, а только текст. Для этого нужно сделать белую область copyright-знака прозрачной. Необходимая для этого функция работает с идентификаторами цветов, поэтому прежде чем сделать область прозрачной, следует получить индекс белого цвета. Для этого существует несколько способов — можно использовать функцию `imagecolorat()`, которая возвращает индекс цвета для точно указанного пикселя в изображении, или точно указать цвет и попытаться получить индекс этого цвета в палитре. Единственный недостаток последнего способа заключается в том, что если указанный цвет отсутствует в палитре изображения, то получить корректный идентификатор изображения не удастся. Ранее copyright-знак был сохранен как 8-цветное PNG-изображение. Это было сделано потому, что в работе использовалась только небольшая палитра и можно было уверенно полагать, что белый фон изображения будет одинаковым во всем изображении. Если бы изображение было сохранено как JPEG с миллионами цветов, то сглаживание в изображении могло бы привести к появлению некоторых вариаций фонового цвета, что усложнило бы

точное указание белого цвета (который необходимо сделать прозрачным). (Сглаживание цветов означает метод, при котором цвет формируется путем помещения рядом двух пикселей разных цветов.) Сохраняя изображение в формате PNG с небольшим количеством цветов, можно избежать эффекта сглаживания.

Для возвращения индекса цвета (в данном случае белого) используется функция `imagecolorexact()`. (Ее синтаксис описывается в приложении Б, “Справочник по PHP-функциям”.) Функция `imagecolorexact()` принимает четыре параметра — идентификатор ресурса изображения и значения красного, зеленого и синего для цвета, который необходимо найти. Поскольку в данном случае изображение имеет только восемь цветов, можно быть уверенным, что будет возвращен корректный индекс цвета.

После получения индекса цвета используется функция `imagecolortransparent()`, которая делает указанный цвет в изображении прозрачным. Эта функция принимает два параметра: идентификатор ресурса изображения и индекс цвета, который необходимо сделать прозрачным. Показанные ниже выделенные строки кода вставляются в первоначальную часть сценария — они следуют сразу за вычислением конечных координат x и y и перед копированием данных из одного изображения в другое.

```
$destY = ($destHeight - $srcHeight) / 2;
```

```
$white = imagecolorexact($myCopyright, 255, 255, 255);
imagecolortransparent($myCopyright, $white);
```

```
imagecopy($myImage, $myCopyright, $destX, $destY, 0, 0, $srcWidth, $srcHeight);
```

Теперь вывод сценария выглядит корректнее (рис. 16.19).



Рис. 16.19.

Белесый контур вокруг цифр 2004 в изображении связан с эффектом сглаживания (anti-aliasing) шрифта. Если такой эффект нежелателен, то можно отключить сглаживание шрифта в графическом редакторе либо сохранить copyright-знак как двухцветное изображение в формате PNG. Как и следовало ожидать, очень сложно получить гладкую кривую в небольшом пиксельном диапазоне, который характерен для символов шрифта. При сглаживании шрифта используются оттенки цвета шрифта для создания плавного перехода кривых шрифта в фон изображения, чтобы буквы не имели зазубренных краев. Это означает, что закругления шрифта состояются из различных оттенков цвета, и когда фоновый цвет изображения становится прозрачным, последствия сглаживания шрифта оказываются видимыми (см. рис. 16.19). Выключение сглаживания или сохранение изображения как двухцветного делает шрифт немного неровным, но вместе с тем удаляет контур вокруг цифр 2004. Чтобы уменьшить этот эффект и сделать водяной знак менее бросающимся в глаза, можно было бы изменить непрозрачность копируемых в изображение данных.

Использование непрозрачности

Степень *непрозрачности* (*opacity*) изображения может изменяться от полностью прозрачного (через изображение видно другое изображение) до полностью непрозрачного, когда сквозь одно изображение другого изображения не видно. В предыдущем разделе черный текст copyright-знака был непроницаемым, хотя фон был прозрачным. Чтобы водяной знак не так бросался в глаза, можно использовать функцию `imagecopymerge()`, которая позволяет придать копируемым данным некоторую степень прозрачности. Эта функция работает аналогично `imagecopy()` кроме того, что она принимает девятый параметр, который управляет прозрачностью копируемого изображения. Если значение этого параметра равно 0, то цвет будет полностью прозрачным и копируемые данные будут не видны, а значение 100 приводит к полной непрозрачности копируемого изображения, и в этом случае данная функция ведет себя как `imagecopy()`. Изменим следующую строку сценария:

```
imagecopy($myImage, $myCopyright, $destX, $destY, 0, 0, $srcWidth, $srcHeight);
```

на

```
imagecopymerge($myImage, $myCopyright, $destX, $destY, 0, 0, $srcWidth, $srcHeight, 50);
```

Функция `imagecopy()` заменена `imagecopymerge()` и указан дополнительный параметр со значением 50 — среднее значение между прозрачностью и непрозрачностью. Вывод сценария теперь намного больше похож на водяной знак (рис. 16.20).

Читателям, которые интересуются эффектами прозрачности в изображениях, можно порекомендовать ознакомиться с описанием функций `imagecolorallocatealpha()` и `imagealphablending()` в руководстве по PHP. И хотя воспроизвести в PHP все возможности, доступные для профессиональной программы редактирования графики, не удастся, некоторые интересные эффекты все-таки получить можно.

В следующем разделе рассматривается изменение существующих изображений, но на этот раз речь пойдет об уменьшении размеров изображения до пиктограммы.

Создание пиктограмм

При создании пиктограммы изображения используется метод, аналогичный внедрению водяного знака, за исключением того, что копирование осуществляется в другом направлении. Вместо копирования меньшего изображения в большее, большее изображение копируется в новое, пропорционально уменьшенное изображение.



Рис. 16.20.

1. Создайте файл `thumbnail.php`, выберите изображение, сохраните его как `moegie.jpg` и следуйте описанным далее этапам создания пиктограммы.
2. Откройте изображение, для которого планируется создать пиктограмму:

```
<?php
$mainImage = imagecreatefromjpeg('moegie.jpg');
```

3. Используйте функции `imagesx()` и `imagesy()` для определения ширины и высоты исходного изображения. Эти данные необходимы для вычисления размера нового изображения (пиктограммы):

```
$mainWidth = imagesx($mainImage);
$mainHeight = imagesy($mainImage);
```

4. В этом примере создается пиктограмма, размеры которой в четыре раза меньше размеров исходного изображения, поэтому исходную ширину и высоту нужно разделить на 4. В результате должны получиться целые числа, потому что невозможно использовать половину пикселя:

```
$thumbWidth = intval($mainWidth / 4);
$thumbHeight = intval($mainHeight / 4);
```

Необязательно масштабировать изображение таким способом. Можно уменьшать изображение до определенных размеров, так чтобы ширина и высота новой пиктограммы попадали в заданный диапазон. Или можно сделать так, чтобы все пиктограммы имели одинаковую ширину, и масштабировать их соответствующим образом.

Важно то, что ширина и высота должны изменяться пропорционально, чтобы пиктограмма не получилась “сплюсненной”.

5. Создайте новое пустое изображение, имеющее ширину и высоту пиктограммы. Как правило, пиктограммы создаются для изображений фотографического типа, поэтому пиктограмма должна иметь большое количество цветов; для ее создания используется функция `imagecreatetruecolor()`:

```
$myThumbnail = imagecreatetruecolor($thumbWidth, $thumbHeight);
```

Теперь следует уменьшить исходное изображение и скопировать его в новую пиктограмму. Это можно сделать с помощью одной из двух функций — `imagecopyresized()` или `imagecopyresampled()`. Разница между ними заключается в том, что `imagecopyresized()` работает быстрее, но совсем не сглаживает изображение. Если с помощью этой функции создать пиктограмму, а затем увеличить ее, то будет замечен эффект мозаичности изображения (рис. 16.21).



Рис. 16.21.

Функция `imagecopyresampled()` хотя и работает медленнее, интерполирует пиксели так, что изображение не искажается. Обе функции: как `imagecopyresized()`, так и `imagecopyresampled()` — принимают одинаковый набор из 10 параметров. Первые два параметра — идентификатор конечного изображения и идентификатор изображения, из которого копируются данные. Вторая пара параметров — x и y координаты левого верхнего угла блока данных в конечном изображении, в который вставляются данные копируемого изображения. В этом примере используются координаты 0, 0, т.е. пиктограмма будет полностью заполнена изображением. Следующие два параметра сообщают функции, откуда начинать копирование данных исходного изображения. В рассматриваемом примере используются все данные, поэтому копирование начинается с левого верхнего угла в позиции 0, 0. Два следующих параметра сообщают функции, как изменится ширина и высота блока данных в конечном

изображении. Значения этих параметров должны быть вычислены заранее — ширина и высота пиктограммы. Два последних параметра определяют ширину и высоту блока графических данных, которые копируются из исходного изображения. В этом примере используется ширина и высота исходного изображения, поскольку в пиктограмму копируется все изображение.

Копировать все изображение необязательно, можно скопировать только небольшую его часть. Так как в конечном изображении разрешается устанавливать любую ширину или высоту блока данных, можно случайно скопировать блок непропорциональных размеров, поэтому нужно тщательно вычислять ширину и высоту изображений.

6. Для копирования данных изображения в пиктограмму используется функция `imagecopyresampled()`:

```
imagecopyresampled($myThumbnail, $mainImage, 0, 0, 0, 0, $thumbWidth,
    $thumbHeight, $mainWidth, $mainHeight);
```

7. Изображение отправляется браузеру (рис. 16.22), а память, используемая сценарием, очищается:

```
header("Content-type: image/jpeg");
imagejpeg($myThumbnail);
imagedestroy($myThumbnail);
imagedestroy($mainImage);
?>
```

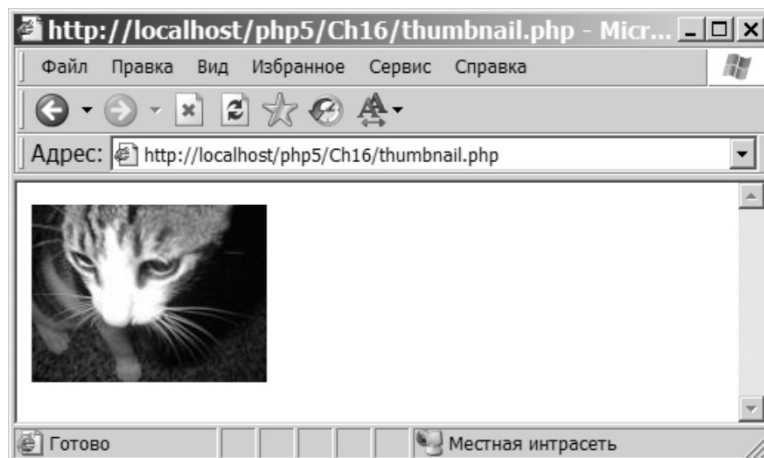


Рис. 16.22.

Использование текста в изображениях

Возможность добавлять с помощью РНР-функций текст в изображения позволяет аннотировать изображения или создавать динамические диаграммы и графики. Самый простой и самый быстрый способ добавления текста в изображение заключается в использовании функции `imagestring()`, которая позволяет вставить строку текста в указанную область изображения.

Добавление стандартного текста

Функция `imagestring()` облегчает вставку текста в изображение, потому что она может использовать встроенные системные шрифты и не вынуждает пользователя загружать на сервер шрифты в подходящем формате.

Ниже показан синтаксис функции `imagestring()`:

```
imagestring (image, font, x, y, text, color)
```

Как обычно, в качестве первого параметра передается идентификатор изображения. Второй параметр определяет используемый шрифт. Значением этого параметра является целое число больше нуля. Значения от 1 до 5 представляют встроенные системные шрифты, а любые шрифты, загруженные позднее, представлены значениями больше 5.

Чтобы загрузить шрифт для функции `imagestring()`, применяется функция `imageloadfont()`. Она загружает растровый зависимый от архитектуры шрифт — это означает, что шрифт должен быть сгенерирован на системе такого же типа, что и серверная система, на которой шрифт будет использоваться. Гораздо проще использовать шрифты True Type — этот метод рассматривается далее.

Параметры `x` и `y` позволяют позиционировать шрифт в изображении. Эти значения представляют собой координаты левого верхнего угла прямоугольника, в котором будет расположен текст. Два последних параметра — собственно текст, который должен появиться в изображении, и цвет, которым он будет нарисован.

Практика Отображение системных шрифтов

Следующий сценарий (`drawstring.php`) демонстрирует внешний вид системных шрифтов.

```
<?php
$textImage = imagecreate(200,100);
$white = imagecolorallocate($textImage, 255, 255, 255);
$black = imagecolorallocate($textImage, 0, 0, 0);
$yOffset = 0;
for ($i = 1; $i <=5; $i++) {
    imagestring($textImage, $i, 5, $yOffset, "This is system font $i", $black);
    $yOffset += imagefontheight($i);
}
header("Content-type: image/png");
imagepng($textImage);
imagedestroy($textImage);
?>
```

На рис. 16.23 показан результат работы этого сценария.

Как это работает

Сначала создается пустое изображение для текста, а затем добавляются белый и черный цвет. Так как белый цвет добавляется первым, он будет фоновым цветом изображения.

```
<?php
$textImage = imagecreate(200,100);
$white = imagecolorallocate($textImage, 255, 255, 255);
$black = imagecolorallocate($textImage, 0, 0, 0);
```

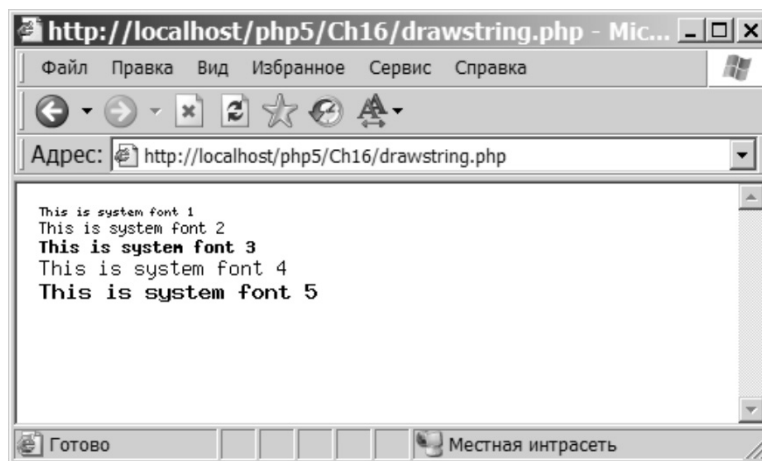


Рис. 16.23.

Затем инициализируется переменная для хранения у-координаты (расстояние от верхнего края изображения) строки текста. В рассматриваемом примере текст размещается вверху изображения, поэтому значение переменной равно 0.

```
$yOffset = 0;
```

Далее в цикле просматриваются все встроенные системные шрифты.

```
for ($i = 1; $i <= 5; $i++) {
```

В каждой итерации цикла создается строка с использованием шрифта с идентификатором `$i`, который изменяется в цикле от 1 до 5. Строка позиционируется на расстоянии 5 пикселей от левой границы изображения и на расстоянии `$yOffset` от верхней границы.

```
imagestring($textImage, $i, 5, $yOffset, "Это системный шрифт $i", $black);
```

Функция `imagefontheight()` принимает идентификатор шрифта и возвращает высоту шрифта в пикселях. Возвращенное этой функцией значение прибавляется к `$yOffset` для того, чтобы строки текста выводились одна под другой и не накладывались друг на друга. (Ширину шрифта можно получить с помощью функции `imagefontwidth()`.)

```
$yOffset += imagefontheight($i);
```

Наконец, изображение отправляется браузеру и выполняется очистка памяти.

```
}
header("Content-type: image/png");
imagepng($textImage);
imagedestroy($textImage);
?>
```

Использование шрифтов True Type

При создании простейших диаграмм и графиков предпочтительнее использовать встроенные системные шрифты, поскольку они моноширинные — все символы шрифта имеют одинаковую ширину — и это облегчает позиционирование. Однако чтобы сделать текст более изящным, следует использовать шрифты True Type.

GD-библиотека для Windows-инсталляции PHP поддерживает шрифты True Type. В Unix пользователям перед компиляцией PHP необходимо установить библиотеку FreeType. Библиотека FreeType доступна на сайте www.freetype.org.

Результатом использования шрифтов True Type является большая гибкость — можно не только контролировать внешний вид текста, выбирая подходящий шрифт, но и указывать размер шрифта и угол наклона строки текста. Для этого предпочтительнее применять функцию `imagefttext()`, которая использует библиотеку FreeType 2 для рисования True Type-текста. Эта функция принимает параметры, аналогичные параметрам функции `imagestring()`, но в другом порядке, и некоторые параметры используются иначе.

Практика Шрифты True Type

Рассмотрим сценарий `truetype.php`:

```
<?php
$textImage = imagecreate(200,100);
$white = imagecolorallocate($textImage, 255, 255, 255);
$black = imagecolorallocate($textImage, 0, 0, 0);
imagefttext($textImage, 16, 0, 10, 50, $black, "C:\Windows\fonts\arial.ttf",
"Arial, 16 pixels");
header("Content-type: image/png");
imagepng($textImage);
imagedestroy($textImage);
?>
```

На рис. 16.24 показан текст, написанный шрифтом Arial размером 16 пикселей. Значение 0 указывает на то, что текст рисуется слева направо под углом 0 градусов (т.е. горизонтально).

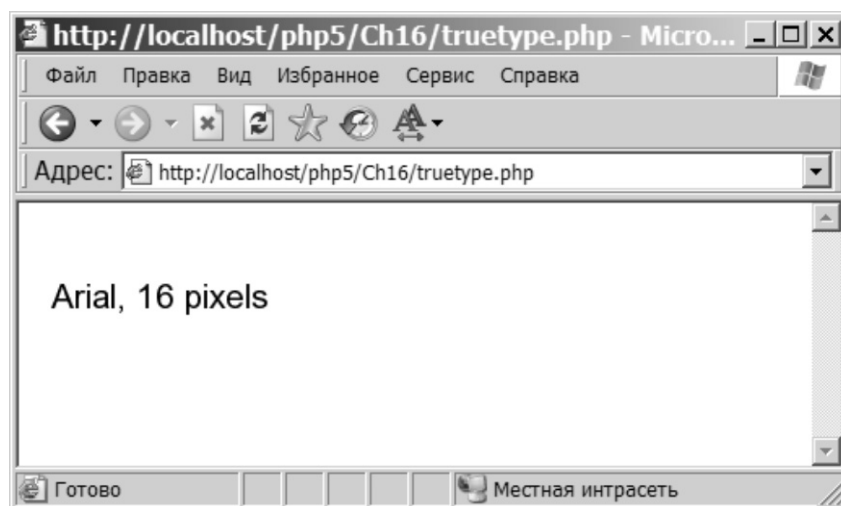


Рис. 16.24.

Как это работает

Сначала создается пустое изображение и выделяется два цвета.

```
<?php
$textImage = imagecreate(200,100);
$white = imagecolorallocate($textImage, 255, 255, 255);
$black = imagecolorallocate($textImage, 0, 0, 0);
```

Затем в изображение вставляется строка текста:

```
imagefttext($textImage, 16, 0, 10, 50, $black, "C:\Windows\fonts\arial.ttf",
"Arial, 16 pixels");
```

Первый параметр — идентификатор изображения, которое необходимо изменить. Второй параметр определяет размер шрифта — высоту шрифта в пикселях. Следующий параметр определяет угол наклона текста. Угол в данном случае отсчитывается так же, как и в функции `imagearc()`, но угол определяет направление написания текста. Так как позиция 0 градусов соответствует 3 часам на циферблате часов, текст рисуется в направлении слева направо. Эта функция значительно отличается от `imagearc()` тем, что вращение выполняется против часовой стрелки (пример вращения показан ниже). Следующие два параметра определяют *x*- и *y*-координаты начала строки — левый нижний угол ограничивающего блока вокруг текста; это отличается от функции `imagestring()`, в которой координаты отсчитываются от левого верхнего угла ограничивающего блока. Затем необходимо указать полный системный путь к файлу шрифта (TTF) на жестком диске. Последний параметр — строка, которую необходимо вставить в изображение.

Работа сценария завершается как обычно:

```
header("Content-type: image/png");
imagepng($textImage);
imagedestroy($textImage);
?>
```

Чтобы изобразить повернутый текст, необходимо заменить строку:

```
imagefttext($textImage, 16, 0, 10, 50, $black, "C:\Windows\fonts\arial.ttf",
"Arial, 16 pixels");
```

на:

```
imagefttext($textImage, 16, -30, 10, 30, $black, "C:\Windows\fonts\arial.ttf",
"Arial, 16 pixels");
```

Изменилось направление рисования текста: -30 градусов вместо 0. Как уже было сказано, вращение в этой функции выполняется против часовой стрелки, поэтому отрицательное число соответствует повороту текста по часовой стрелке. Если 0 градусов совпадает с 3 часами, то 30 градусов по часовой стрелке совпадает с 4 часами на циферблате. На рис. 16.25 отчетливо видно, как наклонен текст.

Резюме

В этой главе рассматривалось создание и вывод изображений с помощью PHP. В примерах было показано, как комбинировать два изображения с целью вставки водяного знака, как открывать и использовать существующие изображения для создания пиктограмм. Полученные знания помогут разработчикам расширить свои сценарии и Web-страницы.

Кроме того, в главе освещалась работа PHP-функций с цветами, был также описан процесс рисования различных фигур и текста в изображениях.

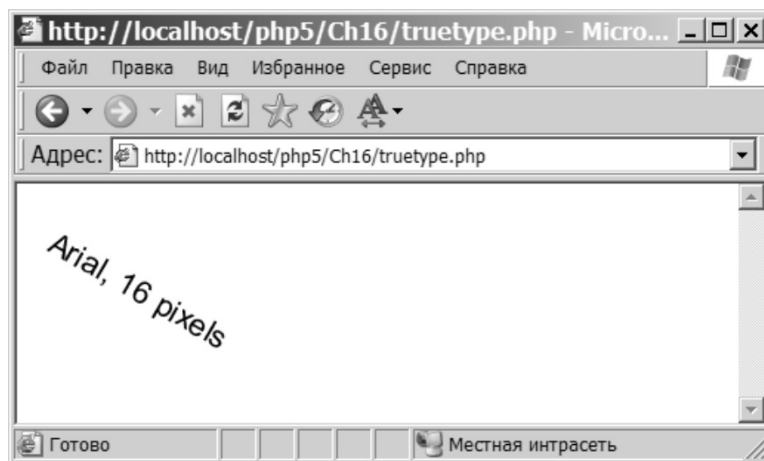


Рис. 16.25.

Упражнения

Следующие упражнения помогут читателям попрактиковаться в использовании рассмотренных выше методик и применить их в реальных разработках. Решения для этих упражнений приведены в Приложении А.

1. Создайте PHP-сценарий, который открывает файл изображения и добавляет в изображение черную рамку толщиной в один пиксель.
2. Используя функции `disk_total_space()` и `disk_free_space()`, покажите графическим способом объем используемого пространства на жестком диске.

17

Учебный пример: диспетчер протоколирования на PHP

К этому времени читатель имеет хорошие теоретические знания о PHP-программировании. В предыдущих главах рассматривались различные темы и понятия, включая хорошие методики программирования, объектно-ориентированное программирование, PEAR и основы программирования, такие как условные операторы, типы данных, область видимости и структура кода. В этой, последней главе рассмотренные аспекты PHP-программирования связываются в единое целое для создания реального приложения.

Конечно, создание крупного приложения не означает только написание огромного количества кода. Как уже отмечалось, создание высококачественного программного обеспечения предполагает большой объем планирования и всестороннее обдумывание задачи. В этой главе представлен учебный пример в формате цикла разработки программного обеспечения. В дополнение к этому в главе предлагается введение в шаблоны Smarty, описаны возможности и методики их применения.

Сначала разработчик рассматривает приложение в понятиях постановки задачи и анализирует возможности ее решения. После того как выбран подходящий метод реализации решения, начинается разработка программы. В ходе разработки придется определять, как создаваемые классы и объекты будут работать вместе и формировать решение. В главе 13 рассматривалось использование языка UML. В этой главе на стадии проектирования приложения также используется UML.

В главе изучаются жизненно важные вопросы реализации кода и исследуется весь код, который используется для выполнения необходимых задач. Полный листинг кода в главе не приводится, поскольку он был бы чрезмерно длинным, но все работающее приложение можно загрузить с Web-сайта Wrox.

Прежде чем переносить приложение в реальную среду, необходимо тщательно протестировать код. В этой главе представлено всестороннее описание комплекта тестов, которые можно использовать для оценки готовности приложения к реальному использованию. После того как разработчик убедится, что отдельные фрагменты кода работают устойчиво, можно исследовать работу приложения в целом. В этой главе читателю также предлагаются некоторые улучшения, которые можно реализовать в случае, если код будет использоваться как основа для других приложений.

Итак, приступим к изучению учебного примера.

Почему именно диспетчер протоколирования?

В наши дни многие компании и даже частные лица имеют по несколько Web-сайтов. Например, многие издательства создают отдельные Web-сайты для каждого из своих изданий. Причины у всех разные, но важно то, что компании часто считают обязательным накапливать информацию, полученную от нескольких Web-сайтов. Например, если издательство предлагает зарегистрированным пользователям всех сайтов скидку на все свои книги, то маркетинговому отделу для формирования бюджета, вероятно, понадобится список всех пользователей, которым предоставляется эта скидка.

Существует множество разных мнений относительно типов технологий, которые следует применять для создания Web-сайтов. Поэтому весьма вероятно, что информация, хранящаяся в базах данных или серверных журналах каждого из этих сайтов, значительно отличается форматом или даже на более низком уровне — типом данных. Попытки извлечь регистрационные данные, просматривая серверные журналы или записи баз данных, на практике могут потребовать больших затрат времени.

Даже если предположить, что кому-нибудь действительно понадобилось решить эту задачу, то одного только получения всей информации, вероятно, было бы недостаточно для отдела маркетинга. Сотрудники, которым понадобилась эта информация, скорее всего хотели бы получить ее в унифицированном читабельном формате. Получение журналов сервера, смешанных с записями из нескольких разных баз данных, могло бы сбивать с толку, потому что в отчетах, разбросанных по разным таблицам, была бы посторонняя информация. Форматирование всех этих данных вручную также потребовало бы значительного объема работы.

Еще хуже, что такую работу, возможно, придется выполнять каждые несколько месяцев, чтобы согласовывать данные о новых книгах. Повторяющаяся работа такого рода определенно должна быть автоматизирована, но как все-таки решить проблему?

Короткий ответ — протоколировать информацию, полученную от каждого посетителя на каждом Web-сайте непосредственно во время ввода. Это, несомненно, поможет систематизировать необходимую информацию, и на каждом сайте можно было бы гарантировать, что информация накапливается в стандартном формате — например, в CSV-файлах. Это не самое плохое решение, но оно все равно предполагает посещение каждого сайта для сбора необходимой информации. Но даже в этом случае CSV-файлы не самые удобные для чтения, и маркетологи, вероятно, были бы недовольны, если бы вся нужная им информация предоставлялась в виде одного массивного CSV-файла.

Кроме того, возникает риск дублирования больших объемов информации, потому что один человек может быть зарегистрирован на нескольких сайтах. Попытки проверять каждого пользователя по журналам и базам данных других сайтов перед вставкой

записи выглядят просто нелепо. Поэтому, скорее всего, решение будет связано с необходимостью создания центрального сервера для протоколирования информации — это в частности означает, что для вставки новой записи потребовалось бы сделать всего одну проверку.

Предположим, решение принято: требуется приложение, принимающее информацию от различных Web-сайтов, протоколирующее, а затем трансформирующее ее в удобочитаемый формат. Решение проблемы форматирования еще не рассматривалось в книге, но те читатели, которые использовали Smarty-шаблоны, скорее всего, будут уверены, что это идеальная ситуация для использования шаблонов с целью вывода информации в удобной и согласованной форме. И они будут правы. Разрабатываемое приложение позволит опрашивать центральный сервер с базой данных, которая в данном случае реализована на основе SQLite. Затем приложение будет выдавать результаты в нескольких предопределенных форматах — в этой главе для иллюстрации основных принципов рассматривается только два из них, но отчеты можно расширить с тем, чтобы представлять настолько подробные данные, насколько это необходимо. Следует помнить о том, что Smarty также можно использовать и в других целях (такие возможности рассматриваются в этой главе далее).

Часть учебного примера представляет собой краткое введение в систему Smarty, так как некоторые из читателей, возможно, не сталкивались с ней раньше. Существенно облегчает тестирование приложения использование пакета PHPUnit, поэтому в главе есть также небольшой раздел, посвященный этому продукту. Читатели, имеющие хорошие практические навыки использования Smarty и PHPUnit, могут смело пропустить эти разделы.

Smarty

Почему в этой книге рассматривается Smarty? Тому, кто работал над Web-приложениями как Web-дизайнер, либо как программист, возможно, приходилось модифицировать свой участок кода всякий раз после того, как другой участник проекта изменял свой. Например, разработчик получает HTML-страницы для определенного сайта и в течение недели добавляет PHP-код на уровне представления, чтобы пользователи могли выполнять поиск в таблицах базы данных или отправлять информацию (в зависимости от поставленных перед сайтом задач).

Неделю спустя клиент решает изменить внешний вид сайта. Дизайнер страниц раздражен, но переделывает дизайн согласно новым требованиям. Затем он уходит в недельный отпуск, оставляя программисту исправлять ошибки красивого, но совершенно не работающего Web-сайта. Так как весь HTML-код был модифицирован, ни один блок встроеного в Web-страницы PHP-кода не работает так, как должен, поэтому остается только просматривать код строка за строкой и восстанавливать работоспособность приложения.

Гораздо лучше было бы отделить логику представления от бизнес-логики. И здесь система Smarty оказывается весьма полезной.

Установка Smarty

В этом учебном примере будет использоваться система Smarty, поэтому ее необходимо установить. После установки следует убедиться в работоспособности Smarty, запустив небольшой тест. Затем можно приступить к работе над учебным примером.

Ниже описаны этапы установки Smarty.

1. Загрузите систему шаблонов Smarty с Web-сайта <http://smarty.php.net>.
2. Распакуйте архив в любой каталог.
3. Убедитесь, что PHP может использовать классы, предоставляемые Smarty. Эти классы расположены в каталоге `libs` инсталляции Smarty. Добавьте абсолютный путь к этому каталогу в переменную `include_path` в файле `php.ini`:

```
include_path = ".;c:\php5\includes;c:\php5\pear;C:\Program Files\
Smarty-2.6.2\libs"
```

4. Естественно, Smarty-классы можно рассматривать как любые другие PHP-классы и подключать их по отдельности в `.php`-файлы, например:

```
require('C:\Program Files\Smarty-2.6.2\libs\Smarty.class.php');
$smarty = new Smarty;
```

Используя Smarty, необходимо учитывать некоторые обстоятельства. Для работы Smarty требуется наличие четырех каталогов, которые (по умолчанию) называются `templates`, `templates_c`, `configs` и `cache`. Эти имена можно явно определить в файле `Smarty.class.php` — каждое из них содержится в соответствующей переменной: `$template_dir`, `$compile_dir`, `$config_dir`, `$cache_dir`. Убедитесь, что Web-сервер имеет права на запись в каталог `$compile_dir` (`templates_c` по умолчанию).

Документация рекомендует создавать отдельную группу таких каталогов для каждого приложения, использующего Smarty.

С целью обеспечения безопасности сайтов рекомендуется располагать эти каталоги за пределами корневого каталога Web-сайта. Web-браузеры ни при каких обстоятельствах не должны получать доступ к этим каталогам.

Следующий краткий пример позволяет убедиться, что Smarty работает корректно.

Практика Использование Smarty

В этом примере создается шаблон, позволяющий представить полученную из MySQL-таблицы информацию в виде аккуратно отформатированного списка. Для этого создается один `.php`-файл, ответственный за создание подключения к базе данных, выполнение запросов и возвращение результатов в Smarty-шаблон. Затем шаблон форматирует информацию, полученную данным PHP-сценарием, с помощью одной из встроенных в Smarty функций.

1. Сохраните следующий код в файле с именем `index.php`. Для работы сценария необходим доступ к таблице `user`, которая упоминалась в главах 11–13, либо можно создать отдельную таблицу специально для этого сценария:

```
<?php
require 'Smarty.class.php';
$smarty = new Smarty;

$hostname = "localhost";
$db_user = "phpuser";
$db_pass = "phppass";
$db_name = "sample_db";

// подключение к базе данных
$conn = mysql_connect($hostname, $db_user, $db_pass);
if (!$conn){
    die ("Не удалось подключиться к базе данных!");
```

```

}

mysql_select_db($db_name);

$sql = "SELECT DISTINCT userposition FROM user";

$res = mysql_query($sql);
$results = array();
$i=0;
while ($pos=mysql_fetch_row($res)) {
    $results[$i++] = $pos[0];
}

$smarty->assign('results', $results);
$smarty->display('index.tpl');
?>

```

2. Теперь создайте новый файл, назовите его `index.tpl` и сохраните в нем следующий код:

```

<HTML>
<BODY>
<B>Перечень позиций игроков: </B><BR><BR>
{section name=position loop=$results}
{$results[position]}<BR>
{/section}
</BODY>
</HTML>

```

3. Вызовите в Web-браузере страницу `index.php`; результат должен быть похож на рис. 17.1.

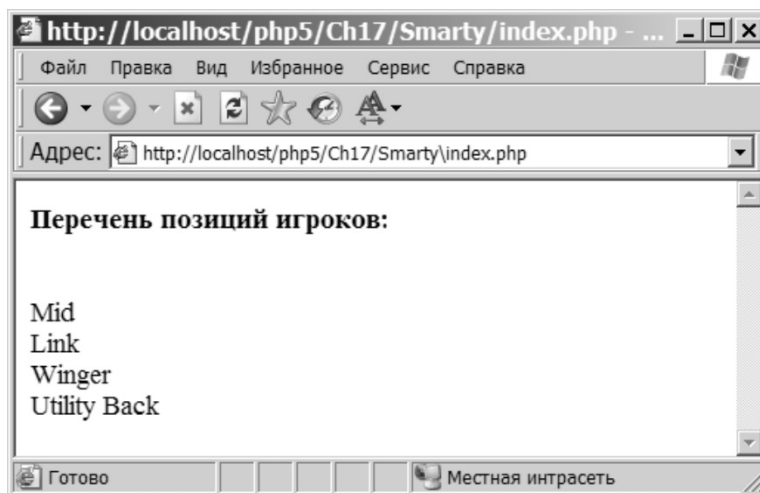


Рис. 17.1.

Как это работает

PHP-сценарий начинается с подключения класса `Smarty` и создания нового `Smarty`-объекта:

```

<?php
require 'Smarty.class.php';
$smarty = new Smarty;

```


В последующем коде очень немного нововведений. Создается MySQL-подключение, а если подключение невозможно, то обеспечивается завершение работы программы с выдачей сообщения об ошибке:

```
$hostname = "localhost";
$db_user = "phpuser";
$db_pass = "phppass";
$db_name = "sample_db";

// подключение к базе данных
$conn = mysql_connect($hostname, $db_user, $db_pass);
if (!$conn){
    die ("Не удалось подключиться к базе данных!");
}

mysql_select_db($db_name);
```

Затем формируется SQL-запрос, который будет использоваться для получения необходимых результатов из базы данных. Так как требуется создать список не всех записей, а только уникальных игровых позиций, используется ключевое слово `DISTINCT`. Затем результаты вставляются в массив:

```
$sql = "SELECT DISTINCT userposition FROM user";
$res = mysql_query($sql);
$results = array();
$i=0;
while ($pos=mysql_fetch_row($res)) {
    $results[$i++] = $pos[0];
}
```

Весьма интересна завершающая часть сценария. Результаты присваиваются переменной шаблона (в данном случае она называется `results`), а затем отображается шаблон (`index.tpl`):

```
$smarty->assign('results', $results);
$smarty->display('index.tpl');
?>
```

Это конец PHP-сценария, поэтому теперь можно перейти собственно к шаблону, файлу `index.tpl`. Код шаблона начинается со стандартных HTML-тегов и краткого поясняющего текста:

```
<HTML>
<BODY>
<B>Перечень позиций игроков: </B><BR><BR>
```

В программной части шаблона используется функция `section` для обработки массива результатов в цикле. Для этой функции атрибуты `name` и `loop` являются обязательными. В рассматриваемом примере цикл выполняется до тех пор, пока в массиве `$results` есть необработанные значения. Значение атрибута `name` функции `section` должно быть указано рядом с именем переменной в фигурных скобках:

```
{section name=position loop=$results}
    {$results[position]}<BR>
{/section}
```

Код шаблона завершается закрывающими HTML-тегами:

```
</BODY>
</HTML>
```

Также следует отметить, что сгенерированный PHP-файл сохраняется в каталоге `templates_c`. Этот файл является результатом работы шаблона. Нет необходимости разбираться в коде, который содержится в этом файле, — он представлен здесь только для наглядности:

```

<?php /* Smarty version 2.6.2, created on 2004-04-12 21:22:51
        compiled from index.tpl */ ?>
<HTML>
<BODY>
<B>Перечень позиций игроков: </B><BR><BR>
<?php if (isset($this->_sections['position']))
unset($this->_sections['position']);
$this->_sections['position']['name'] = 'position';
$this->_sections['position']['loop'] = is_array($_loop=$this->_tpl_vars['results']) ? count($_loop) : max(0, (int)$_loop); unset($_loop);
$this->_sections['position']['show'] = true;
$this->_sections['position']['max'] = $this->_sections['position']['loop'];
$this->_sections['position']['step'] = 1;
$this->_sections['position']['start'] = $this->_sections['position']['step'] > 0 ? 0 : $this->_sections['position']['loop']-1;
if ($this->_sections['position']['show']) {
    $this->_sections['position']['total'] =
        $this->_sections['position']['loop'];
    if ($this->_sections['position']['total'] == 0)
        $this->_sections['position']['show'] = false;
} else
    $this->_sections['position']['total'] = 0;
if ($this->_sections['position']['show']):

for ($this->_sections['position']['index'] =
$this->_sections['position']['start'],
$this->_sections['position']['iteration'] = 1;
$this->_sections['position']['iteration'] <=
$this->_sections['position']['total'];
$this->_sections['position']['index'] +=
$this->_sections['position']['step'],
$this->_sections['position']['iteration']++):
$this->_sections['position']['rownum'] =
$this->_sections['position']['iteration'];
$this->_sections['position']['index_prev'] =
$this->_sections['position']['index'] -
$this->_sections['position']['step'];
$this->_sections['position']['index_next'] =
$this->_sections['position']['index'] +
$this->_sections['position']['step'];
$this->_sections['position']['first'] =
($this->_sections['position']['iteration'] == 1);
$this->_sections['position']['last'] =
($this->_sections['position']['iteration'] == $this->_sections['position']['total']);
?> <?php echo $this->_tpl_vars['results'][$this->_sections['position']
['index']]; ?>
<BR>
<?php endfor; endif; ?>
</BODY>
</HTML>

```

Этим примером завершается вводный материал по системе шаблонов Smarty. Более подробная информация о Smarty представлена на странице документации проекта <http://smarty.php.net/manual/ru/>. Теперь, имея достаточное представление о Smarty, вы сможете разобраться с особенностями использования системы в описываемом учебном примере.

PHPUnit

Получить самую свежую версию пакета PHPUnit можно по адресу <http://sourceforge.net/projects/phpunit/>. После загрузки пакета и его размещения в каком-либо каталоге необходимо убедиться, что файл `phpunit.php` доступен PHP, поэтому путь к выбранному каталогу нужно указать в параметре `include_path` в `php.ini`. Если изменять `php.ini` нежелательно, то можно установить пакет в тот же каталог, в котором расположен тестируемый код.

Рассмотрим практическое применение пакета PHPUnit.

Работа с PHPUnit

В рамках одного небольшого раздела невозможно рассмотреть все функции PHPUnit, однако, имея базовые знания, читатель может самостоятельно поэкспериментировать с различными классами пакета.

Во-первых, PHPUnit предоставляет функции, которые можно использовать для проверки определенных условий. Например, требуется определить, какое значение возвращает заданный метод, `true` или `false`. Чтобы это сделать, необходимо сформулировать утверждение об этом методе и проверить, является ли оно верным. Для проверки правильности утверждений PHPUnit предоставляет метод `assert()`:

```
function assert($boolean, $message=0) {
    if (! $boolean)
        $this->fail($message);
}
```

Например, чтобы для проверки открытия соединения с базой данных, может использоваться следующая функция:

```
function isOpen($db){
    If ($db == null){
        return false;
    }
    return true;
}
```

Зная, что функция `isOpen()` возвращает булево значение, можно использовать функцию `assert()` так:

```
$this->assert(isOpen($db), "Соединение не создано!");
```

Функция `assert()` не единственная используемая функция в PHPUnit, а кроме того, существуют разные варианты ее применения (хотя пока это все, что требуется знать об использовании методов PHPUnit).

Как же добраться до этих методов, ведь очевидно, что должен существовать проинициализированный PHPUnit-объект? Здесь проявляется вся ценность PHPUnit. Использование этого пакета позволяет унифицированным и интуитивно понятным способом запускать за один раз все тесты. Ниже описана последовательность тестирования кода с помощью PHPUnit.

1. Для формирования контекста, в котором можно будет запускать тесты, необходимо создать класс, расширяющий класс `TestCase` пакета PHPUnit:

```
class MultiLogTestSuite extends TestCase
```

2. Специфический контекст, в котором запускаются тесты, создается путем определения метода `setUp()` — буквально создается среда, в которой будет тестироваться код. В рассматриваемом учебном примере в методе `setUp()`, который впоследствии будет использоваться для запуска тестов, создается два `UserLog`-объекта и определяются значения для них. (В действии это будет показано в конце главы в разделе “Тестирование приложения”).

3. Создаются необходимые тесты с именами, которые следуют описанному ниже соглашению:

```
testMeaningfulName()
testAnotherMeaningfulName()
```

4. Создается новый комплект тестов:

```
$suiteOfTests = new TestSuite;
```

5. В комплект добавляются тесты:

```
$suiteOfTests->addTest(New MultiLogTestSuite("testMeaningfulName");
$suiteOfTests->addTest(New MultiLogTestSuite("testAnotherMeaningfulName");
```

6. Создается новый `TestRunner`-объект. В классе `TestRunner` просто определена функция `run()` для запуска всех созданных тестов и вывода отчета о тестировании.

```
$testRunner = new TestRunner();
```

7. Наконец, вызывается метод `run()` (определенный на предыдущем этапе):

```
$testRunner->run($suiteOfTests);
```

Это все. Теперь `PHPUnit` добросовестно выполнит все тесты в комплекте и выведет отчет обо всех результатах (все эти этапы подробно описаны далее в настоящей главе). В этом разделе приведено достаточно сведений для начала работы. После прочтения главы читателю рекомендуется самостоятельно поэкспериментировать с классом `PHPUnit`, потому что само по себе умение использовать блочные тесты имеет определенную ценность.

Теперь, имея необходимые базовые знания, можно приступить к созданию и тестированию учебного примера.

Проектирование диспетчера протоколирования

Прежде чем браться за написание кода приложения, необходимо многое обдумать. Начиная новый проект, нужно в первую очередь ответить на следующие вопросы.

- ☐ Чего вы пытаетесь добиться?
- ☐ Как вы собираетесь это сделать?

Для любого разрабатываемого приложения стоит обдумать несколько различных аспектов. Например, как будет реализована проверка и обработка ошибок, и нужно ли реализовать какую-либо систему безопасности?

В данном случае особая система безопасности не требуется, потому что приложение не будет записывать информацию с кредитных карт или другие секретные сведения. По существу, в учебном примере рассматривается протоколирование таких данных,

как параметры пользователей, идентификаторы сайтов и любой другой информации, которую могут отправлять пользователи. Информации личного характера здесь нет. В то же время необходимо обеспечить хорошую обработку исключительных ситуаций, с этой целью реализуется интерфейс `DataValidation` для обработки ошибок (подробнее об этом далее).

Приложение предположительно можно разделить на две подпрограммы в зависимости от направления передачи данных — в базу данных или из нее. С одной стороны необходимо получать POST- и GET-информацию с каждого сайта и заставить приложение точно протоколировать полученные данные. С другой стороны должна быть возможность получать сохраненные данные и выводить их в удобном, читабельном формате. Все это представляет собой высокоуровневую организацию приложения.

В примере используются современные методики программирования, а именно — объектно-ориентированный подход к созданию приложения. PHP5, как известно, хорошо поддерживает объектно-ориентированный принцип, поэтому, реализуя приложение, определенно стоит воспользоваться эффективностью объектов. Одним из значительных преимуществ ОО-программирования является возможность создать с помощью UML визуальное представление намеченного решения (как это делается, будет показано далее).

В ходе фазы тестирования необходимо будет создать целую группу тестов, гарантирующих, что приложение работает предсказуемо и что во время его реального использования не возникнут сбои. Часто отладка небольших сценариев сравнительно проста и не должна занимать много времени, но по мере роста размеров и сложности приложений увеличивается и время, необходимое для отладки. Как уже было сказано, `PHPUnit` позволяет сократить это время и предоставляет некоторые полезные функции, которые можно использовать для создания структурированного единообразного комплекта тестов.

Далее, следует учитывать то, что может возникнуть необходимость разделить приложение на несколько уровней, иначе говоря, создать многоуровневую архитектуру. Это в дальнейшем способствует отладке и поддержке приложения, потому что слабосвязанные архитектуры позволяют разделить логически обособленный код на модули. Например, `Smarty` помогает отделить код приложения от уровня представления. С другой стороны, для уровня данных можно создать класс, содержащий множество функций для работы с базами данных (аналогичный классу `DataManager`, который рассматривался в главе 13).

В качестве СУБД будет использоваться `SQLite`, а необходимые базы данных будут созданы с помощью специальных сценариев. Знание структуры базы данных предполагаемого приложения подталкивает разработчика обдумывать тип необходимого кода, поэтому таблицы баз данных представляют собой хорошую отправную точку для начала разработки приложения.

База данных `sitelogs.db`

Чтобы спланировать таблицы для разрабатываемого здесь проекта, необходимо сформулировать несколько предположений. Предположим, что существует два сайта для данного проекта — сайт комиксов и сайт информации об аппаратном обеспечении. При желании количество сайтов можно увеличить. Каждый сайт при регистрации требует от пользователя ответов на несколько вопросов. Типы сайтов, с которых будет собираться информация, не важны — для создания подходящей базы данных необходимо только знать тип дескрипторов полей для хранения данных. Например, для протоколирования даты доступа используется поле типа `date`, для хранения имен — поле строкового типа и т.д.

Естественно, само по себе простое протоколирование персональных сведений о пользователе на центральном сайте не особенно полезно. Вероятнее всего впоследствии необходимо будет узнать, на каких сайтах пользователь регистрировался, когда зарегистрировался, а также получить любые другие сведения по каждому сайту.

База данных `sitelogs.db` состоит из трех таблиц, в которых будет храниться пользовательская информация, а также вопросы и ответы с различных сайтов. Беспокоиться о создании таблиц не стоит — на сайте издательства представлены сценарии, позволяющие автоматизировать этот процесс. Достаточно лишь определить, какая информация будет храниться и где она будет храниться.

SQLite поставляется с PHP5 и в этой главе не рассматривается. Если в процессе работы с этой СУБД возникнут какие-либо проблемы, обратитесь к приложению В “Использование SQLite”.

Первая таблица называется `user_log` и создается с помощью следующего SQLite-запроса:

```
CREATE TABLE user_log (
  user_log_id integer primary key,
  visit_date   date,
  visit_time   time,
  site_id      int,
  demo_id      int,
  login_id     string,
  session      string,
  firstname    string,
  lastname     string,
  address1     string,
  address2     string,
  city         string,
  state        string,
  zip          string
)
```

В этой таблице содержится необходимая информация по каждому пользователю. Как видно из запроса, протоколируется дата и время посещения сайта, информация сеанса и регистрационная информация и т.д. Следует отметить, что также записывается идентификатор сайта; он необходим для того, чтобы определить, на какие вопросы отвечал тот или иной пользователь каждого сайта.

Следующая таблица, `user_demographics`, создается с помощью такого запроса:

```
CREATE TABLE user_demographics (
  user_log_id integer,
  seq          int,
  answer       string,
  primary key (user_log_id, seq)
)
```

В этой таблице содержатся ответы пользователей на вопросы сайта. Данная таблица связана с таблицей `user_log` по полю `user_log_id`.

Последняя таблица называется `demographic_description` и создается с помощью такого запроса:

```
CREATE TABLE demographic_description (
  demo_id      int,
  seq          int,
  question     string,
  primary key (demo_id, seq)
)
```

Очевидно, что эта таблица содержит вопросы, задаваемые на каждом сайте. В поле `demo_id` хранится идентификатор типа вопроса. Например, идентификатор 1 представляет вопросы для сайта комиксов, а идентификатор 2 — вопросы для сайта об аппаратном обеспечении. Так как вопросы задаются в определенном порядке, их порядковые номера хранятся в поле `seq`.

Таблица `demographic_description` предварительно заполняется необходимыми вопросами. Таблица `user_log` будет содержать базовую информацию, а ответы будут записываться в таблицу `user_demographic` и сортироваться по полю `seq`.

Использование UML для планирования диспетчера протоколирования

Создание диаграмм, описывающих задуманное приложение, — отличный способ наглядно представить себе это приложение. Все главные аспекты приложения необходимо обдумать до визуализации различных частей предполагаемого решения с помощью диаграмм классов и диаграмм последовательностей. Диаграммы последовательностей помогают понять движение информационного потока в диспетчере протоколирования от уровня представления к базе данных и обратно.

UML-диаграммы классов помогают понять конструкцию каждого класса, а также взаимосвязь классов. Диаграмма последовательности показывает, как все должно работать. Реализация кода будет описана далее.

Планирование классов обработки данных

Для начала предположим, что будут использоваться два главных типа журналов: `UserLog` и `UserDemographic`. В первом из них содержится информация об определенном пользователе. База данных хранит информацию, связанную с пользователями. Необходимо только способ внутреннего представления этой информации, потому что невозможно просто передать массив запроса непосредственно в базу данных. Строго говоря, это можно сделать, но данные, которые будут сохранены таким способом, скорее всего, окажутся бесполезными.

Наилучший способ представления пользовательской информации — объект, позволяющий выполнять над данными любые операции, которые могут понадобиться. Можно создать класс для обработки пользовательской информации, выбрав свойства и методы, необходимые для получения информации, выполнения некоторых операций (например, проверки достоверности данных) и сохранения информации в базе данных.

Конструктор `__construct()` можно использовать для инициализации объекта и наполнения массива, который будет использоваться для представления данных. Таким образом, класс `UserLog` будет иметь следующее свойство:

```
#contentBase: array
```

и метод:

```
+__construct(initdict)
```

Кроме того, в приложении должна быть возможность сохранять данные в базе, поэтому нужно реализовать соответствующий метод, а также добавить общедоступный метод `persist()`. Также необходим способ проверять данные объекта. Для этого будет использоваться метод `getInvalidData()`. На самом деле проверка данных будет реализована с помощью интерфейса, поэтому упомянутый метод будет просто обеспечивать эту функциональность для интерфейса `DataValidation`.

Говоря об интерфейсах, стоит упомянуть другую операцию, которая должна быть реализована во всех классах обработки данных: отображение информации пользователю. Для этого можно использовать интерфейс `DataOutput`, в котором объявлена абстрактная функция `toHTML()`. Это означает, что в классе `UserLog` должна быть определена функция `toHTML()`.

И все же для класса `UserLog` нужно сделать еще кое-что. Массив наполняется данными из запроса. Нужно, чтобы этот объект сохранял пользовательскую информацию в базе данных, но очевидно, демографические данные пользователей (ответы) должны быть частью глобальной переменной `$_REQUEST`. Определенно нужен способ извлечения демографических данных пользователей из этого массива и передачи этих данных объекту `UserDemographic`, который применяется для их обработки. В данном случае эту задачу решает конструктор класса.

Поскольку класс `UserLog` будет принимать всю собранную информацию, класс `UserDemographic` будет содержаться внутри него. Иначе говоря, объект `UserDemographic` может быть создан только из класса `UserLog`. Это имеет смысл, потому что обрабатывать демографические данные пользователя необходимо только во время записи его данных. (Не путайте с наследованием — лучше представлять себе эту ситуацию как существование одного объекта внутри другого объекта.)

Связь между классами `UserDemographic` и `UserLog` вынуждает определить в классе `UserLog` метод `gatherUserDemographics()`. Этот метод будет отвечать за получение уникальной демографической информации из базы данных для использования в классе `Logcontainer` (он будет описан далее).

Остается еще один вопрос, касающийся хранения пользовательских журналов: какое поле необходимо использовать в качестве первичного ключа? Для хранения идентификатора журнала (идентификатор используется в качестве первичного ключа в таблице `user_log`) необходимо частное свойство. Поэтому в классе создается частное свойство с именем `id` с целым значением по умолчанию равным 0.

Демографические данные пользователей также должны сохраняться и обрабатываться, поэтому необходим класс `UserDemographic` для выполнения соответствующих операций. В разрабатываемом проекте класс будет сохранять демографические данные одного пользователя и связанный вопрос. Вопрос проще всего хранить в частном свойстве, но один из самых важных аспектов этого класса заключается в том, что необходимо сохранить связь между пользователем, вопросами и порядком, в котором эти вопросы задаются (очевидно, что информация будет бесполезной, если для одного пользователя из базы данных будут извлекаться ответы другого). Поэтому в класс `UserDemographic` стоит добавить два общедоступных свойства `getId()` и `setSequence(sequence)`.

Класс `UserDemographic` не будет отвечать за сохранение своих данных, поскольку он содержится в классе `UserLog`, однако это не помешает заставить его генерировать собственные SQL-запросы, которые будет использовать класс `UserLog`. (Не следует реализовывать в классе `UserLog` метод специально для другого класса.) В результате в класс `UserDemographics` также добавляется функция `toSQL()`.

Естественно, эти два класса также имеют общие функции. Поэтому стоит задуматься над возможностью создать шаблонный класс, свойства и методы которого наследовались бы этими классами. Не осознавая этой возможности, вы не сможете в полной мере реализовать ОО-подход. Рассмотрим организацию класса `PersistableLog`.

Класс `PersistableLog` будет отвечать за реализацию методов `__get()`, `__set()`, `getProperty()` и `setProperty()`, а также деструктора `__destruct()`. По сути это будет базовый класс для двух других классов, которые обсуждались выше.

Объекты должны сохранять информацию в нескольких различных форматах. Один тип объектов будет массивом ключей и значений, относящимся к базе данных, другой будет содержать ключи и значения, подлежащие выводу в удобочитаемом для человека формате. В результате используемые объекты будут содержать массив, который можно будет применить для вставки значений в базу данных (посредством имен полей), и массив, который можно будет использовать для возвращения удобочитаемых для человека данных.

Наконец, рассмотрим класс `LogContainer`. Как следует из имени этого класса, он предназначен для хранения последовательностей `UserLog`-объектов. Зачем это нужно? При опросе базы данных должна быть возможность получить группу `UserLog`-объектов, созданных на основании определенных критериев. Например, чтобы просмотреть все журналы одного сайта, можно создавать объекты класса `LogContainer` путем получения и хранения всех `UserLog`-объектов, соответствующих этому сайту.

Внутренние `UserLog`-объекты должны сохраняться в частном массиве. Кроме того, понадобится частное соединение с базой данных, а также два удобных метода `getCount()` (возвращающий количество журналов, полученных в ходе запроса) и `getUserLogs()`.

UML-диаграмма на рис. 17.2 представляет рассмотренную выше организацию диспетчера протоколирования.

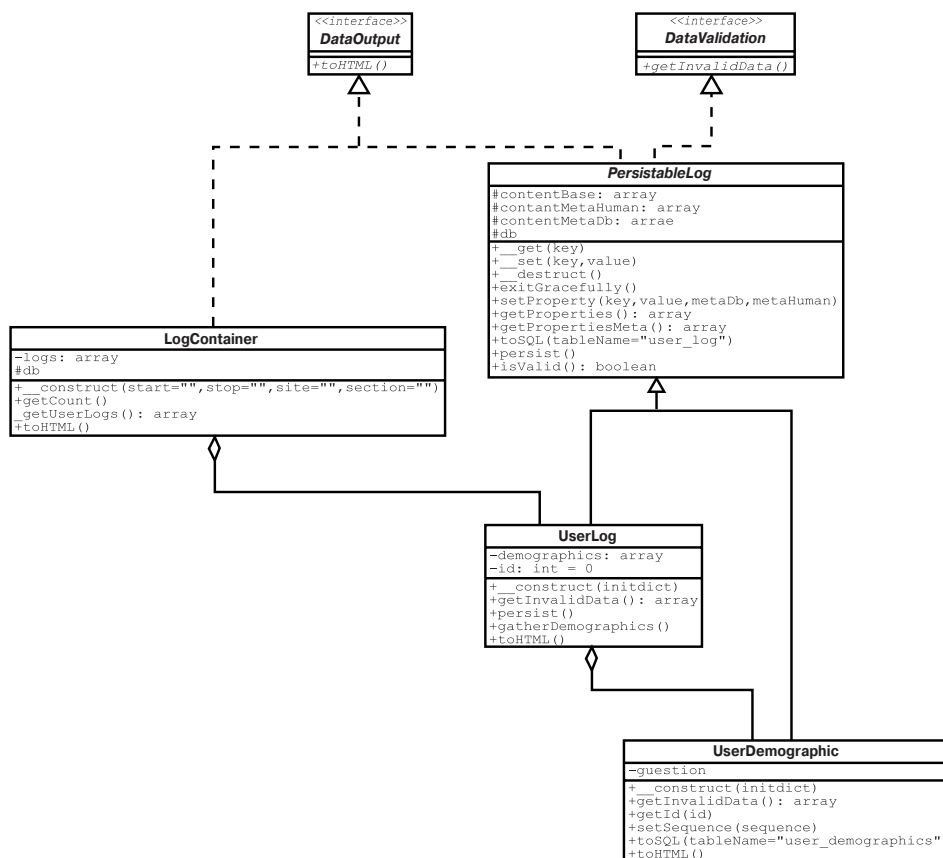


Рис. 17.2.

Еще стоит упомянуть класс `LogUtils`. Его основные функции для работы с базами данных наследуются описанными выше классами. Класс `LogUtils` представляет собой пример хорошей практики программирования, поддерживающей абстракции баз данных. Иными словами, чтобы заменить используемую СУБД, необходимо внести всего несколько изменений в этом классе. Класс `LogUtils` здесь не рассматривается, поскольку он, по сути, не является частью бизнес-логики приложения. Подробнее этот класс обсуждается в разделе описания кода.

Теперь рассмотрим диаграммы классов обработки исключений.

Планирование классов обработки исключений

В диспетчере протоколирования базовый класс обработки исключений `MultiLogException` расширяет встроенный PHP-класс `Exception` в соответствии с рекомендациями. Класс хранит сообщение об ошибке в виде частного свойства, значение которого присваивается в конструкторе `__construct($message)`. Естественно, этот класс также предоставляет две общедоступные функции `suggestedSolutions()` и `getErrorMessage()`.

Здесь следует отметить, что функция `suggestedSolutions()` определяется как абстрактная, поэтому она должна быть реализована в классах-наследниках. В связи с этим стоит рассмотреть диаграмму класса, поскольку существует множество различных типов исключений, которые должны предоставлять пользователю различные предложения по устранению ошибок.

Кроме этого, для работы со специфическими исключениями, такими как исключения классов `UserLog` и `LogContainer`, исключениями, связанными с некорректным подключением к базе данных, и просто исключительными ситуациями обработки данных, предназначен целый ряд специальных классов обработки исключений. Эти классы показаны на диаграмме (рис. 17.3).

Наконец, чтобы понять, как все это связано вместе, следует изучить диаграмму последовательностей приложения.

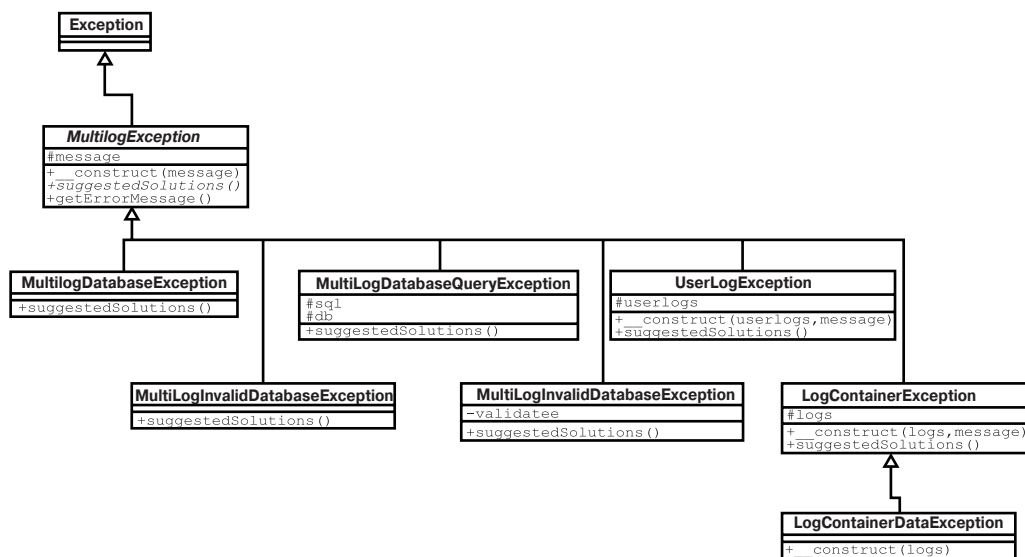


Рис. 17.3.

Диаграмма последовательностей диспетчера протоколирования

Из диаграммы последовательностей диспетчера протоколирования, представленной на рис. 17.4, должно быть очевидно, что большая часть приложения сосредоточена вокруг объекта UserLog. Это именно тот класс UserLog, который отвечает за создание объектов UserDemographic, выполнение запросов и возвращение информации из базы данных, а также используется при отправке и получении информации от Smarty-уровня (уровня представления).

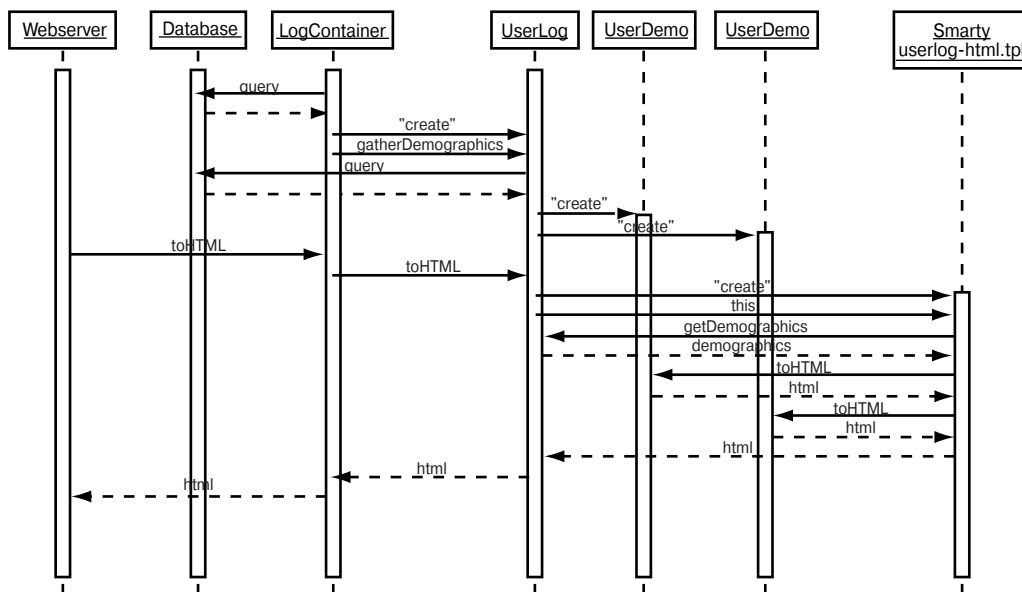


Рис. 17.4.

Код приложения

Итак, вы уже знаете, как организован диспетчер протоколирования, из чего состоит каждый класс, и хорошо представляете себе, как они связаны вместе. Во многом это самая трудная часть работы. Теперь остается только написать все, что запланировано. Для начала рассмотрим код главных классов приложения.

Вспомогательные сценарии

Прежде чем двигаться дальше, необходимо упомянуть несколько весьма небольших сценариев. Эти файлы нельзя назвать главными для функционирования приложения, но их стоит рассмотреть, чтобы избежать путаницы впоследствии.

settings.php

Файл `settings.php` содержит несколько глобальных значений, к которым приложение должно иметь доступ, и эти значения необходимо изменить, чтобы отразить настройки используемой платформы:

```
<? $GLOBALS['basepath'] = "/var/www/php5/";
$GLOBALS['baseurl'] = "http://localhost/php5/";
$GLOBALS['dbpath'] = "/var/www/php5/"; // хранить вне
// корневого каталога Web-сайта,
// в конце пути косая черта обязательна
$GLOBALS['dbname'] = "sitelogs.db";
$GLOBALS['smarty-path'] = $GLOBALS['basepath'] . "lib/smarty/";
$GLOBALS['maxdemo'] = 1024;
?>
```

common.php

В файле `common.php` подключается несколько файлов, которые чаще всего требуются для работы остальных классов:

```
<?
require_once ("class.exceptions.php");
require_once ("class.LogUtils.php");
?>
```

setup.php

Файл `setup.php` отвечает за инициализацию базы данных и в случае неудачи возвращает одно из нестандартных сообщений об ошибках:

```
<?
require_once ("settings.php");
require_once ("lib/common.php");

try {
    // инициализация базы данных
    require_once ("logs/initialize.php");
} catch (MultiLogException $e) {
    print "<h3>Произошла ошибка.</h3>";
    print $e->getErrorMessage();
}
?>
<h3>Готово</h3>
```

Сам по себе этот класс не отвечает за инициализацию базы данных. Это делается в файле `initialize.php`.

initialize.php

Сценарий `initialize.php` создает используемые в приложении базы данных. Эти базы данных уже обсуждались, поэтому нет необходимости подробно анализировать сценарий. Сценарий открывает подключение к базе данных, создает первую таблицу `user_log` и некоторые индексы в ней для ускорения поиска информации:

```
<?
$db = LogUtils::openDatabase ();

LogUtils::executeQuery($db, "drop table user_log");
LogUtils::executeQuery($db, "
CREATE TABLE user_log (
    user_log_id integer primary key,
    visit_date date,
    visit_time time,
    site_id int,
    demo_id int,
```

```

        login_id    string,
        session     string,
        firstname   string,
        lastname    string,
        address1    string,
        address2    string,
        city        string,
        state       string,
        zip         string
    )");

LogUtils::executeQuery($db,
    "CREATE INDEX index_user_log_site ON user_log (site_id)");
LogUtils::executeQuery($db,
    "CREATE INDEX index_user_log_demo ON user_log (demo_id)");
LogUtils::executeQuery($db,
    "CREATE INDEX index_user_log_login ON user_log (login_id)");
LogUtils::executeQuery($db,
    "CREATE INDEX index_user_log_date ON user_log (visit_date)");

```

Затем создается таблица `user_demographics`:

```

LogUtils::executeQuery($db, "drop table user_demographics");
LogUtils::executeQuery($db, "
CREATE TABLE user_demographics (
    user_log_id integer,
    seq          int,
    answer       string,
    primary key (user_log_id, seq)
)");

```

Таблица `demographic_description` создается несколько иначе, поскольку в ней не будет содержаться информация, полученная с сайта:

```

LogUtils::executeQuery($db, "drop table demographic_description");

LogUtils::executeQuery($db, "
CREATE TABLE demographic_description (
    demo_id    int,
    seq        int,
    question   string,
    primary key (demo_id, seq)
)");

```

В этой таблице хранятся вопросы (демографические) для каждого сайта, поэтому ее необходимо проинициализировать некоторыми данными (в реальной среде разработчику понадобилось узнать вопросы для сайта и по отдельности добавить их в эту таблицу). Фактически вопросы, размещаемые на определенном сайте, могут даже извлекаться из этой базы данных — иными словами, вопросы формулируются, согласовываются и записываются в базу данных. Когда все вопросы будут готовы, администраторы сайта загрузят их на сайт.

Эта таблица является важнейшей для приложения — в ней содержатся данные, которые необходимо связать с ответами, хранящимися в таблице `user_demographics`. Без этой таблицы невозможно определить, какой ответ соответствует тому или иному вопросу. Примеры вопросов, которые можно задать пользователям обоих сайтов, можно найти в приведенных далее INSERT-операторах:

```

LogUtils::executeQuery($db,
    "CREATE INDEX index_demographic_description_demo_pk ON
    demographic_description (demo_id, seq)");

```

```

LogUtils::executeQuery($db,
"INSERT INTO demographic_description (demo_id, seq, question)
VALUES (1, 0, 'Комиксы, приобретенные в течение месяца.' )");

LogUtils::executeQuery($db,
"INSERT INTO demographic_description (demo_id, seq, question)
VALUES (1, 1, 'Названия' )");

LogUtils::executeQuery($db,
"INSERT INTO demographic_description (demo_id, seq, question)
VALUES (1, 2, 'Цены' )");

LogUtils::executeQuery($db,
"INSERT INTO demographic_description (demo_id, seq, question)
VALUES (1, 3, 'Кредит' )");

LogUtils::executeQuery($db,
"INSERT INTO demographic_description (demo_id, seq, question)
VALUES (2, 0, 'В каком возрасте Вы впервые воспользовались отверткой?' )");

LogUtils::executeQuery($db,
"INSERT INTO demographic_description (demo_id, seq, question)
VALUES (2, 1, 'Средняя периодичность приобретения отверток.' )");

LogUtils::executeQuery($db,
"INSERT INTO demographic_description (demo_id, seq, question)
VALUES (2, 2, 'Общее количество отверток, приобретенное в течение жизни.' )");

LogUtils::executeQuery($db,
"INSERT INTO demographic_description (demo_id, seq, question)
VALUES (2, 3, 'Периодичность использования отвертки.' )");

```

Сценарий завершается закрытием подключения к базе данных:

```

LogUtils::closeDatabase ($db);
?>

```

Сценарии обработки данных

Сценарии обработки данных образуют основу приложения и отвечают за всю обработку данных начиная с открытия и закрытия подключений к базе данных, преобразования входящей информации в корректный внутренний формат и заканчивая получением и обработкой данных, т.е., по сути, за все приложение. В данном разделе рассматриваются следующие классы:

- ❑ class.LogUtils.php
- ❑ class.PersistableLog.php
- ❑ class.UserLog.php
- ❑ class.LogContainer.php
- ❑ class.UserDemographic.php

class.LogUtils.php

Класс class.LogUtils.php содержит большинство функций, позволяющих поддерживать абстракцию базы данных. Этот класс содержит несколько удобных функций (которые могут оказаться полезными). Сначала, чтобы реализовать обработку

ошибок, подключается класс Exceptions. Затем класс объявляется как абстрактный, поскольку сам по себе он ничего не должен делать — он лишь содержит методы, необходимые другим классам, поэтому создавать объекты этого класса не нужно:

```
<?
require_once ("class.exceptions.php");

abstract class LogUtils
{
```

Код класса начинается с объявления нескольких полезных методов, которые используются другими классами. Например, следующий метод возвращает стандартное значение, в случае если источник отсутствует:

```
public static function getDef (&$source, $default)
{
    if (!empty ($source)) return $source;
    return $default;
}
```

Этот метод, в частности, используется в конструкторе класса UserLog для присвоения значения user_log_id по умолчанию:

```
$this->id = LogUtils::getDef ($initdict["user_log_id"], 0);
```

Следующая полезная функция работает так же, как PHP-функция implode, в отличие от последней она заключает переданные ей строки в кавычки, а числа оставляет без кавычек:

```
public static function implodeQuoted (&$values, $delimiter)
{
    $sql = "";
    $flagIsFirst = true;
    foreach ($values as $value) {
        if ($flagIsFirst) {
            $flagIsFirst = false;
        } else {
            $sql .= $delimiter;
        }

        if (gettype ($value) == "string") {
            $sql .= "'".$value."'";
        } else {
            $sql .= $value;
        }
    }
    return $sql;
}
```

Эта функция впоследствии используется функцией generateSQLInsert() для форматирования запросов.

Затем объявляются две функции для работы с датами и временем:

```
public static function formattedDate ()
{
    return substr(date("c"), 0, 10);
}

public static function formattedTime ()
{
    return substr(date("c"), 11, 8);
}
```

Другая полезная функция необходима для преобразования N-мерных массивов в одномерные:

```
public static function arrayNth ($a, $n)
{
    $out = array();
    foreach ($a as $item) {
        $out[$item[$n]]=$item[$n];
    }
    return $out;
}
```

Так как данные внутри приложения желательно представлять в виде массивов, следующая функция является особенно полезной. Она принимает запрос и помещает его результат в массив:

```
public static function queryToMultidimArray ($db, $sql)
{
    $out = array();
    $query = LogUtils::executeQuery ($db, $sql);

    if ( !$query ) {
        throw new MultiLogDatabaseQueryException ($sql);
    }
    while ($row = LogUtils::getQueryArray($query)) {
        array_push ($out, $row);
    }
    return $out;
}
```

В файл для удобства включены две функции. Первая из них возвращает массив всех имеющихся в базе данных сайтов, а вторая возвращает все уникальные разделы (или `demo_ids`):

```
public static function gatherSites ($db)
{
    return LogUtils::queryToMultidimArray(
        $db, "select distinct site_id from user_log order by site_id");
}
public static function gatherSections ($db)
{
    return LogUtils::queryToMultidimArray(
        $db, "select distinct demo_id from user_log order by demo_id");
}
```

Одной из главных функций, которыми необходимо обеспечить этот класс, является функция, открывающая SQLite-базы данных. Приведенный ниже код предназначен именно для этого. (Функция возвращает одно из нестандартных исключений `MultiLogOpenDatabaseException()`, если во время доступа к базе данных возникла какая-либо проблема):

```
public static function openDatabase ()
{
    $db = sqlite_popen($GLOBALS['dbpath'].$GLOBALS['dbname'], 0666, $err);
    if ( !$db ) {
        throw new MultiLogOpenDatabaseException ();
    }
    return $db;
}
```


Обычно если открывается подключение к базе данных, то предоставляется функция для его закрытия. Однако так как эта функция создает постоянное соединение, его закрытие может негативно отразиться на других частях приложения, использующих это же соединение. Если решено создавать отдельные соединения каждый раз, когда необходим доступ к данным, то закрывать эти соединения можно с помощью примерно такого кода:

```
public static function closeDatabase ($db)
{
    // вот как это может выглядеть:
    if ( $db ) {
        sqlite_close ($db);
    }
}
```

В следующей функции используется созданная ранее функция `implodeQuoted`. Она возвращает сгенерированный SQL-оператор:

```
public static function generateSqlInsert ($tableName, &$amp;metas, &$amp;values)
{
    return "insert into ".$tableName.
        " (" .implode ($metas, ", ")." ) ".
        " values(" .LogUtils::implodeQuoted($values, ", ").")";
}
```

Следует позаботиться об ошибках, которые могут возникать при работе с базой данных. Можно создать функцию, которая возвращала бы последнюю сгенерированную базой данных ошибку. Такая функция используется в классе `Exceptions` для вывода на экран сообщения об ошибке, в случае если SQL-запрос сформирован неверно.

Такой подход может показаться неверным, поскольку выводить в браузер сообщения об ошибках, которые раскрывают устройство приложения, не следует даже в самом крайнем случае. В реальной среде это действительно недопустимо, однако здесь это делается лишь в качестве упражнения.

```
public static function databaseError ($db)
{
    $err = "";

    if ( $db ) {
        $err = "error #".sqlite_last_error($db).
            " : ".sqlite_error_string (sqlite_last_error($db));
    }
    return $err;
}
```

Код класса завершается определениями нескольких функций, поддерживающий абстракцию баз данных:

```
public static function executeQuery ($db, $sql)
{
    return sqlite_query($sql, $db);
}

public static function getLastInsertedRowId ($db)
{
    return sqlite_last_insert_rowid($db);
}

public static function getQueryArray ($query)
{
    return sqlite_fetch_array ($query);
}
```

```
}
}
?>
```

Теперь можно перейти к изучению класса `class.PersistableLog.php`, свойства и методы которого наследуются остальными классами.

class.PersistableLog.php

Основная задача файла `class.PersistableLog.php` заключается в предоставлении инструментов и функций, которые необходимы другим классам для сохранения информации объекта в базе данных. Естественно, этот класс для создания подключений к базе данных использует средства класса `LogUtils` (описанного выше) — это представляет собой один уровень абстракции баз данных.

Сначала создается абстрактный класс (поскольку его объекты создавать не требуется), а также объявляется несколько массивов, которые будут использоваться для хранения метаданных базы данных об обрабатываемой информации. Иначе говоря, массив `$contentBase` должен содержать фактические значения, которые вставляются в базу данных, а массив `$contentMetaDb` — имена полей, в которые вставляются эти значения. Массив `$contentMetaHuman` содержит значения в удобочитаемом формате.

```
abstract class PersistableLog implements DataValidation
{
    protected $contentBase = array();
    protected $contentMetaHuman = array();
    protected $contentMetaDb = array();

    protected $db = null;
```

Затем объявляется функция `__get()`:

```
function __get ($key)
{
    if (array_key_exists ($key, $this->contentBase)) {
        return $this->contentBase[$key];
    }
    return null;
}
```

Естественно, можно также объявить деструктор для закрытия всех открытых ресурсов:

```
function __destruct ()
{
    $this->exitGracefully();
}

function exitGracefully ()
{
    LogUtils::closeDatabase ($this->db);
    $this->db = null; // общий способ проверки открытого подключения
                     к базе данных
}
```

Создается функция `setProperty()` для установки локального свойства, а также функции `getProperties()` и `getPropertiesMeta()` для возвращения массивов свойств:

```
function setProperty ($key, $value, $metaDb, $metaHuman)
{
    $this->contentBase[$key] = $value;
    $this->contentMetaHuman[$key] = $metaHuman;
    $this->contentMetaDb[$key] = $metaDb;
}

function getProperties () {
    return $this->contentBase;
}

function getPropertiesMeta () {
    return $this->contentMetaHuman;
}
```

Теперь с помощью следующей функции формируется SQL-оператор (обратите внимание на использование LogUtils-функции generateSqlInsert()):

```
function toSQL ($tableName = "user_log")
{
    return LogUtils::generateSqlInsert (
        $tableName, $this->contentMetaDb, $this->contentBase);
}
```

Далее начинается бизнес-часть класса. Функция persist() отвечает за вставку информации в базу данных. В случае неудачи она изящно завершает свою работу.

```
function persist ()
{
    $rowid = 0;
    // если объект некорректен, то нет смысла продолжать.
    if (!$this->isValid()) {
        throw new MultiLogInvalidDataException ($this);
    }
}
```

Так как после сохранения данных следует закрыть соединение, сначала необходимо проверить, активно ли оно. Если нет, то:

```
if ($this->db == null) {
    throw new MultiLogInvalidDatabaseException($this);
}
```

Если все идет как следует, то генерируется необходимый SQL-оператор, после чего соединение закрывается:

```
if (LogUtils::executeQuery($this->db, $this->toSQL())) {
    $rowid = LogUtils::getLastInsertedRowId ($this->db);
    $this->exitGracefully();
} else {
    throw new MultiLogDatabaseQueryException ($this->toSQL(),
        $this->db);
}
```

Применение первичного ключа часто оказывается очень удобным, поскольку, по сути, первичный ключ представляет собой уникальный (для базы данных) идентификатор. Если он есть, то возвращаем его:

```
    return $rowid;
}
```

Наконец, определяется функция `isValid()`, которая используется в `persist()` — это булева версия функции `getInvalidData()`:

```
function isValid ()
{
    if (count ($this->getInvalidData()) == 0) {
        return true;
    }
    return false;
}
?>
```

class.UserLog.php

Основное назначение класса `class.UserLog.php` — связать данные пользовательского журнала с базой данных. Кроме того, здесь связываются ответы с вопросами определенного сайта. Класс `UserLog` реализует интерфейс `DataValidation` и может сам с помощью описанного выше класса `PersistableLog` сохранять собранные данные в SQLite-базе данных. Как и остальные классы обработки данных, `UserLog` для работы с SQLite-соединениями использует класс `LogUtils`.

Для начала подключаются все необходимые файлы. Следует отметить, что здесь также подключается класс `Smarty`. Это сделано потому, что одна из функций наглядно показывает, как можно использовать `Smarty` для вывода данных в HTML-формате. Здесь также используется класс `LogUtils`, но он подключается посредством файла `common.php`:

```
<?
require_once ("common.php");
require_once ("class.PersistableLog.php");
require_once ("class.UserDemographic.php");
require_once ("interface.DataValidation.php");
require_once ('Smarty.class.php');
```

Затем объявляются необходимые частные переменные и создается конструктор. Назначение конструктора — связать поступающие данные с внутренними свойствами и именами уровня базы данных. Это возможно, поскольку данные, передаваемые классу `UserLog` во время создания объекта, организованы в форме массива:

```
class UserLog extends PersistableLog implements DataValidation
{
    private $demographics = array();
    private $id = 0;

    function __construct ($initdict) // передача по значению
    {
        $key = "";
        $this->db = LogUtils::openDatabase ();
        $this->id = LogUtils::getDef ($initdict["user_log_id"], 0);
    }
}
```

Чтобы ясно представить происходящее, нужно знать, как обрабатывается одно значение. Ниже показано, как устанавливается свойство `visit_date` — этот процесс повторяется для всех остальных ключей. Сначала создается имя поля базы данных:

```
$key="visit_date";
```

С помощью функции `setProperty()`, содержащейся в классе `PersistableLog`, ключ связывается с соответствующим значением, а также с удобочитаемым описанием (которое, например, можно выводить на Web-странице). Функция `setProperty()` имеет следующую форму:

```
function setProperty ($key, $value, $metaDb, $metaHuman)
```

Зная об этом, можно понять, что `$value` в данном случае задается следующим выражением:

```
LogUtils::getDef ($initdict[$key], LogUtils::formattedDate())
```

Известно, что метод `getDef()` возвращает второй аргумент в случае отсутствия первого. В результате, чтобы обеспечить внутреннее представление данных со значениями по умолчанию (если реальные значения не были получены с сайта), используется следующий оператор:

```
$this->setProperty ($key,  
    LogUtils::getDef ($initdict[$key], LogUtils::formattedDate()),  
    $key, "Date of visit");
```

Установка остальных свойств выполняется точно так же. Необходимо лишь обработать каждое значение и вызвать функцию `setProperty()`:

```
$key="visit_time";  
$this->setProperty ($key,  
    LogUtils::getDef ($initdict[$key], LogUtils::formattedTime()),  
    $key, "Time of visit");  
  
$key="site";  
$this->setProperty ($key,  
    LogUtils::getDef ($initdict[$key], 0),  
    "site_id", "Site ID");  
  
$key="site_id";  
$this->setProperty ("site",  
    LogUtils::getDef ($initdict[$key], $this->site),  
    $key, "Site ID");  
  
$key="section";  
$this->setProperty ($key,  
    LogUtils::getDef ($initdict[$key], 0),  
    "demo_id", "Section ID");  
  
$key="section_id";  
$this->setProperty ("section",  
    LogUtils::getDef ($initdict["demo_id"], $this->section),  
    "demo_id", "Section ID");  
  
$key="login";  
$this->setProperty ($key,  
    LogUtils::getDef ($initdict[$key], ""),  
    "login_id", "Login ID");  
  
$key="login_id";  
$this->setProperty ("login",  
    LogUtils::getDef ($initdict[$key], $this->login),  
    $key, "Login ID");  
  
$key="session";  
$this->setProperty ($key,  
    LogUtils::getDef ($initdict[$key], ""),  
    $key, "Session");
```

```

$key="firstname";
$this->setProperty ($key,
    LogUtils::getDef ($initdict[$key], ""),
    $key, "First Name");

$key="lastname";
$this->setProperty ($key,
    LogUtils::getDef ($initdict[$key], ""),
    $key, "Last/Sur Name");

$key="address1";
$this->setProperty ($key,
    LogUtils::getDef ($initdict[$key], ""),
    $key, "Address Line 1");

$key="address2";
$this->setProperty ($key,
    LogUtils::getDef ($initdict[$key], ""),
    $key, "Address Line 2");

$key="city";
$this->setProperty ($key,
    LogUtils::getDef ($initdict[$key], ""),
    $key, "City");

$key="state";
$this->setProperty ($key,
    LogUtils::getDef ($initdict[$key], ""),
    $key, "State/Province");

$key="zip";
$this->setProperty ($key,
    LogUtils::getDef ($initdict[$key], ""),
    $key, "ZIP/Postal Code");

```

Прежде чем закончить с конструктором, необходимо обработать ответы (демографические данные) на вопросы каждого сайта, которые также отправляются в запросе вместе с UserLog-информацией. Для хранения демографических данных имеется отдельная таблица, поэтому для создания объектов, которые можно отправлять в эту таблицу, вызывается класс UserDemographics. Количество ответов в запросе заранее не известно, поэтому необходимо использовать цикл до достижения границы массива. Этот цикл реализован в классе class.UserDemographic.php, который подключается в начале class.UserLog.php.

Для каждого ответа (отправленного в запросе в форме demo1=>value1, demo2=>value2 и т.д.) создается новый UserDemographic-объект, а его внутренним свойствам присваиваются значения, отправленные в запросе. (Класс UserDemographic будет рассматриваться далее.)

```

for ($i=0; $i <= $GLOBALS['maxdemo']; $i++) {
    $key = 'demo'.$i;
    if (array_key_exists($key, $initdict)) {
        $d = new UserDemographic (
            array("id"=>$this->id, "demo"=>$initdict[$key]));
        array_push ($this->demographics, $d);
    } else {
        break;
    }
}
}

```

Кроме этого, нужна функция для связывания уникальных ответов с предопределенными вопросами. Для этого необходим довольно громоздкий SQL-оператор, который помещается в функцию `gatherDemographics()`:

```
function gatherDemographics ()
{
    if (!$this->db) $this->db = LogUtils::openDatabase ();
    $sql = "SELECT ud.user_log_id AS id, ud.answer AS demo,
dd.question AS question FROM user_demographics ud
LEFT JOIN user_log ul ON
    ud.user_log_id = ul.user_log_id
LEFT OUTER JOIN demographic_description dd ON (
    ul.demo_id = dd.demo_id and ud.seq = dd.seq ) WHERE
    ul.user_log_id =
    '". $this->id.'" AND ul.site_id = '". $this->site.'" AND
    ul.demo_id =
    '". $this->section.'" ORDER BY ud.seq";

    $query = LogUtils::executeQuery ($this->db, $sql);
    if ( !$query ) {
        throw new MultiLogDatabaseQueryException ($sql);
    }

    while ($row = LogUtils::getQueryArray($query)) {
        $demo = new UserDemographic ($row);
        if ($demo->isValid()) {
            array_push ($this->demographics, $demo);
        }
    }

    LogUtils::closeDatabase ($this->db);
}
```

Также следует организовать некоторую проверку корректности данных. На самом деле эту проверку при желании можно расширить, однако здесь представлено только три теста. Фактически этот класс используется из функции `PersistableLog::isValid()` и классов обработки исключений (которые обсуждаются далее):

```
function getInvalidData ()
{
    $badDataEntries = array();
    if ($this->site == 0) {
        array_push ($badDataEntries, "'site' is zero");
    }
    if ($this->section == 0) {
        array_push ($badDataEntries, "'section' is zero");
    }

    if ($this->login == "") {
        array_push ($badDataEntries, "'login' is missing");
    }

    return $badDataEntries;
}
```

Наконец, определяется собственная функция `persist`, которая позволяет сохранить `UserDemographic`-данные в таблице `Demographics`. Необходимо также связать ее с `UserLog`-объектами, поэтому требуется доступ к значению поля `user_log_id`, которое можно получить только после сохранения `UserLog`-объекта:

```

function persist()
{
    // сохранить свои данные
    $this->id = parent::persist();
    //сохранить демографические данные, порядок имеет значение
    $i=0;
    foreach ($this->demographics as $demo) {
        $demo->setId($this->id);
        $demo->setSequence($i++);
        $demo->persist();
    }
}
}
?>

```

class.LogContainer.php

Имя этого класса ясно дает понять, что он выполняет роль контейнера для UserLog-объектов. Как обычно, в начале файла подключаются другие необходимые файлы, а затем объявляется класс:

```

<?php
require_once ("common.php");
require_once ("class.UserLog.php");

class LogContainer
{
    private $sql;
    private $logs = array();
    protected $db = null;

```

Конструктор инициализирует SQL-запрос, используемый для получения UserLog-информации, которую необходимо вернуть. (Этот класс вызывается для создания отчета по журналам, которые содержатся в базе данных.)

```

function __construct ($start = "",
                      $stop = "",
                      $site = "",
                      $section = "")
{
    $initsql = "SELECT * FROM user_log WHERE 1=1 and";
    if($site <> "") $initsql .= " site_id = '".$site."' and";
    if($section <> "") $initsql .= " demo_id = '".$section."' and";
    if($start <> "") $initsql .= " visit_date >= '".$start."' and";
    if($stop <> "") $initsql .= " visit_date <= '".$stop."'";

```

Далее проверяется, не заканчивается ли SQL-оператор ключевым словом “and”, после чего он используется для опроса базы данных:

```

// удалить последнее слово "and"?
if (substr($initsql, strlen($initsql)-4, 4) == " and") {
    $initsql = substr($initsql, 0, strlen($initsql) - 4);
}

// запрос к базе данных
$this->db = LogUtils::openDatabase();
$query = LogUtils::executeQuery ($this->db, $initsql);

if ( !$query ) {
    throw new MultiLogDatabaseQueryException ($initsql);
}

```


Теперь сценарий проверяет результаты, и если они корректны, то заново связывает их с вопросами. Если результаты некорректны, то они записываются в массив `$badLogs`, который можно использовать для выдачи сообщения об ошибке (в конце кода конструктора):

```
$badLogs = array();
while ($row = LogUtils::getQueryArray($query)) {
    $ul = new UserLog ($row);
    if ($ul->isValid()) {
        $ul->gatherDemographics(); //заново связать
                                //демографические вопросы с ответами
        array_push ($this->logs, $ul);
    } else {
        array_push ($badLogs, $ul);
    }
}

LogUtils::closeDatabase ($this->db);
if (count ($badLogs) > 0) {
    throw new LogContainerInvalidDataException($badLogs);
}
```

В остальной части класса представлено две удобные функции. Например, функция `getCount()` позволяет определить количество имеющихся журналов; эту функцию можно использовать для вывода сообщений в браузер:

```
function getCount ()
{
    return count ($this->logs);
}
function toHTML ()
{
    $html = '<table border="0" class="LogContainerTable">';
    foreach ($this->logs as $ul) {
        $html .= $ul->toHTML();
    }
    $html .= "</table>";
    return $html;
}
function getUserLogs ()
{
    return $this->logs;
}
?>
```

class.UserDemographic.php

Последний класс для обработки данных в диспетчере протоколирования — `class.UserDemographic.php` — фактически используется исключительно из класса `UserLog` (потому что вся демографическая информация так или иначе передается в `UserLog`). Класс хранит один пользовательский ответ, связанный с ним вопрос и `UserLog`-идентификатор. Ниже приведено начало кода:

```
<?
require_once ("common.php");
require_once ("class.PersistableLog.php");

class UserDemographic extends PersistableLog
{
    private $question = "";
```

Следует отметить, что этот класс расширяет класс `PersistableLog`, поэтому в нем можно использовать некоторые функции `PersistableLog`. Ниже приведен код конструктора:

```
function __construct ($initdict) // передается по значению
{
    $key = "";
    $this->db = LogUtils::openDatabase ();
    $this->setId(LogUtils::getDef ($initdict["id"],0));
    $this->setId(LogUtils::getDef ($initdict["user_log_id"], $this->id));
    $this->setProperty ("demo",
        LogUtils::getDef ($initdict["demo"], 0),
        "answer",
        LogUtils::getDef ($initdict["question"], "Demographic Answer"));
    $this->setProperty ("demo",
        LogUtils::getDef ($initdict["answer"], $this->demo),
        "answer",
        LogUtils::getDef ($initdict["question"], "Demographic Answer"));
    $this->question = LogUtils::getDef($initdict["question"], "");
}
```

Затем определяется функция `setId()`, используемая в конструкторе:

```
function setId ($id)
{
    $this->setProperty ("id", $id, "user_log_id", "User ID");
}
```

Порядок следования ответов очень важен:

```
function setSequence ($seq)
{
    $this->setProperty ("seq", $seq, "seq", "Sequence");
}
```

Следует проверить корректность `UserLog`-объекта, поэтому далее определяется функция `getInvalidData()`. Демографические данные проверять не следует, потому что отсутствие ответа вполне может означать корректный ответ:

```
function getInvalidData ()
{
    $badDataEntries = array();

    if ($this->id == null || $this->id <= 0) {
        array_push ($badDataEntries, "'id' is zero");
    }

    return $badDataEntries;
}
```

Наконец, переопределяем функцию `toSQL` следующим образом:

```
function toSQL ($tableName = "user_demographics")
{
    return LogUtils::generateSqlInsert ($tableName,
        $this->contentMetaDb,
        $this->contentBase);
}
```

Сценарии проверки данных и обработки ошибок

С самого начала очень важно продумать, как будет осуществляться проверка корректности данных и обработка возможных ошибок. В предыдущем разделе уже было показано, как можно с помощью интерфейса реализовать проверку корректности данных, а также как обеспечить класс для базовой обработки ошибок, которым впоследствии смогут воспользоваться другие классы. Это еще один пример того, как хорошие методики программирования позволяют выделить функции приложения с тем, чтобы максимально увеличить повторное использование кода и обеспечить единственную точку модификации определенной функциональности.

В этом разделе представлено два сценария. Один из них — интерфейс для проверки корректности данных, а другой — класс обработки исключений, методы и свойства которого наследуют все классы обработки данных. Очевидно, вы уже видели эти сценарии в действии, потому что в сценариях обработки данных многократно используются вызовы функций `MultiLogException()` и `getInvalidData()`.

Прежде чем углубляться в код, рассмотрим один из способов использования интерфейса `DataValidation` в диспетчере протоколирования.

Упомянутый интерфейс реализуется классом `class.exceptions.php`, который представляет собой базовый класс для всех исключений и обеспечивает несколько различных классов для обработки всевозможных ошибок. Согласно UML-диаграммам, представленным в этой главе ранее, файл `class.exceptions.php` содержит определения следующих классов:

- ❑ `MultiLogException`
- ❑ `MultiLogOpenDatabaseException`
- ❑ `MultiLogDatabaseQueryException`
- ❑ `MultiLogInvalidDatabaseException`
- ❑ `MultiLogInvalidDataException`
- ❑ `UserLogException`
- ❑ `LogContainerException`
- ❑ `LogContainerInvalidDataException`

Некоторые из этих классов заставляют класс, генерирующий исключение, реализовать для них функцию `getInvalidData()`. Например, в классе `MultiLogInvalidDataException` объявлена следующая функция:

```
function suggestedSolutions ()
{
    return $this->validatee->getInvalidData();
}
```

В классе `UserDemographic`, который уже рассматривался, функция `getInvalidData()` определена следующим образом:

```
function getInvalidData ()
{
    $badDataEntries = array();

    if ($this->id == null || $this->id <= 0) {
        array_push ($badDataEntries, "'id' is zero");
    }
}
```

```
// отсутствие ответа тоже может быть корректным ответом,
// поэтому проверять демографические данные не следует
return $badDataEntries;
}
```

Конструктор класса `UserDemographic` содержит следующий оператор:

```
$this->db = LogUtils::openDatabase ();
```

Если просмотреть код класса `LogUtils`, то станет ясно, что он расширяет базовый класс исключений, и это необходимо, потому что функция `openDatabase()` имеет следующую структуру:

```
public static function openDatabase ()
{
    $db = sqlite_popen($GLOBALS['dbpath'].$GLOBALS['dbname'], 0666, $err);
    if ( !$db ) {
        throw new MultiLogOpenDatabaseException ();
    }
    return $db;
}
```

Чтобы понять, как вызывается функция `suggestedSolutions()` в классе `MultiLogOpenDatabaseException`, необходимо вспомнить, что этот класс расширяет класс `MultiLogException`, в котором упомянутая функция объявлена как абстрактная, т.е. любой вызов этой функции должен быть реализован вызывающим классом. В данном случае класс `MultiLogException` действительно вызывает функцию `suggestedSolutions()`:

```
function getErrorMessage()
{
    $message = "<h3>Ошибка</h3>".$this->message."<br>";
    $message .= "<ul>";
    foreach ($this->suggestedSolutions() as $solution) {
        $message .= "<li>".$solution;
    }
    $message .= "</ul>";
    return ($message);
}
```

Приведенных выше примеров должно быть достаточно для понимания некоторых способов использования интерфейсов. Теперь рассмотрим код класса.

В начале следует весьма простое объявление — в нем нет ничего нового. По существу, эта функция используется для определения достоверности текущего состояния заданного объекта:

```
<?
interface DataValidation
{
    abstract function getInvalidData();
}
?>
```

Как уже отмечалось, `class.exceptions.php` представляет собой базовый класс обработки исключений, который наследуется всеми остальными классами, поэтому для начала ничего кроме интерфейса подключать не нужно:

```
<?
require_once ("interface.DataValidation.php");
```

Первым объявляется класс `MultiLogException`, который используется многими другими классами обработки исключений. Его необходимо объявить как абстрактный, поскольку в нем реализована одна абстрактная функция. Абстрактная функция используется для того, чтобы заставить подклассы реализовать их собственные функции `suggestedSolutions`:

```
abstract class MultiLogException extends Exception
{
    protected $message = ""; // каждая исключительная
                             // ситуация имеет свою причину

    /**
     * Инициализация экземпляра с сообщением об ошибке
     */
    function __construct ($msg)
    {
        $this->message = $msg;
    }

    abstract function suggestedSolutions ();
}
```

Кроме того, создается функция, возвращающая сообщение об ошибке. Очевидно, что она для реализации функции `suggestedSolutions()` опирается на вызывающую функцию, о чем уже говорилось ранее:

```
function getErrorMessage()
{
    $message = "<h3>Ошибка</h3>".$this->message."<br>";
    $message .= "<ul>";
    foreach ($this->suggestedSolutions() as $solution) {
        $message .= "<li>".$solution;
    }
    $message .= "</ul>";
    return ($message);
}
```

Затем определяется класс `MultiLogOpenDatabaseException`, расширяющий только что определенный класс `MultiLogException`. Конструктор этого класса просто передает сообщение родительскому конструктору (в `MultiLogException`):

```
class MultiLogOpenDatabaseException extends MultiLogException
{
    function __construct ()
    {
        parent::__construct ("Ошибка при открытии базы данных");
    }
}
```

В данном случае функция `suggestedSolutions()` реализуется непосредственно, потому что вызывающий класс этого не делает — функция вызывается, только когда возникает ошибка открытия базы данных, поэтому необходимо просто вывести стандартное сообщение. (Не стоит забывать, что в реальной среде сообщения о таких ошибках не следует выводить в браузер — это связано с большим риском для безопасности системы.)

```
function suggestedSolutions ()
{
    return array ("Запускался ли сценарий инициализации?",
                 "Доступен ли каталог (".$GLOBALS['dbpath'].")");
}
```

```

        для чтения и записи Web-сервером?",
        "Доступен ли файл (".$GLOBALS['dbname'].") для
        чтения и записи Web-сервером?",
        "Существует ли база данных и была ли она
        создана той же версией SQLite, что и в PHP?"
    );
    }
}

```

Теперь необходимо объявить класс `MultiLogDatabaseQueryException` для обработки исключений при выполнении запросов к базе данных. Этот класс тоже является наследником класса `MultiLogException` и передает сообщение родительскому конструктору. В классе реализуется функция `suggestedSolutions()`, поэтому конструктор должен принимать два аргумента, которые можно использовать в этой функции:

```

class MultiLogDatabaseQueryException extends MultiLogException
{
    protected $sql = "";
    protected $db = null;

    function __construct ($sql = "(no sql supplied)", $db = null)
    {
        parent::__construct ("Исключение при запросе к базе данных");
        $this->sql = $sql;
        $this->db = $db;
    }
}

```

Затем реализуется метод `suggestedSolutions()`, который использует значения переменных `$db` и `$sql` (соединение с базой данных и SQL-запрос соответственно), если они переданы, для вывода некоторых полезных сообщений:

```

function suggestedSolutions ()
{
    $err = array();

    if ($this->db) {
        array_push (
            $err, "Ошибка базы данных: ".LogUtils::databaseError ($this->db));
    }
    array_push ($err, "Ошибка базы данных, ожидалось SQL = ".$this->sql);
    array_push ($err, "Проверьте права на запись");
    array_push ($err, "Проверьте, достаточно ли дискового пространства для
        работы базы данных");

    return $err;
}
}

```

Следующий класс возвращает “подсказки”, если выясняется, что используется некорректное соединение с базой данных:

```

class MultiLogInvalidDatabaseException extends MultiLogException
{
    function __construct ()
    {
        parent::__construct ("Неправильно указанная база данных");
    }

    function suggestedSolutions ()
    {
        return array ("Корректно ли открылась база данных?",

```

```

    }
    "Была ли база данных закрыта дважды или
    сценарий пытается сохранить одни и те же
    данные дважды?");
}

```

Следующий класс весьма важен, потому что он сообщает причину появления некорректных данных. В этот класс передается объект, реализующий интерфейс `DataValidation` — это обязательно, потому что функция `getInvalidData()` в рассматриваемом классе не реализована. Исключение генерируется до сохранения объекта в базе данных. Как уже было сказано, прежде чем класс `PersistableLog` запишет информацию в базу данных, выполняется проверка корректности объекта. Если данные объекта некорректны, то объект передается функции, которая выясняет причину и сообщает о ней:

```

class MultiLogInvalidDataException extends MultiLogException {
    private $validatee = null;
    function __construct ($v)
    {
        parent::__construct ("Некорректные данные");
        $this->validatee = $v;
    }

    function suggestedSolutions ()
    {
        return $this->validatee->getInvalidData();
    }
}

```

Затем определяется несколько специальных классов для обработки `UserLog`-исключений. После отображения короткого сообщения об ошибке вызывается метод `exitGracefully()`, который закрывает соединение с базой данных — помните, что этот метод определен в `PersistableLog`, родительском по отношению к `UserLog` классу:

```

abstract class UserLogException extends MultiLogException
{
    protected $userLog = null;

    function __construct ($userLog, $message)
    {
        parent::__construct ($message);
        $this->userLog = $userLog;
        $this->userLog->exitGracefully(); // метод определен в PersistableLog
    }
}

```

В классе `LogContainer` также должна быть реализована обработка исключений, поэтому далее определяется класс, который обрабатывает все `UserLog`-объекты, созданные в контейнере, и извлекает из них некорректные данные:

```

abstract class LogContainerException extends MultiLogException
{
    protected $logs = null;

    function __construct ($logs, $message) {
        parent::__construct ($message);
        $this->logs = $logs;
    }
    function suggestedSolutions ()
    {

```

```

        $solutions = array();
        foreach ($this->logs as $log) {
            $solutions = array_merge (
                $solutions, $log.DataValitation::getInvalidData());
        }
        return $solutions;
    }
}

```

На самом деле генерируемое исключение не сообщает о том, в каком журнале содержатся некорректные данные. Для того чтобы это сделать, определяется класс-наследник `LogContainerException`:

```

class LogContainerInvalidDataException extends LogContainerException
{
    function __construct ($logs)
    {
        parent::__construct ($logs, "Объекты UserLog содержат некорректные данные.");
    }
}
?>

```

Таким образом, с помощью множества разнообразных объектно-ориентированных методов (начиная от простого вызова класса и заканчивая более сложным использованием наследования и интерфейсов) реализуется обработка исключений в различных классах обработки данных. Очевидно, что реализованную здесь обработку исключений можно расширить — описанные классы представляют собой хорошую базу для дальнейшего развития.

Наличие класса исключений значительно облегчает определение нового исключения (для обработки какой-либо проблемы) и его вызов из генерирующего исключение класса. Это наглядная демонстрация эффективности инкапсуляции, краеугольного камня объектно-ориентированной разработки программного обеспечения.

Сценарии уровня представления и шаблоны

После того как большая часть диспетчера протоколирования готова, следует рассмотреть пользовательский интерфейс приложения. Индексная страница позволяет инициализировать базу данных, запустить все тесты (их создание рассматривается в следующем разделе), отправить информацию диспетчеру протоколирования и посмотреть результаты. В этом разделе основное внимание уделено Smarty-шаблонам. Кроме этого, рассматривается также индексная страница — главная точка контакта пользователя с приложением.

index.php

Файл `index.php` служит входной точкой для всего приложения. Как видно из следующего кода, сценарий формирует набор гиперссылок, связанных с основными тестовыми файлами, а также со сценарием `userlog.php`, который использует классы обработки данных:

```

<H3>Настройки</H3>
<UL>
<LI><A HREF="setup.php">Инициализация базы данных</A> (Внимание! Данные будут удалены!)
<LI><A HREF="lib/test.UserLog.php">Запуск комплекта тестирования класса UserLog</A>

```



```
<LI><A HREF="lib/test.LogContainer.php">Запуск комплекта тестирования класса
LogContainer</A>
<LI><A HREF="lib/test.UserDemographic.php">Запуск комплекта тестирования
класса UserDemographic</A>
<LI><A HREF="post.php">Отправка журналов</A>
<LI><A HREF="report.php">Просмотр отчетов</A>
</UL>
```

Последняя ссылка в коде связана со страницей `report.php`, которая отвечает за отображение пользовательского интерфейса, где пользователь может выбрать интересующий его журнал для просмотра.

report.php

В этом сценарии (`report.php`) используются Smarty-шаблоны, поэтому сначала вместе с остальными необходимыми классами нужно подключить класс `Smarty`. Если вы используете собственный дистрибутив `Smarty`, следует скорректировать путь к классу `Smarty`:

```
<?
require_once ("settings.php");
require_once ("lib/common.php");
require_once ($GLOBALS["smarty-path"].'Smarty.class.php');
```

Затем следует создать `Smarty`-объект и присвоить `Smarty`-переменной `title` заголовок страницы:

```
$smarty = new Smarty;
$smarty->assign ('title', "MultiLog Report");
```

Затем требуется объявить функцию, которая будет форматировать элементы “сайт” и “раздел” для отображения в выпадающем списке. Для этого создается массив, а затем вставляется пустая строка для показываемого по умолчанию раздела. Функция `arrayNth()` применяется для преобразования ассоциативного массива в простой одномерный массив:

```
function optionMessage (&$a) {
    $a = array_pad ($a, count($a)*(-1)-1, ""); // увеличиваем массив
    $a[0][0] = ""; // вставляем пустую строку
    $a = LogUtils::arrayNth($a,0); // преобразовываем массив в одномерный,
    включающий только значения
    return $a;
}
```

Теперь необходимо решить, какой отчет показывать. Сначала отображается `report.tpl`, поскольку этот шаблон позволяет задать критерии для отбора журналов:

```
if ($_REQUEST['action'] <> "html") {

    // отобразить входные данные
    $db = LogUtils::openDatabase();

    $smarty->assign ('sites', optionMessage(LogUtils::gatherSites($db)));
    $smarty->assign ('sections', optionMessage(LogUtils::gatherSections($db)));

    LogUtils::closeDatabase($db);

    $smarty->display ("report.tpl");
}
```

После того как пользователь сделал выбор, для отображения результатов запроса используется шаблон `report-html.tpl`. Для этого сначала создается новый `LogContainer`-объект, который наполняется значениями, полученными в результате запроса, сделанного с помощью шаблона `report.tpl`. Затем для отображения результатов необходимо передать значения следующему шаблону:

```

} else {

    // отображение результатов
    require_once ("lib/class.UserLog.php");
    require_once ("lib/class.LogContainer.php");

    $logs = new LogContainer ($ _REQUEST["start_Year"].
        "-".$ _REQUEST["start_Month"]."-".$ _REQUEST["start_Day"],
        $ _REQUEST["stop_Year"]."-".$ _REQUEST["stop_Month"]."-".$
        $ _REQUEST["stop_Day"],      $ _REQUEST["site"], $ _REQUEST["section"]);

    $smarty->assign ('logs', $logs);
    $smarty->display ("report-html.tpl");
}

?>

```

Результаты работы этих сценариев показаны в конце главы, здесь рассмотрены непосредственно файлы шаблонов.

report.tpl

Шаблон `report.tpl` отображает несколько выпадающих списков, которые можно использовать для создания запроса на выбор и отображение необходимых журналов. Следует отметить, что форма возвращает значение `html` для параметра `action`. Это позволяет сценарию `report.php` определить, какой отчет отображать:

```

<html><title>MultiLog Report</title>
<body>

<h3>{$title}</h3>

<form action="report.php" method="post">
<table>
<tr><td align="right">Начальная дата:</td><td>{html_select_date prefix="start_"
start_year="-2"}</td></tr>
<tr><td align="right">Конечная дата:</td> <td>{html_select_date prefix="stop_"
start_year="-2"}</td></tr>
<tr><td align="right">Сайт:</td> <td><select name=site> {html_options
options=$sites} </select> </td></tr>
<tr><td align="right">Раздел:</td> <td><select name=section> {html_options
options=$sections} </select></td></tr>
<tr><td align="right"></td> <td><input type="submit" value="Показать
отчет"></td></tr>
</table>
<input type="hidden" name="action" value="html">
</form>

</body>
</html>

```

report-html.tpl

Если в базе имеются журналы для отображения, то для вывода их на экран используется шаблон `report-html.tpl`. Это самый простой шаблон для вывода, но его можно при необходимости расширить. Просто предположим, что шаблон отображает результаты в таблице и выводит количество показанных журналов. По диаграмме последовательности, которая была представлена выше, можно сделать вывод, что на самом деле запрос выполняется Smarty-шаблоном — в данном случае к методам `toHTML()` и `getCount()` класса `UserDemographic`:

```
<html><title>MultiLog Report</title>
<body>

<h3>{$title}</h3>

<table width="2048"><tr><td>
{$logs->toHTML ()}
</td></tr></table>

<h3>Количество журналов: {$logs->getCount ()}</h3>

</body>
</html>
```

В основном приложение можно считать законченным. В нем имеется вся бизнес-логика, а также реализована проверка корректности данных и обработка исключений. Однако это еще не все...

Тестирование приложения

Как узнать, что все действительно работает? Конечно, можно запускать сценарий несколько раз и проанализировать результаты всех попыток. Однако на самом деле необходим более серьезный подход, позволяющий проверить приложение на всех этапах ее работы. Здесь оказывается полезным пакет `PHPUnit`, который уже обсуждался в этой главе. Ниже рассматривается создание комплекта тестов, которые позволяют проверить, устойчиво ли работает приложение и работает ли оно так, как ожидалось.

Для проверки работы классов `UserLog`, `UserLogContainer` и `UserDemographic` служат три набора блочных тестов. Для каждого класса формируется специфический контекст, в котором запускаются тесты, а затем различными способами используются функции, предоставленные `PHPUnit`.

Для проверки работы класса `UserLog` следует разработать серию из 11 тестов. В этом разделе также рассматривается новая `PHPUnit`-функция `assertEquals()`, которая позволяет проверить несколько условий.

Объявление класса начинается как обычно. Как уже было сказано, любой тестовый класс, для того чтобы сформировать контекст и выполнять другие операции, должен быть подклассом `PHPUnit`-класса `TestCase`. Две объявленные в классе `TestUserLog` переменные служат в качестве контейнеров для тестируемых объектов.

Важно отметить, что тесты получают свои параметры из `settings-test.php` (отдельный файл настроек), потому что разные тесты для полной сквозной проверки приложения очищают и заполняют разные таблицы базы данных. Кроме того, в число подключаемых файлов включается `PHPUnit`:

```
<?php

require_once ("../settings-test.php");
require_once ("class.UserLog.php");
require_once ("phpunit/phpunit.php");

class TestUserLog extends TestCase {
    private $ulGood = null;
    private $ulBad = null;
}
```

Затем определяется специфический контекст, в котором будут выполняться тесты. Это делается путем определения значений в PHPUnit-функции `setUp()`. В данном случае создается два `UserLog`-объекта:

```
function setUp()
{
    $this->ulGood = new UserLog (
        array ('site'      => 1,
              'section'   => 2,
              'login'     => "3",
              'session'   => "1E23553",
              'firstname' => "Alice",
              'lastname'  => "AppleGate",
              'address1'  => "123Main",
              'city'      => "Sandusky",
              'state'     => "OH",
              'zip'       => "44870",
              'demo0'     => "21",
              'demo1'     => "15",
              'demo2'     => "22",
              'demo3'     => "26"));

    $this->ulBad = new UserLog (
        array ('site'      => 0,
              'section'   => 0,
              'login'     => "",
              'session'   => "1E23553",
              'firstname' => "Alice",
              'lastname'  => "AppleGate",
              'address1'  => "123Main",
              'city'      => "Sandusky",
              'state'     => "OH",
              'zip'       => "44870",
              'demo0'     => "21",
              'demo1'     => "15",
              'demo2'     => "22",
              'demo3'     => "26"));
}
```

Необходимо определить тестовые функции.

Первая функция, `testValid0()`, использует PHPUnit-функцию `assertEquals()` для проверки следующих условий:

- ☐ Sandusky — город, представленный в `UserLog`-объекте `$ulGood`;
- ☐ `$ulGood` — корректный журнал;
- ☐ `$ulGood` не имеет недостоверных данных.

Если тесты выполняются, то контекст, в котором они работают (по крайней мере, для объекта `$ulGood`), работает корректно. Чтобы изучить код функции `assertEquals()`, необходимо просмотреть файл `phpunit.php` из инсталляции PHPUnit:

```
function testValid0()
{
    $this->assertEquals("Sandusky", $this->ulGood->city, "invalid city");
    $this->assertEquals(true, $this->ulGood->isValid(), "valid log");
    $this->assertEquals(0, count ($this->ulGood->getInvalidData()),
        "valid count");
}
```

В следующем тесте создается новый UserLog-объект, \$ul, и заполняется некоторыми некорректными данными (а именно значением site). Формулируется утверждение, что функция isValid() вернет значение false, и что количество некорректных элементов будет равно 1. Если эти тесты выполняются верно, то можно быть уверенным, что программа правильно выбирает недостоверные данные внутри объектов:

```
function testInvalid0()
{
    $ul = new UserLog (array ('site'      => 0,
                             'section'   => 9,
                             'login'     => "user101",
                             'session'   => "1E23553",
                             'firstname' => "Alice",
                             'lastname'  => "AppleGate",
                             'address1'  => "123Main",
                             'city'      => "Sandusky",
                             'state'     => "OH",
                             'zip'       => "44870",
                             'demo0'     => "21",
                             'demo1'     => "15",
                             'demo2'     => "22",
                             'demo3'     => "26"));
    $this->assertEquals(false, $ul->isValid(), "valid ul");
    $this->assertEquals(1, count ($ul->getInvalidData()), "valid site");
}
```

Третий тест также проверяет объект \$ulBad. Используются утверждения о том, что данные недостоверны и что количество некорректных элементов равно трем:

```
function testInvalid1()
{
    $this->assertEquals(false, $this->ulBad->isValid(), "invalid ul");
    $this->assertEquals(3, count ($this->ulBad->getInvalidData()),
        "invalid site, section, and login");
}
```

Предыдущие тесты обрабатывали имеющиеся данные, содержащиеся внутри объектов. Однако как проверить следующий этап, т.е. сохранение объектов в базе данных? Для этого, естественно, придется разработать еще несколько тестов. В следующем тесте используется блок try-catch, а также метод assert(). Тест представляет собой попытку отправить в базу данных информацию, содержащуюся в объекте \$ulGood; если попытка удалась, то ошибок нет и утверждение истинно. В противном случае методу assert() передается значение false, а на экран выводится сообщение об ошибке.

Следует заметить, что метод assert() не используется для отображения сообщения об ошибке. Это связано с тем, что желательно получить более подробное описание возникшей ошибки. Поэтому в данном случае используется сообщение, сгенерированное в блоке try:

```
function testGoodPersist0()
{
    try {
        $this->ulGood->persist();
        $this->assert(true);
    } catch (MultiLogException $e) {
        $this->assert(false);
        print ($e->getErrorMessage());
    }
}
```

Теперь следует точно так же проверить объект `$ulBad`. Естественно, ожидается, что проверка выдаст отрицательный результат, поэтому утверждения необходимо соответствующим образом изменить:

```
function testBadPersist0()
{
    try {
        $this->ulBad->persist();
        $this->assert(false); // здесь должен быть сбой
    } catch (MultiLogInvalidDataException $e) {
        $this->assert(true);
    }
}
```

Конечно, программа не должна сохранять один и тот же объект дважды в одной строке. Чтобы протестировать такую ситуацию, следует сформулировать утверждение о том, что вторая попытка сохранения не удалась. После этого вторая попытка сохранения приведет к возникновению исключения, и это можно будет использовать для формулировки правильного утверждения:

```
function testBadPersist1()
{
    try {
        $this->ulGood->persist();
        $this->assert(true);
        $this->ulGood->persist();
        $this->assert(false); // здесь должен быть сбой
    } catch (MultiLogInvalidDatabaseException $e) {
        $this->assert(true);
    } catch (MultiLogDatabaseQueryException $e) {
        $this->assert(false); // неверное утверждение
    }
}
```

А что если приложение работает не так, как должно? Например, в коде была случайно допущена опечатка. Чтобы проверить это, вставим намеренно слово “where” в значение сеанса. При попытке сохранения таких данных должно быть сгенерировано исключение.

В блоке `catch` результат метода `$ul->persist()` ошибочно принимается равным `true`. Это должно привести к отрицательному результату тестирования (вывод будет показан в конце этого раздела):

```
function testBadPersist2_fail()
{
    $ul = new UserLog(array('site' => "1",
                           'section' => "9",
```

```

        'login' => "user101",
        'session' => "1' where '1' and '1')");

    try {
        $ul->persist();
        $this->assert(false); // здесь должен быть сбой
    } catch (MultiLogDatabaseQueryException $e) {
        $this->assert(true);
    }
}

```

Это, конечно, не единственный способ использования утверждений. В следующем примере показано, как вставить в метод `assert()` любой оператор — при условии, что он возвращает булево значение. Кроме того, в этом примере метод `assert()` используется для вывода сообщения об ошибке.

В целях эксперимента сформируем некорректный SQL-оператор. Например, можно заменить слово “insert” на “inert”:

```

function testSqlGeneration0()
{
    $this->assert(
        strpos (" ".$this->ulGood->toSQL(),
            "insert into user_log(visit date,
            visit_time, site_id, demo_id, login_id, session,
            firstname,
            lastname, address1, address2, city, state, zip )" ) > 0,
            "invalid SQL generation first");

    $this->assert(
        strpos (" ".$this->ulGood->toSQL(), "1, 2, '3', '1E23553', 'Alice',
            'AppleGate', '123Main', ", 'Sandusky', 'OH',
            '44870' )" ) > 0, "invalid SQL generation last");
}

```

Теперь проверим два других метода — функции `toHTML()` и `getDemographics()`. В формулировке утверждений для этого теста нет ничего нового:

```

function testHtmlGeneration0 ()
{
    $this->assert(strpos (" ".$this->ulGood->toHTML(), "<tr") > 0);
    $this->assert(strpos ($this->ulGood->toHTML(), ">AppleGate</td><td")
        > 0);
}

function testDemo0 ()
{
    $this->assertEquals(4, count($this->ulGood->getDemographics()));
    $this->assertEquals(4, count($this->ulBad->getDemographics()));
}

```

В последнем тесте заново создается база данных, снова сохраняется объект `$ulGood`, а затем база данных опрашивается вручную, чтобы определить количество возвращенных значений. Этот тест подтверждает результаты предыдущего теста:

```

function testDemo1 ()
{
    include ("../logs/initialize.php"); // заново создаем базу

    try {
        $this->ulGood->persist();
        $this->assert(true);
    }
}

```

```

    } catch (MultiLogException $e) {
        $this->assert(false);
        print ($e->getErrorMessage());
        return;
    }

    $this->db = LogUtils::openDatabase();
    $query = LogUtils::executeQuery ($this->db,
        "SELECT COUNT(*) FROM user_demographics");

    $row = LogUtils::getQueryArray($query);
    $this->assertEquals ("4", $row[0]);
}
}

```

Все тесты разработаны, поэтому можно объединить их в тестовый комплект, так чтобы они выполнялись все вместе:

```

$suite = new TestSuite;
$suite->addTest(new TestUserLog("testValid0"));
$suite->addTest(new TestUserLog("testInvalid0"));
$suite->addTest(new TestUserLog("testInvalid1"));
$suite->addTest(new TestUserLog("testGoodPersist0"));
$suite->addTest(new TestUserLog("testBadPersist0"));
$suite->addTest(new TestUserLog("testBadPersist1"));
$suite->addTest(new TestUserLog("testBadPersist2_fail"));
$suite->addTest(new TestUserLog("testSqlGeneration0"));
$suite->addTest(new TestUserLog("testHtmlGeneration0"));
$suite->addTest(new TestUserLog("testDemo0"));
$suite->addTest(new TestUserLog("testDemo1"));

```

Следующий код создает новый объект класса `TestRunner` и вызывает метод `run()`, который выполняет все тесты и отображает их результаты:

```

$testRunner = new TestRunner();
$testRunner->run( $suite );

```

Первый тестовый комплект готов, и можно проанализировать его результаты. Откройте в браузере страницу `test.UserLog.php` (рис. 17.5) и перейдите вниз — отображается ошибка, сгенерированная `testBadPersist2_fail()`. Следует заметить, что `PHPUnit` распечатывает стандартные сообщения `ok` и `FAIL` зелеными и красными буквами соответственно в зависимости от того, как выполнялся тест: так, как ожидалось, или нет.

Следующий тестовый класс создается так же, как и предыдущий — объявляются классы, к которым необходим доступ:

```

<?php

require_once ("../settings-test.php"); // это важно!
require_once ("class.UserLog.php");
require_once ("class.LogContainer.php");
require_once ("phpunit/phpunit.php");

```

Так как тестируется класс `LogContainer`, необходимо включить несколько `UserLog`-объектов для проверки внутри контекста. В этом примере создается и сохраняется в базе три новых `UserLog`-объекта:

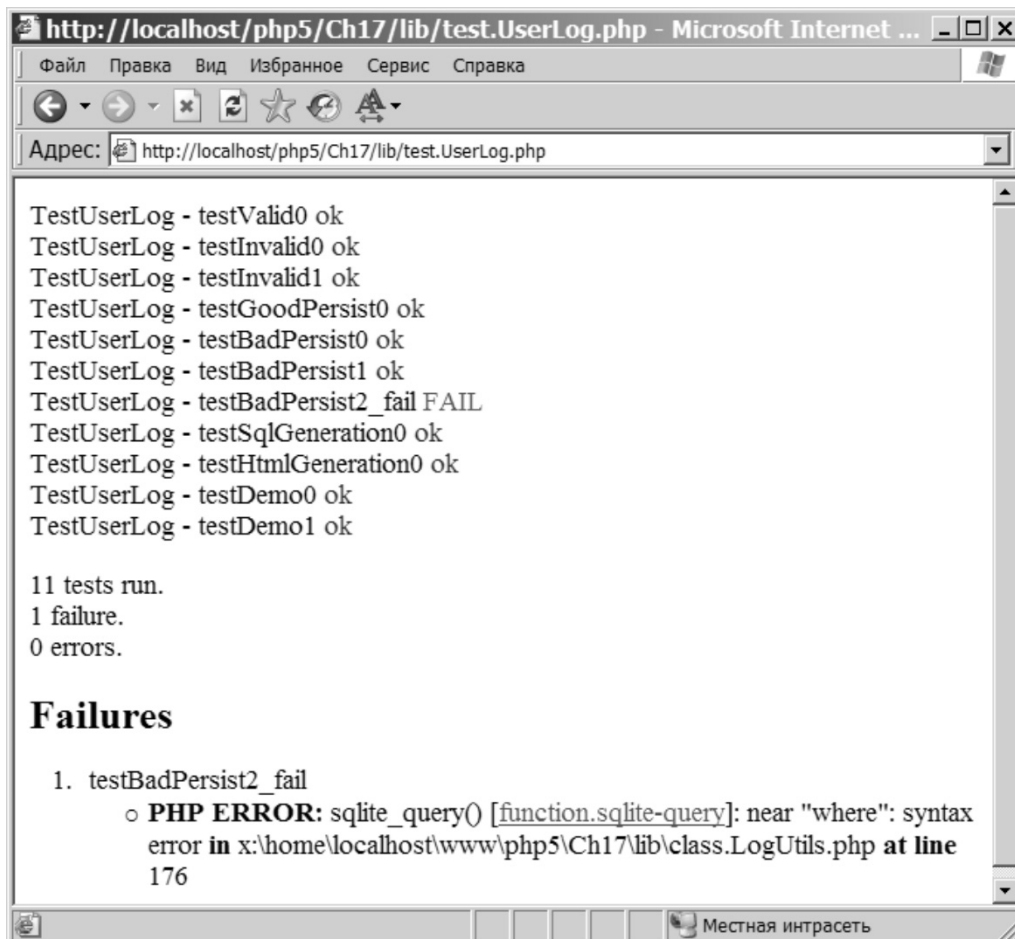


Рис. 17.5.

```

class TestLogContainer extends TestCase {
    function setUp()
    {include ("../logs/initialize.php"); // заново создаем базу
      $ul0 = new UserLog (
        array ('site'      => "1",
              'section'   => "111",
              'login'     => "aapple",
              'session'   => "1E23553",
              'firstname' => "Aurthor",
              'lastname'  => "Andersen",
              'address1'  => "123 Main St.",
              'city'      => "Sandusky",
              'state'     => "OH",
              'zip'       => "44870",
              'demo0'     => "21",
              'demo1'     => "15",
              'demo2'     => "22",
              'demo3'     => "26"));
      $ul0->persist();
    }
}

```

```

        $ul1 = new UserLog (
            array ('site'      => "1",
                  'section'   => "112",
                  'login'     => "bbarker",
                  'session'   => "3F398",
                  'firstname' => "Blob",
                  'lastname'  => "Barker",
                  'address1'  => "100 Hollywood Blvd.",
                  'city'      => "LA",
                  'state'     => "CA",
                  'zip'       => "90036",
                  'demo0'     => "78",
                  'demo1'     => "21",
                  'demo2'     => "12",
                  'demo3'     => "7"));
        $ul1->persist();

        $ul2 = new UserLog (
            array ('site'      => "2",
                  'section'   => "200",
                  'login'     => "ccabbage",
                  'session'   => "L33T",
                  'firstname' => "Capitan",
                  'lastname'  => "Cabbage",
                  'address1'  => "55 Broadway",
                  'city'      => "NYC",
                  'state'     => "NY",
                  'zip'       => "10001",
                  'demo0'     => "14",
                  'demo1'     => "15",
                  'demo2'     => "41",
                  'demo3'     => "0"));
        $ul2->persist();
    }

```

Имея специфический контекст, можно создать тест. Для этого примера включается только один тест. Он создает новый LogContainer-объект, содержащий все UserLog-объекты, поступающие с сайта 1:

```

function testContainerValid0()
{
    try {
        $lsc = new LogContainer ("", "", "1", "");
    }
}

```

Зная контекст, сформулируем утверждение о том, что внутри LogContainer содержится только два UserLog-объекта:

```

$this->assertEquals(2, $lsc->getCount(), "count");

```

Затем создается утверждение о том, что два UserLog-объекта, содержащиеся в LogContainer, имеют регистрационные имена aapple и bbarker соответственно:

```

        $logs = $lsc->getUserLogs();
        $this->assert(strcmp ($logs[0]->getLogin, "aapple"),
            "login_id incorrect a");
        $this->assert(strcmp ($logs[0]->getLogin, "bbarker"),
            "login_id incorrect b");
    } catch (MultiLogException $e) {
        $this->assert(false);
        print $e->getErrorMessage();
    }

```

```

    }
}
}

```

Наконец, создается и запускается тестовый комплект:

```

$suite = new TestSuite;
$suite->addTest(new TestLogContainer("testContainerValid0"));

$testRunner = new TestRunner();
$testRunner->run( $suite );

```

В браузере результат работы этого сценария должен выглядеть аналогично рис. 17.6.

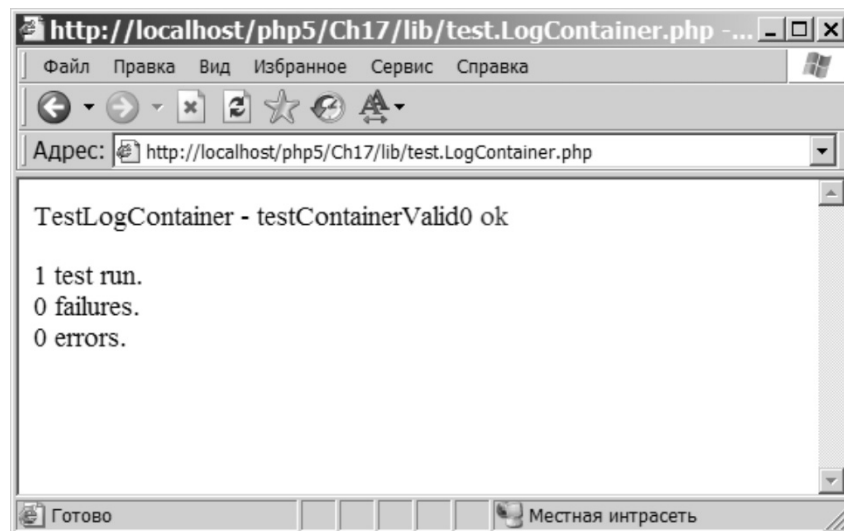


Рис. 17.6.

Последний тестовый класс проверяет демографические данные в UserLog-объектах. Сначала подключаются необходимые файлы:

```

<?php

require_once ("../settings-test.php");
require_once ("class.UserDemographic.php");
require_once ("phpunit/phpunit.php");

```

Объявление класса начинается с определения переменных, в которых будут храниться UserDemographic-объекты. Кроме того, объявляется функция initializeDb(), которая будет использоваться для повторного заполнения базы данных — это позволит переустанавливать контекст:

```

class TestUserDemographic extends TestCase
{
    private $demoGood = null;
    private $demoBad = null;

    function initializeDb () {
        include "../logs/initialize.php";
    }
}

```

Затем контекст инициализируется — следует заметить, что демографические данные содержатся только в последних элементах массива, поэтому можно спокойно игнорировать первые элементы каждого UserLog-объекта:

```
function setUp()
{
    $this->demoGood = new UserDemographic (
        array ('id'      => 1,           // игнорируется
              'site'     => 1,           // игнорируется
              'section'  => 2,           // игнорируется
              'login'    => "3",         // игнорируется
              'session'  => "1E23553",  // игнорируется
              'firstname'=> "Alice",     // игнорируется
              'lastname' => "AppleGate", // игнорируется
              'address1' => "123Main",   // игнорируется
              'city'     => "Sandusky",  // игнорируется
              'state'    => "OH",        // игнорируется
              'zip'      => "44870",     // игнорируется
              'demo'     => "21",
              'demo1'    => "15",
              'demo2'    => "22",
              'demo3'    => "26"));

    $this->demoBad = new UserDemographic (
        array ('site'     => 1,           // игнорируется
              'section'  => 2,           // игнорируется
              'login'    => "3",         // игнорируется
              'session'  => "1E23553",  // игнорируется
              'firstname'=> "Alice",     // игнорируется
              'lastname' => "AppleGate", // игнорируется
              'address1' => "123Main",   // игнорируется
              'city'     => "Sandusky",  // игнорируется
              'state'    => "OH",        // игнорируется
              'zip'      => "44870",     // игнорируется
              'demo1'    => "15",
              'demo2'    => "22",
              'demo3'    => "26"));
}
```

Первый тест для UserDemographic-объектов выглядит так:

```
function testValid0()
{
    $this->assertEquals(null, $this->demoGood->city, "invalid city");
    $this->assertEquals("21", $this->demoGood->demo, "invalid demo0");
    $this->assertEquals(null, $this->demoGood->demo1, "invalid demo1");
    $this->assertEquals(null, $this->demoGood->demo2, "invalid demo2");
    $this->assertEquals(null, $this->demoGood->demo3, "invalid demo3");
    $this->assertEquals(null, $this->demoGood->demo4, "invalid demo4");
    $this->assertEquals(true, $this->demoGood->isValid(), "valid log");
    $this->assertEquals(0, count ($this->demoGood->getInvalidData()),
        "valid count");
}
```

Приведенный ниже тест позволяет убедиться, что предположительно недостоверные данные действительно недостоверны:

```
function testInvalid0()
{
    $this->assertEquals(false, $this->demoBad->isValid(), "invalid log");
    $this->assertEquals(1, count ($this->demoBad->getInvalidData()),
        "expecting 1 invalid item");
}
```

Наконец, создается тест для проверки возможности сохранения объекта в базе данных:

```
function testGoodPersist0()
{
    $this->initializeDb();
    try {
        $this->demoGood->persist();
        $this->assert(true);
    } catch (MultiLogException $e) {
        $this->assert(false);
        print ($e->getErrorMessage());
    }
}
```

В конце используется уже знакомый код:

```
$suite = new TestSuite;
$suite->addTest(new TestUserDemographic("testValid0"));
$suite->addTest(new TestUserDemographic("testInvalid0"));
$suite->addTest(new TestUserDemographic("testGoodPersist0"));

$testRunner = new TestRunner();
$testRunner->run( $suite );
```

Результат работы тестов показан на рис. 17.7.

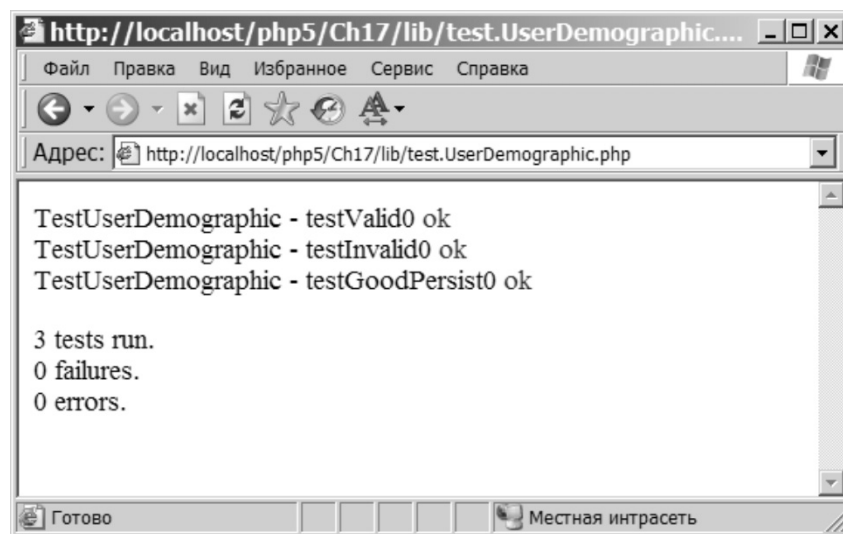


Рис. 17.7.

Стоит отметить, что пакет PHPUnit содержит множество других функций. В главе были описаны функции, с помощью которых можно начинать работу с пакетом, но можно использовать и другие функции для создания необходимых тестов — изучение этих функций остается в качестве упражнения читателям.

Работа с диспетчером протоколирования

Наконец, диспетчер протоколирования готов. Для удобства использования в приложении имеется индексная страница, с которой можно инициализировать базу данных, запускать все тесты, записывать информацию в базу данных и просматривать отчеты. Однако прежде чем использовать приложение, необходимо рассмотреть один файл, который связывает все приложение в единое целое: `userlog.php`.

userlog.php

Сценарий `userlog.php` отвечает за непосредственный сбор данных запроса. Он принимает строку запроса и создает новый `UserLog`-объект, который затем записывается в базу данных — при условии, что информация, переданная в URL, проходит функции проверки данных. Все это делается в блоке `try-catch`, который используется для генерирования одного из исключений `MultiLogExceptions()` в случае возникновения каких-либо ошибок:

```
<?
require_once ("settings.php");
require_once ("lib/class.UserLog.php");

try {
    $ul = new UserLog ($_REQUEST);
    $ul->persist();
} catch (MultiLogException $e) {
    print $e->getErrorMessage();
    print "<h3>Информация phpinfo:</h3>";
    phpinfo(); // небезопасно
}

?>
```

Каждый сайт может передавать информацию с помощью URL-строки:

```
http://logging_site/userlog.php?site=101&section=1&login=user101&sessionid=1E23553&firstname=Alice&lastname=AppleGate&address1=123Main&city=Sandusky&state=OH&zip=44870
```

Для обработки данных можно использовать методы, а поскольку необходимо сохранить данные (т.е. записать их в базу), следует вызвать для этих данных метод `persist()`.

Просмотр интерфейса диспетчера протоколирования

Откройте в Web-браузере индексную страницу диспетчера протоколирования (рис. 17.8).

После щелчка по ссылке Инициализация базы данных на странице отобразится сообщение Готово (при успешном выполнении). Выше уже описывались результаты запуска трех тестов, поэтому нет необходимости проверять эти ссылки снова. Щелкните по ссылке Отправка журналов.

В результате на странице отобразится полная распечатка двух `UserLog`-объектов, `UserDemographic`-объектов и ссылка для просмотра отчетов (подтверждающая, что эти объекты были записаны в базу данных), как показано на рис. 17.9.

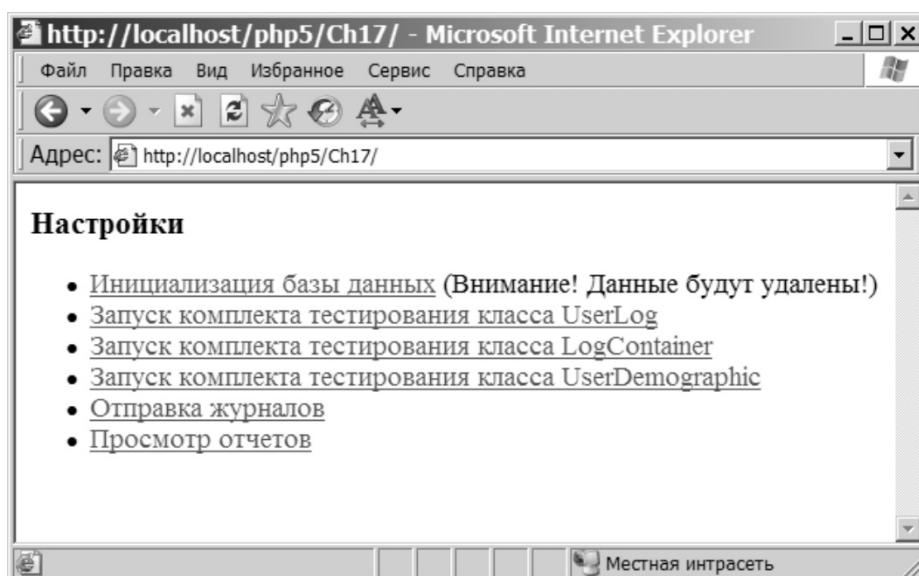


Рис. 17.8.

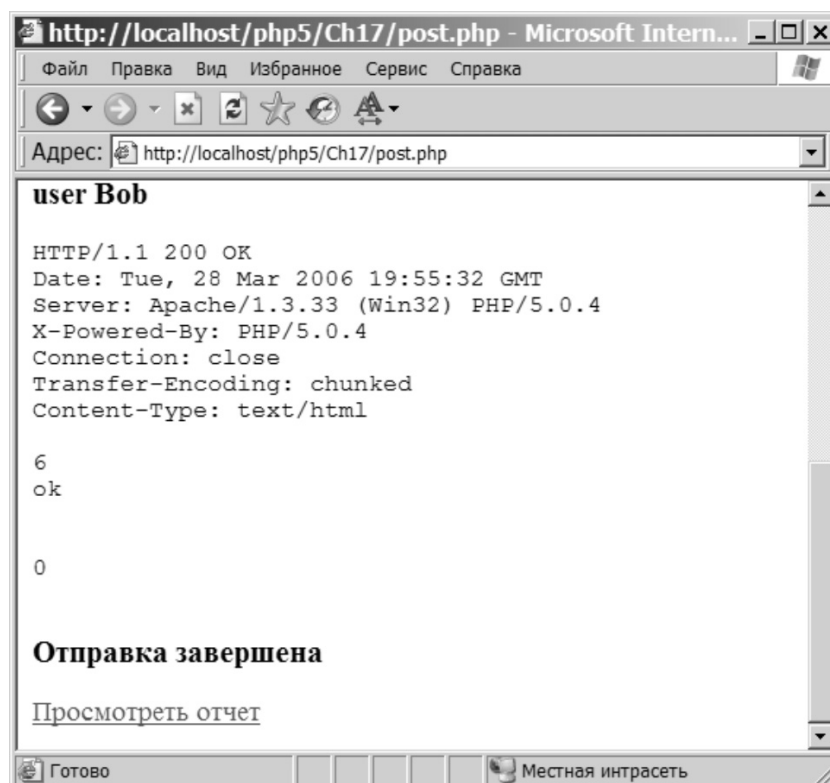


Рис. 17.9.

Щелкните по ссылке Просмотр отчетов. Выберите даты, сайт и разделы, для которых необходимо просмотреть отчеты. На рис. 17.10 показан пример.

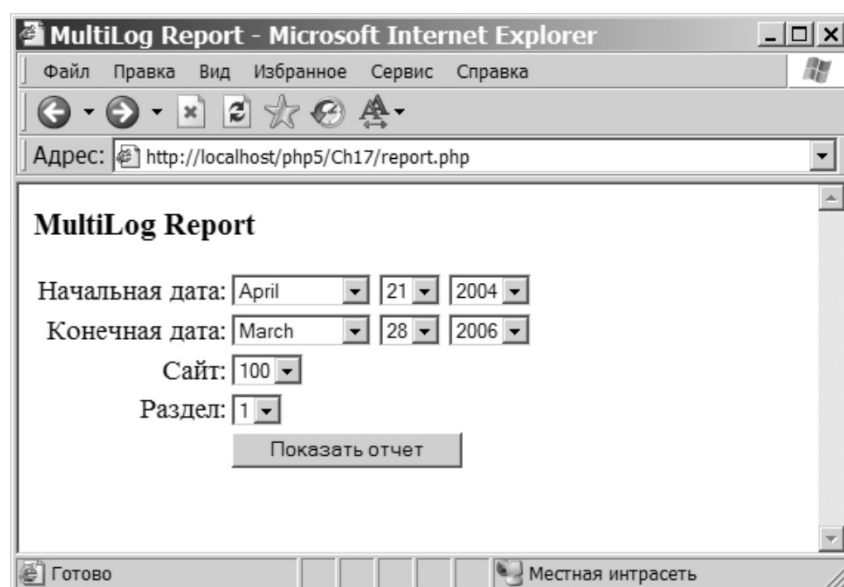


Рис. 17.10.

Результаты должны выглядеть аналогично рис. 17.11.

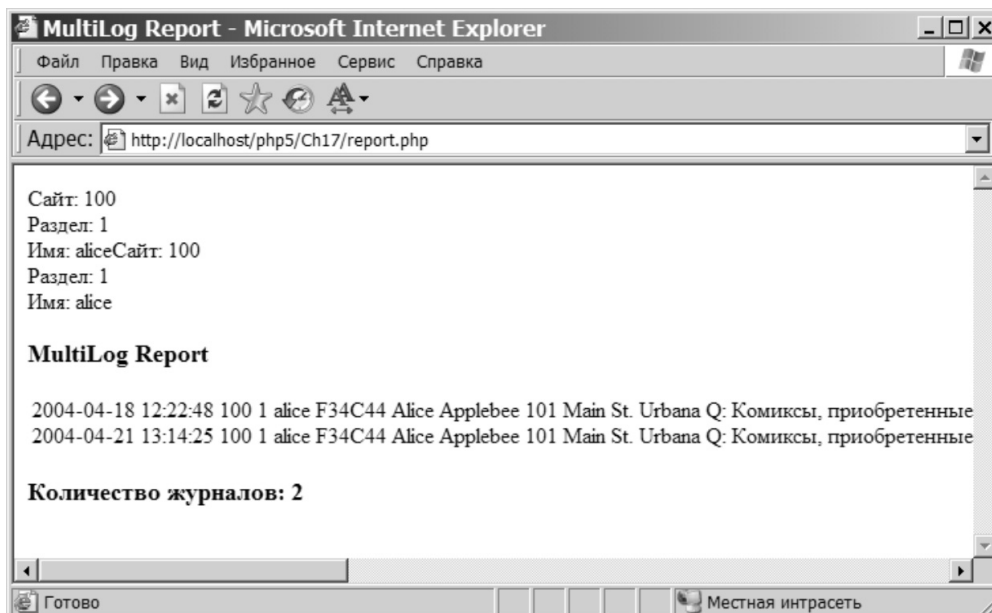


Рис. 17.11.

Можно убедиться, что собранная информация записывается в базу данных. Модифицируйте URL, упомянутый в “*userlog.php*”, так, чтобы он отражал некоторую персональную информацию, и передайте его в браузер. Например:

```
http://logging_site/userlog.php?site=100&section=1&login=user101&sessionid=2D56553&firstname=David&lastname=Mercer&address1=65SomePlace&city=CapeTown&state=WP&zip=8001
```

В случае успеха должно появиться краткое подтверждение (рис. 17.12).

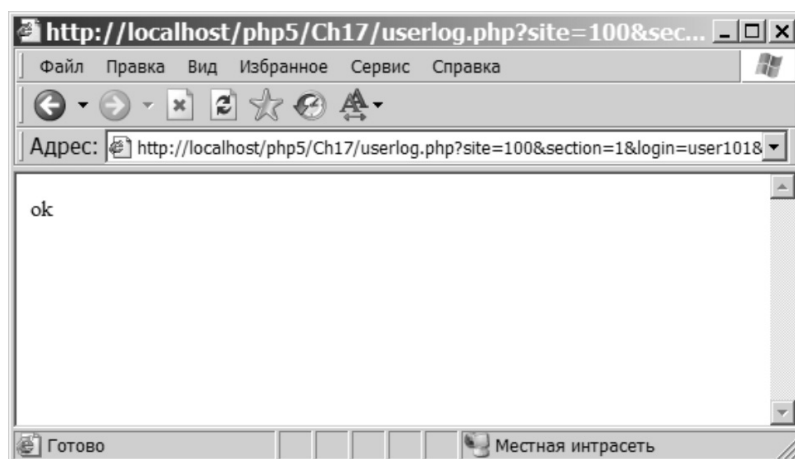


Рис. 17.12.

Теперь вернитесь к странице отчетов и проверьте, записался ли данный журнал в базу данных. На рис. 17.13 показаны ожидаемые результаты.

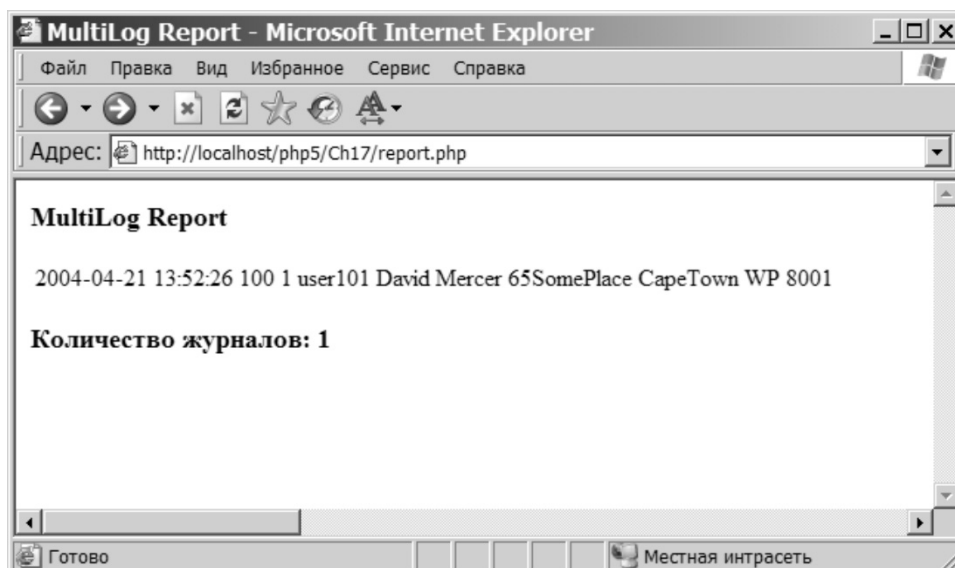


Рис. 17.13.

Это все. Теперь можно записывать запросы к диспетчеру протоколирования от различных сайтов, с которых необходимо собирать информацию. Естественно, это приложение можно модифицировать, чтобы:

- ☐ получать данные разных типов, а не только демографические;
- ☐ отображать более сложные отчеты;
- ☐ реализовать более развитую обработку исключений и проверку достоверности данных.

Резюме

В отличие от остальных глав данной книги, в каждой из которых освещался один аспект PHP-программирования, в этой главе подробно рассмотрено завершенное приложение. Читатель познакомился с различными проблемами, которые приходится решать во время создания более сложных приложений. В этой главе представлены многочисленные практические приемы программирования и объектно-ориентированный подход, при этом акцент делается на эффективные методики проверки достоверности данных и обработки исключений.

В ходе анализа учебного примера рассматривался полный цикл разработки приложения, диспетчера протоколирования. Здесь были представлены новые технологии, которые впоследствии разработчик может использовать для усовершенствования своих программ. В частности, в главе рассматривалась система Smarty, позволяющая выделить в приложении уровень представления, а также пакет PHPUnit, обеспечивающий блочное тестирование.

UML-диаграммы классов способствовали наглядному представлению основных классов учебного примера, а диаграммы последовательностей продемонстрировали планируемый поток информации и различные уровни приложения. В главе также рассматривалась фактическая реализация всего кода блок за блоком. Приложение было полностью протестировано и проверено в действии.

Досконально изучив эту главу, вы сможете работать уже над более сложными проектами. Конечно, кривая изучения никогда не заканчивается (а часто становится круче), но работа будет интересной, а награда достойной.



Ответы

Упражнения, представленные в этой книге, способствуют закреплению новых знаний и усовершенствованию полученных навыков. Это приложение содержит решения упражнений по главам. Фрагменты кода вы найдете здесь либо сможете загрузить с сайта www.wrox.com. Представленный здесь код также доступен на этом сайте.

Многие упражнения допускают несколько возможных работоспособных решений, поэтому ваше собственное решение может отличаться от представленного в приложении. Решения задуманы как типовые, и в некоторых случаях код написан скорее как удобочитаемый пример, а не как самый быстрый и кратчайший метод достижения результата. Некоторые из упражнений, особенно в первых главах книги, вообще не содержат кода либо включают в себя фрагменты псевдокода или планов. Однако все упражнения в книге важны, поскольку они знакомят программиста с процессом создания программного обеспечения.

Глава 1

Упражнение

Предлагаемое упражнение будет полезным всякий раз при установке или работе с PHP на новой платформе.

Чтобы выполнить данное упражнение, необходимо создать документ, в котором собраны все следующие сведения:

- ☐ Каковы аппаратные возможности компьютера, на котором работает PHP? Опишите процессор, жесткий диск, оперативную память и т.д., а также любые предполагаемые ограничения.
- ☐ Какая операционная система работает на данном оборудовании? Укажите версию, а также все текущие заплатки и известные дефекты.
- ☐ Какой Web-сервер работает на данной машине? Укажите версию, заплатки и известные дефекты. Кроме того, опишите конфигурацию Web-сервера, корневой

каталог, то, как PHP настраивается для работы с сервером, а также права доступа, которые пришлось установить.

- ☐ Какая версия PHP была установлена? Укажите версию, установленные файлы, каталоги, в которые они были установлены, а также все параметры реестра, настроенные или созданные для поддержки инсталляции PHP.
- ☐ Какие конфигурационные установки были заданы или изменены (по сравнению со стандартными) для установки PHP? Перечислите их.
- ☐ Какие расширения были включены? Перечислите их все, а также укажите причины, по которым вы их включили.

Наличие подобного документа в будущем в случае возможных затруднений значительно облегчит поиск всей информации (поскольку вся она будет сосредоточена в одном месте), а также поможет другим программистам понять, почему PHP-программы работают именно так.

Решение

Создайте для своей системы новый документ, в котором будут содержаться следующие сведения:

<i>Документация — Сервер01</i>	
Процессор – жесткий диск – оперативная память	Pentium 4, 1,7 ГГц – 120 Гб – 512 Мб
Операционная система	Red Hat Linux Fedora
Web-сервер	Apache 2.0
Интерпретатор сценариев	PHP 5.0
Конфигурационные параметры PHP	register_globals = off, стандартные настройки для разработки сценариев

Глава 2

Упражнение 1

1. Создайте PHP-программу, которая трансформирует первое предложение во второе, и выводит результат. Оба предложения представлены ниже:
 - А. Теперь пора всем хорошим людям прийти на помощь стране.
 - Б. Пора теперь стране прийти на помощь всем хорошим людям.

Решение

Напишите такой PHP-сценарий:

```
<?php
$my_array = explode(" ", "теперь пора всем хорошим людям прийти на помощь стране");
echo $my_array[1] . " " . $my_array[0] . " " . $my_array[8] . " " . $my_array[5] . " " . $my_array[6] . " " . $my_array[7] . " " . $my_array[2] . " " . $my_array[3] . " " . $my_array[4];
?>
```

Упражнение 2

1. Напишите PHP-программу, которая создает два массива чисел и прибавляет значения одного массива к соответствующим (по индексу) значениям другого массива. Массивы должны содержать следующие значения:

А. 2, 4, 6, 8, 10;

Б. 3, 5, 7, 9, 11.

Решение

Напишите такой PHP-сценарий:

```
<?
$arr01 = array(2,4,6,8,10);
$arr02 = array(3,5,7,9,11);

for ($counter = 0; $counter <= 4; $counter ++){
    $arr03[] = $arr01[$counter] + $arr02[$counter];
    echo "Ответ " . "$arr03[$counter]" . "<br>";
}
?>
```

Глава 3

Упражнение

В PHP имеется очень полезная функция `isset()`, которая сообщает PHP-программе, установлена ли определенная переменная. Например, предположим, что существует страница с формой, содержащей submit-кнопки `login` и `logout`. С помощью функции `isset()` можно узнать, какая кнопка была нажата:

```
if (isset($login)) {
    //делаем что-то
} elseif (isset($logout)) {
    //делаем что-то другое
}
```

В этом упражнении требуется создать Web-страницу с формой, которая отправляет данные самой себе, и заставить PHP-программу сообщать об отправке формы, используя функцию `isset()`. Если форма не была отправлена, то программа должна отобразить форму (без сообщения), запрашивающую имя и фамилию пользователя. Иначе программа должна вместо формы отобразить сообщение (короткое предложение, например: “Ваше имя XX, а фамилия YY”).

Совет: чтобы определить, была ли отправлена форма, следует использовать скрытое поле, а для того чтобы заставить форму отправлять данные самой себе, можно использовать переменную `$PHP_SELF`.

Решение

```
<html><head><title>Заголовок страницы</title></head>
<body>
<?php
if (isset($_POST['posted'])) {
    echo "Спасибо. Ваше имя - $_POST[first_name], фамилия $_POST[last_name]";
} else {
    ?>
```

```
<h2>Пожалуйста, заполните форму</h2>
<form method="POST" action="<?php echo "$_SERVER[PHP_SELF]"; ?>">
<input type="hidden" name="posted" value="true">
Имя<input type="text" name="first_name"><br>
Фамилия<input type="text" name="last_name">
<input type="submit" value="Отправить информацию">
</form>
<?php
}
?>
</body>
</html>
```

Глава 4

Упражнение

“У нас есть Web-сайт, но информация на нем устарела. Он выглядит не очень привлекательно, но мы уже наняли дизайнера, которой переделает логотип, поэтому как только он закончит свою работу, трафик, вероятно, возрастет. Мы собираемся нанять персонал для поддержки растущих запросов и хотим разместить на сайте программу для сбора резюме.”

“Посетители должны легко находить на сайте перечень вакансий, но эта ссылка не должна быть главным элементом страницы. Мы должны предоставить пользователю возможность искать работу и оставлять свою контактную информацию на сайте. Те, кто не имеет высшего образования, могут претендовать только на работу начального уровня в отделе продаж или отделе доставки. Зарботные платы ни по одной из вакансий для начала не превышают 20 000 долларов, на руководящих постах предлагаются поощрительные премии. Соискатели должны отправлять свои пожелания по заработной плате и иметь минимум двухлетний опыт работы. Желающие получить работу на руководящей позиции, в том случае если они не имеют докторской степени, должны иметь, по крайней мере, пятилетний опыт работы. Мы бы хотели, чтобы люди могли искать на сайте работу и подавать заявки на интересующие их позиции, а также чтобы они могли отправлять свои резюме и получать информацию о том, на какую работу они могут претендовать.”

Именно так часто выглядят первоначальные требования к приложению со стороны заказчиков: малознакомый с программированием заказчик просит разработать программу для выполнения определенной функции.

Выполнение этого упражнения поможет получить практические навыки, вы узнаете о том, как, начиная с формулировки проблемы, двигаться к созданию законченного продукта. Задача в данном случае заключается в том, чтобы изложить требования заказчика в более четкой форме и использовать возможности РНР для выполнения сбора и обработки информации, а также для выдачи ответа пользователям данного Web-сайта. Чтобы выполнить упражнение, в дополнение к законченной программе следует включить следующие пункты:

- ☐ перечень всей необходимой информации в дополнение к уже представленной;
- ☐ описание Web-страниц, которые вы будете разрабатывать, причины их разработки (зачем эти страницы нужны) и то, как пользователи будут взаимодействовать с ними;
- ☐ краткое описание интеграции созданных страниц с существующим Web-сайтом.

Для этого упражнения нет единого программного решения, написанного на РНР, хотя все решения выполняют подобную обработку данных и используют такие же этапы. Базы данных и файловые системы в книге еще не рассматривались. Поэтому пока читатель не владеет способами длительного сохранения резюме, переданная пользователем информация будет потеряна после завершения обработки.

Задание описано — можно приступать к проектированию.

Совет: чтобы разработать программное решение, попытайтесь сначала описать проблему как последовательность Web-страниц и задать себе вопрос, что пользователь хотел бы увидеть. Вспомните уже описанные средства РНР, которые можно было бы применить для получения информации, а также точно опишите данные, необходимые для выполнения следующего этапа. Создать работоспособное решение гораздо проще путем разбиения всего цикла разработки на небольшие этапы.

Решение

Сначала напишите список необходимой информации. Очевидно, что заказчик хочет собирать резюме. Он, скорее всего, не собирается регистрировать пользователей для доступа к сайту, поэтому можно выделить несколько основных категорий информации.

1. Контактная информация (имя, адрес, номера телефонов и т.д.). Во многих случаях от пользователя также требуется ввести идентификационный код.
2. Опыт работы.
3. Образование.
4. Специальная информация о квалификации (ученая степень, сертификаты и т.д.).
5. Дополнительная информация.

В требованиях также описана возможность поиска работы. Из этого следует, что нужно составить список внутренних работ, актуальность которого поддерживается одним из сотрудников организации заказчика. Этот сотрудник должен вводить поисковые фразы (ключевые слова), применимые к каждому виду работ. Также следует уточнить, нужно ли реализовать возможность поиска работы по заданному диапазону зарплаты.

Есть несколько условий, которые касаются поиска работы и отправки резюме. Первое условие: люди без высшего образования могут претендовать лишь на работу начального уровня в отделе продаж или доставки. Надо ли давать им возможность начинать поиск работы, прежде чем они ответят на некоторые отборочные вопросы (зачем показывать этим людям список работ, к которым они не могут быть допущены?).

Второе условие: соискатели должны иметь, по крайней мере, двухлетний опыт работы для всех вакансий, кроме вакансий управляющего состава, для которых необходимо иметь пятилетний опыт работы или ученую степень.

Указание зарплаты не требует специального программирования, кроме того, что стоит выводить предупреждение для потенциальных работников о том, что зарплата в компании не очень высокая, хотя менеджеры могут получать премии.

Затем можно начинать разделять процесс взаимодействия пользователей с системой. Скорее всего, сначала, чтобы предварительно отобрать соискателей, необходимо выдавать пользователям запрос на некоторую основную информацию, затем дать им возможность искать работу или просто отправить свое резюме, а после этого отобразить перечень работ, к которым они могут быть допущены.

Для этого можно отобразить приветственный экран, приглашающий ответить на некоторые предварительные вопросы, а затем вывести страницу, предлагающую сделать выбор: искать работу или отправить резюме. Этап подачи резюме можно разделить на два шага, так чтобы пользователи не вводили всю информацию в одну огромную форму на одной странице.

После того как форма заполнена и отправлена, приложение должно сохранять информацию между запросами страниц и заполнять поля соответствующей информацией снова, на случай если пользователь захочет вернуться и изменить введенную им информацию.

Итак, рассмотрим страницы, из которых может состоять приложение.

1. Приветственная страница с формой для ввода предварительной информации и ссылка на страницу “Поиск работы/Отправка резюме”.
2. Страница “Поиск работы/Отправка резюме” с формой для ввода информации резюме (разделенной на логические блоки), а также с текстовым полем и кнопкой для поиска.

Рассмотрим логику соответствующей PHP-программы.

- ☐ Отобразить приветственную страницу.
- ☐ Запросить у пользователя предварительную информацию, например, опыт работы, уровень образования и любые другие подходящие сведения (многие вакансии имеют возрастные ограничения и определенные требования, касающиеся места жительства; все эти требования можно получить от заказчика).
- ☐ Сохранить предварительную информацию или сообщить пользователю о том, что он не подходит ни на одну вакансию. Из-за недостатка устройств долговременного хранения данных (в реальных приложениях такого недостатка нет) придется сохранять данные в скрытых полях формы.
- ☐ В зависимости от того, каким квалификационным требованиям удовлетворяет пользователь, отобразить соответствующую форму для отправки резюме и поисковую форму.
- ☐ После отправки каждой части резюме сохранить данные в скрытых полях формы и предоставить пользователю навигационную ссылку, позволяющую при желании вызвать форму для редактирования ранее введенной информации.
- ☐ Создать функцию поиска, отображающую на основании введенных пользователем поисковых фраз перечень работ со ссылками. Эта функция будет зависеть от выборки списка работ из базы данных или файла, но можно предположить, что эта информация будет доступна в форме массива, поэтому все, что требуется сделать, это выполнить поиск по массиву с помощью цикла `for` или `foreach` и найти ключевое слово в массиве, используя PHP-функцию `substr()` для сопоставления введенных поисковых фраз с ключевыми словами.
- ☐ Наконец, если пользователь успешно ввел резюме и претендует на определенную работу, то приложение отображает страницу с благодарностью.

Сравнительно простое приложение с описанным здесь процессом можно полностью реализовать в одном файле, используя простые HTML-формы и Web-страницы, оператор `switch` для каждого случая отправки форм и скрытые поля форм для хранения данных между запросами.

Глава 5

Упражнение

Создайте не менее трех регулярных выражений и вставьте их в соответствующие места последнего примера в данной главе. Примерные регулярные выражения:

- ☐ семи- и десятизначные номера телефонов в США;
- ☐ номера социального обеспечения;
- ☐ буква для обозначения пола пользователя (М — для мужчин и Ж — для женщин).

Решение

Существует множество способов проверки телефонных номеров, а библиотека регулярных выражений содержит несколько примеров поиска телефонных номеров США. Простейший способ — поиск в строке только 7 или 10 цифр, исключая алфавитные символы. Регулярное выражение начинается с символа `^`, за которым следует один символ в диапазоне от 2 до 9, два символа от 0 до 9, дефис, а затем один символ от 2 до 9, два символа от 0 до 9, дефис и, наконец, четыре символа от 0 до 9. Это выражение совпадает с телефонным номером, состоящим из 10 цифр и дефисов между сегментами. Можно использовать символ “или” (`|`) и повторить две вторые части первого образца так, чтобы выражение также совпадало с телефонными номерами, состоящими из 7 цифр. В конце регулярного выражения ставится знак доллара (`$`), который соответствует концу строки.

```
^([2-9][0-9]{2}-[2-9][0-9]{2}-[0-9]{4}|[2-9][0-9]{2}-[0-9]{4})$
```

Для проверки номеров социального страхования используется три символа в диапазоне от 0 до 9, дефис, два символа от 0 до 9, дефис и четыре символа от 0 до 9. Простейшее выражение имеет вид:

```
^[0-9]{3}-[0-9]{2}-[0-9]{4}$
```

Хотя это выражение точно соответствует численному формату номеров социального страхования, оно также совпадает со строкой, состоящей из всех нулей, или со строкой 666, но ни та, ни другая строка не входит в достоверный номер социального страхования. Для достижения большей точности необходимо найти способ исключить комбинации со всеми нулями или подстроки 666.

Для определения буквы пола можно использовать классы символов:

```
^([М] | [Ж])$
```

В этом регулярном выражении используется символ крышки и знак доллара, совпадающие соответственно с началом и концом строки, а также символьные классы для буквы М или Ж (используется оператор “или” `|`) верхнего регистра.

Глава 6

Упражнение

Напишите функцию, с помощью которой пользователь сможет создать собственную Web-форму. В форме должны быть предусмотрены возможности выбора имен полей и типов полей, отправки информации в PHP-страницу и просмотра переданных значений. Для выбора различных функций внутри главной функции необходимо использовать оператор `switch..case`.

Решение

Это решение предполагает использование HTML-тегов в PHP. Начать можно с создания Web-формы: пользователь вводит predetermined количество имен полей и типов полей, а затем приложение определяет количество переменных, в которых будут содержаться HTML-теги для этих имен и типов полей. После того как пользователь отправляет форму, на основании введенных им данных генерируется другая форма. Оператор `switch...case` можно поместить внутри функции, полностью генерирующей новую форму, и выбирать с его помощью HTML-код для генерации имени поля и типа поля внутри predetermined HTML-формы.

Также необходимо предусмотреть возможность запрашивать у пользователя дополнительную информацию, если выбран выпадающий список (тег `<select>`), поскольку для работы этих тегов внутри них должны присутствовать также теги `<option>`.

Ниже приведен пример, демонстрирующий создание функции для формирования HTML-тегов на странице:

```
//функция для формирования HTML-тегов
function createTags($field_name,$field_type)
{
    global $option_text01,$option_text02,$option_text03;
    global $option_value01,$option_value02,$option_value03;
    switch ($field_type) {
        case "text";
            $next_field = "<tr><td>$field_name:</td><td><input type='$field_type'
name='$field_name'></td></tr>";
            break;
        case "radio";
            $next_field = "<tr><td>$field_name:</td><td><input type='$field_type'
name='$field_name'></td></tr>";
            break;
        case "checkbox";
            $next_field = "<tr><td>$field_name:</td><td><input type='$field_type'
name='$field_name'></td></tr>";
            break;
        case "hidden";
            $next_field = "<tr><td>$field_name:</td><td><input type='$field_type'
name='$field_name'></td></tr>";
            break;
        case "textarea";
            $next_field = "<tr><td>$field_name:</td><td><textarea cols='40'
name='$field_name'></td></tr>";
            break;
        case "select";
            $next_field = "<tr><td>$field_name:</td><td>
<select name='$field_name'>
                .<option value='$option_value01'>$option_text01</option>
                .<option value='$option_value02'>$option_text02</option>
                .<option value='$option_value03'>$option_text03</option>
                .</select></td></tr>";
            break;
        default;
            break;
    }
    return $next_field;
}
```

Глава 7

Упражнение

Создайте РНР-приложение, которое можно использовать для поиска определенного каталога в правильно заданном родительском каталоге. Приложение должно просматривать указанный каталог, а также все каталоги, которые в нем находятся.

Решение

В этом приложении для поиска каталогов и обработки всех каталогов и подкаталогов в заданном каталоге можно использовать РНР-функции, предназначенные для работы с файловой системой. Сначала можно сформировать список всех каталогов в заданном родительском каталоге, просмотреть все эти каталоги, сверяя имена их подкаталогов с искомым значением, а затем так же просмотреть все подкаталоги и т.д. Ниже приведен код сценария, который ищет каталог внутри каталога, определенного по умолчанию.

```
//получаем имя искомого каталога
$folder_to_find = "$_POST[folder]";
//устанавливаем каталог по умолчанию
$default_dir = "C:";
//определяем функцию для поиска существующего каталога
function find_folder($default_dir,$folder_to_find) {
    if (!(($dp = opendir($default_dir))) {
        die("Невозможно открыть $default_dir.");
    } else {
        while ($file = readdir($dp)) {
            if ($file == $folder_to_find){
                return ($file);
            }
        }
        closedir($dp);
    }
}
$folder = find_folder($default_dir,$folder_to_find);
if ($folder != "") {
    echo "Найден каталог с именем " . $folder;
} else {
    echo "Каталог не найден.";
}
```

Приведенный выше код ищет заданный каталог внутри каталога по умолчанию, но не ищет в его подкаталогах.

Глава 8

Упражнение

Создайте приложение, которое считывало бы XML-файл с несколькими элементами и атрибутами в simpleXML-объект, изменяло бы значения некоторых или всех элементов или атрибутов файла, а затем записывало бы модифицированный документ обратно в файл.

Решение

В книге уже рассматривалось изменение значений в XML-файлах. Приведенное ниже решение представляет собой расширение этой методики. Ниже показан пример кода, изменяющего значения элементов на основании их индексов в массиве:

```
//определяем изменяемый элемент
$element_to_change = $_POST[element_to_change];
$change_value = $_POST[change_value];

if ($element_to_change == 0) {
    $first_xml_string->program[0]->price = $change_value;
} elseif ($element_to_change == 1) {
    $first_xml_string->program[1]->price = $change_value;
}
```

Недостаток этого кода состоит в том, что индексные номера элементов должны быть жестко запрограммированы. Кроме того, этот код не позволяет сохранять результирующий XML-документ обратно в файл.

Глава 11

Упражнение 1

Создайте список подходящих дескрипторов для каждого столбца в таблице.

Совет: помните о том, что ресторанам ежедневно приходится иметь дело с множеством отдельных заказов (счетов):

Имя	Иденти- фикатор заказа	Иденти- фикатор пользователя	Ресторан	Пароль	Фамилия	Всего (\$)	E-mail
David	2	1	Nandos	12345	Mercer	21.45	davidm@contechst.com
David	4	1	Mimos	12345	Mercer	20.95	davidm@contechst.com
Nic	3	2	St Elmos	23212	Malan	15.45	therot@doggiestouch.co.za
Brian	5	4	Spur	32123	Reid	22.00	pads@doggiestouch.co.za
Darren	1	3	Home	43212	Ebbs	11.85	Bacardi@doggiestouch.co.za

Решение

Ниже приведен список подходящих дескрипторов полей:

```
VARCHAR(30)
MEDIUMINT(7)
MEDIUMINT(7)
VARCHAR(30)
VARCHAR(20)
VARCHAR(50)
FLOAT(7)
VARCHAR(60)
```

Упражнение 2

Администратор сайта решил создать страницу, на которой пользователи смогут просматривать свои предыдущие заказы и итоговые суммы. Используя принципы

нормализации, создайте новые таблицы, в каждой из которых будет представлен один элемент. Присвойте каждой таблице подходящее имя и определите первичный ключ. (Проверьте, уменьшат ли эти изменения избыточность при больших объемах данных.)

Решение

Ниже показана структура таблицы Customer.

<i>User_ID</i>	<i>Firstname</i>	<i>Lastname</i>	<i>Password</i>	<i>Email</i>
1	David	Mercer	12345	davidm@contechst.com
2	Nic	Malan	23212	therot@doggiestouch.co.za
3	Darren	Ebbs	43212	Bacardi@doggiestouch.co.za
4	Brian	Reid	32123	pads@doggiestouch.co.za

Далее показана структура таблицы Orders.

<i>Order_ID</i>	<i>User_ID</i>	<i>Restaurant</i>	<i>TotalPrice</i>
1	3	Home	11.85
2	1	Nandos	21.45
3	2	St Elmos	15.45
4	1	Mimos	20.95
5	4	Spur	22.00

Упражнение 3

Создайте данные таблицы с помощью SQL-операторов.

Решение

Следующий SQL-оператор создает в базе данных таблицу Customer:

```
CREATE TABLE Customer (
  User_ID MEDIUMINT(7) NOT NULL AUTO_INCREMENT,
  Firstname VARCHAR(30) BINARY NOT NULL,
  Lastname VARCHAR(50) BINARY NOT NULL,
  Password VARCHAR(20) BINARY NOT NULL,
  Email VARCHAR(60),
  PRIMARY KEY (User_ID),
  UNIQUE (Email)
);
```

SQL-оператор для создания таблицы Orders в базе данных:

```
CREATE TABLE Orders (
  Order_ID MEDIUMINT (7) NOT NULL AUTO_INCREMENT,
  User_ID MEDIUMINT (7) NOT NULL,
  Restaurant VARCHAR (30) NOT NULL,
  Total_Price FLOAT (7) NOT NULL,
  PRIMARY KEY (Order_ID)
);
```

Упражнение 4

После полугодового использования Web-сайта клиенты отметили, что сайт начинает работать медленнее, когда они пытаются просмотреть свои предыдущие заказы. Администратор сайта изучил таблицу заказов и выяснил, что в ней теперь хранится несколько тысяч записей. Почему сайт работает медленно и как администратор может повысить скорость работы сайта? Обновите таблицы соответствующим образом.

Решение

Сайт работает медленно, поскольку в таблице `Orders` содержится большое количество записей. Поиск производится по полю `User_ID`, так как с помощью значений можно определить, какие записи следует возвращать пользователю. Чтобы предотвратить полное сканирование таблицы при поиске записей, следует добавить индекс по полю `User_ID`. (Следует, однако, учитывать, что хотя это повышает скорость выполнения операторов `SELECT`, скорость операторов `INSERT` и `UPDATE` уменьшается. В данном случае это, вероятно, имеет смысл, поскольку позволяет ускорить работу сайта со стороны клиентов.) Повысить скорость можно с помощью следующего запроса:

```
ALTER TABLE Orders ADD INDEX (User_ID);
```

Упражнение 5

Директор решил узнать, насколько популярен среди клиентов ресторан “Nandos”. Какой запрос можно использовать, чтобы получить общие суммы по всем заказам: а) сделанным одним клиентом и б) в целом по ресторану “Nandos”.

Совет: попытайтесь использовать запрос `SELECT SUM(столбец)`.

Решение

(а):

```
SELECT SUM(Total_price) FROM Orders WHERE User_ID = 1;
```

(б):

```
SELECT SUM(Total_price) FROM Orders WHERE Restaurant = 'Nandos';
```

Упражнение 6

Создайте сценарий, с помощью которого пользователи во время регистрации на сайте смогут вводить в базу данных информацию о себе. База данных должна быть настроена так, чтобы идентификаторы пользователей назначались автоматически. Сразу после регистрации пользователь должен иметь возможность разместить заказ в любом из пяти ресторанов. (Для начала только в одном. Беспокоиться о создании завершенного интерфейса не следует — важна возможность вводить цену.) После нажатия на кнопку `Заказать` в таблице заказов должна создаваться новая запись с увеличенным на 1 идентификатором заказа (`Order_ID`) и корректным идентификатором пользователя (`User_ID`). Совет: желательно использовать сеансы для хранения идентификаторов пользователей (`User_ID`) при регистрации.

Решение

`common_db.inc:`

```
<?php
$dbhost = 'localhost';
$dbusername = 'phpuser';
```

```
$dbuserpassword = 'phppass';
$default_dbname = 'sample_db';
$max_menu_items = 10;
$MYSQL_ERRNO = '';
$MYSQL_ERROR = '';

function db_connect($dbname=") {
    global $dbhost, $dbusername, $dbuserpassword, $default_dbname;
    global $MYSQL_ERRNO, $MYSQL_ERROR;

    $link_id = mysql_connect($dbhost, $dbusername, $dbuserpassword);
    if(!$link_id) {
        $MYSQL_ERRNO = 0;
        $MYSQL_ERROR = "Не удалось подключиться к узлу $dbhost.";
        return 0;
    }
    else if(empty($dbname) && !mysql_select_db($default_dbname)) {
        $MYSQL_ERRNO = mysql_errno();
        $MYSQL_ERROR = mysql_error();
        return 0;
    }
    else if(!empty($dbname) && !mysql_select_db($dbname)) {
        $MYSQL_ERRNO = mysql_errno();
        $MYSQL_ERROR = mysql_error();
        return 0;
    }
    else return $link_id;
}

function sql_error() {
    global $MYSQL_ERRNO, $MYSQL_ERROR;
    if(empty($MYSQL_ERROR)) {
        $MYSQL_ERRNO = mysql_errno();
        $MYSQL_ERROR = mysql_error();
    }
    return "$MYSQL_ERRNO: $MYSQL_ERROR";
}
?>
```

Takeaway.php:

```
<?php
include_once "../common_db.inc";

function login_form() {
    global $PHP_SELF;
    ?>
    <HTML>
    <HEAD>
    <TITLE>Вход в систему</TITLE>
    </HEAD>
    <BODY>
    <FORM METHOD="GET" ACTION="<?php echo $PHP_SELF ?>">
    <INPUT TYPE="HIDDEN" NAME="action" VALUE="register">
    <DIV ALIGN="CENTER"><CENTER>
        <H2>Добро пожаловать в компанию "Быстрая доставка"!</H2>
        <H3>Пожалуйста, зарегистрируйтесь.</H3>
        <TABLE BORDER="1" WIDTH="400" CELLPADDING="2">
            <TR>
                <TH WIDTH="25%" ALIGN="RIGHT" NOWRAP>Имя</TH>
                <TD WIDTH="82%" NOWRAP>
                    <INPUT TYPE="Input" NAME="userfirstname" SIZE="30">
                </TD>
            </TR>
        </TABLE>
    </DIV>
    </BODY>
    </HTML>
```

```

        <TR>
        <TH WIDTH="25%" ALIGN="RIGHT" NOWRAP>Фамилия</TH>
        <TD WIDTH="82%" NOWRAP>
            <INPUT TYPE="Input" NAME="userlastname" SIZE="30">
        </TD>
    </TR>
    <TR>
        <TH WIDTH="25%" ALIGN="RIGHT" NOWRAP>Пароль</TH>
        <TD WIDTH="82%" NOWRAP>
            <INPUT TYPE="PASSWORD" NAME="userpassword" SIZE="30">
        </TD>
    </TR>
    <TR>
        <TH WIDTH="25%" ALIGN="RIGHT" NOWRAP>Email</TH>
        <TD WIDTH="82%" NOWRAP>
            <INPUT TYPE="Input" NAME="useremail" SIZE="30">
        </TD>
    </TR>
    <TR>
        <TD WIDTH="100%" COLSPAN="2" ALIGN="CENTER" NOWRAP>
            <INPUT TYPE="SUBMIT" NAME="Submit">
        </TD>
    </TR>
</TABLE>
</CENTER></DIV>
</FORM>
</BODY>
</HTML>
<?
}

function register_user() {
    global $default_dbname;
    $PHP_SELF = $_SERVER['PHP_SELF'];

    $link_id = db_connect($default_dbname);
    $query = "INSERT INTO Customer
VALUES(',$_GET[userfirstname]','$_GET[userlastname]',
    $_GET[userpassword], '$_GET[useremail]')";
    $result = mysql_query($query);
    if(!$result) {
        Echo "Пожалуйста, перейдите на главную страницу и зарегистрируйтесь.";
    }
    ?>
    <FORM method="GET" action="<?php echo $PHP_SELF ?>">
        <INPUT type="submit" value="На главную">
    </FORM>
    <?php
        exit;
    }
    return $result;
}

function get_userid(){
    global $default_dbname;
    $link_id = db_connect($default_dbname);

    $query = "SELECT User_ID from Customer WHERE Password = '$_GET[userpassword]'";

    $result = mysql_query($query);
    $result_array = mysql_fetch_row($result);
    $userid = $result_array[0];
    if(!$userid) $userid = "";
    return $userid;
}

```



```

function order_menu(){
    $PHP_SELF = $_SERVER['PHP_SELF'];

    ?>
    <HTML>
        <HEAD><TITLE>Рестораны</TITLE></HEAD>
        <BODY>
            <DIV ALIGN="CENTER"><CENTER>
                <H2>Добро пожаловать в "Nandos", <?php echo $_GET['userfirstname']; ?>!</H2>
            </CENTER></DIV>
            <FORM method="GET" action="<?php echo $PHP_SELF ?>">
                <INPUT TYPE="HIDDEN" NAME="action" VALUE="order">
Крылышки:                $1.50
                <INPUT name="Choice1" type="checkbox" value="1.50">
                <BR>
Гамбургер:                $3.00
                <INPUT name="Choice2" type="checkbox" value="3.00">
                <BR>
Безалкогольный напиток: $0.75
                <INPUT name="Choice3" type="checkbox" value="0.75">
                <BR>
                <BR>
                <INPUT type="submit" value="Заказать">
            </FORM>
        </BODY>
    </HTML>

    <?php
    }

    function place_order(){
    global $default_dbname, $max_menu_items;
    $PHP_SELF = $_SERVER['PHP_SELF'];
    $link_id = db_connect($default_dbname);
    $total = 0;
    for ($counter = 1; $counter <= 10; $counter ++){
        {
            $a = "Choice" . "$counter";
            if (isset($_GET[$a])){
                $total += $_GET[$a];
            }
        }
    }

    $query = "INSERT INTO Orders VALUES ('", '$_SESSION[userid]', 'Nandos', $total)";

    $result = mysql_query($query);
    if(!$result) die ("Заказ не принят. Попробуйте еще раз.");
    else {
        //В реальной системе здесь должен быть выполнен переход на
        // страницу подтверждения, простое обновление страницы
        // приводит к дублированию заказа.
        echo "Ваш заказ принят и будет выполнен в течение часа. Общая стоимость
заказа: \$$total";
    ?>
        <FORM method="GET" action="<?php echo $PHP_SELF ?>">
            <INPUT type="submit" value="На главную">
        </FORM>
    <?php
    }
    }

    function customer_session(){
        $_SESSION['useremail'] = $_GET['useremail'];

```

```

    register_user();
    $userid = get_userid();
    $_SESSION['userid'] = $userid;
}

session_start();
session_register("userid", "useremail");
if (empty($_GET['action'])) {
    $_GET['action'] = "";
}

switch($_GET['action']) {
    case "order":
        place_order();
        break;
    case "register":
        customer_session();
        order_menu();
        break;
    default:
        login_form();
        break;
}

?>

```

Глава 12

Упражнение 1

В чем разница между классом и объектом?

Решение

Класс подобен чертежу, в котором лишь указаны инструкции для создания объекта. Объект представляет собой экземпляр класса. Классы используются во время разработки, в ходе изменения кода в PHP-файле. Объекты используются во время выполнения программы, когда присваиваются или изменяются значения свойств и вызываются методы. Класс представлен текстом в текстовом файле. Объект существует в памяти в виде инструкций.

Упражнение 2

Объясните идею наследования и дайте пример того, когда его следует использовать, не повторяя при этом примеров, рассмотренных в данной главе.

Решение

Наследование — способность объекта приобретать свойства и методы родительского объекта. Наследование подразумевает связанную иерархию и повторное использование кода. Дочерний класс — более специализированная версия родительского класса, имеющая дополнительные методы и свойства и/или другую реализацию тех же методов и свойств. В дочернем классе не может быть меньше методов или свойств, чем в родительском. Примером наследования может быть связь между позвоночными животными и млекопитающими. Все позвоночные животные имеют позвоночник, состоящий из множества позвоночных костей, окружающих спинной мозг,

и большой головной мозг, заключенный в черепную коробку. Млекопитающие представляют собой более специализированный вид позвоночных животных. Хотя млекопитающие имеют все свойства позвоночных животных, они также имеют волосяной покров или мех, они теплокровны и вскармливают потомство молоком. Класс Позвоночные может иметь такие свойства, как количествоПозвонков и типСкелета (значением последнего свойства для акул будет хрящевой, а для млекопитающих и рептилий — костяной). Млекопитающие наследуют все эти свойства и имеют свои собственные — цветШерсти и нормальнаяТемператураТела.

Упражнение 3

Опишите полезные особенности интерфейса и его практическое применение в программной архитектуре. Чем интерфейс отличается от наследуемого класса? Приведите дополнительные примеры возможного применения интерфейсов.

Решение

Интерфейс представляет собой абстрактную конструкцию, определяющую “контракт” — жесткое требование обеспечить в реализующих классах определенные методы. Хотя наследование классов предполагает наличие непосредственной связи между родительским классом и наследующими его подклассами, интерфейс не означает такой связи, а используется для определения классов, способных на выполнение тех же функций. Так как все классы, реализующие интерфейс, должны по определению иметь, как минимум, набор методов, определенных в интерфейсе (они могут иметь дополнительные методы, но не меньше), интерфейс можно использовать для передачи не связанных иначе объектов одним и тем же функциям. Например, PHP поставляется с двумя интерфейсами — `Iterator` и `IteratorAggregate`, — которые позволяют передавать объекты в конструкции `foreach` тогда и только тогда, когда эти объекты реализуют один из этих двух интерфейсов. Эти интерфейсы позволяют обрабатывать в `foreach`-циклах объекты, содержащие в себе другие объекты. Любой класс может реализовывать эти интерфейсы, поэтому можно использовать абсолютно не связанные друг с другом объекты в одних и тех же `foreach`-конструкциях.

Глава 15

Упражнение

Представленное в главе приложение хорошо работает в случае создания и отправки одиночных писем. Модифицируйте это приложение так, чтобы оно позволяло отправлять сообщения списку получателей, взятому из базы данных.

Решение

Первый этап создания приложения, которое рассылает e-mail-сообщения по списку адресов, состоит в формировании списка этих адресов в виде массива. Как правило, процесс получения данных из базы (такой как MySQL или PostgreSQL) начинается с выполнения некоторого SQL-оператора `SELECT` с помощью функции `pg_exec()` или `mysql_query()`. Отобранные записи сохраняются в PHP-переменной (которую нередко называют `$res`, т.е. “результат”). Впоследствии количество записей в таком массиве можно определить с помощью функции `pg_numrows()` или `mysql_numrows()`, а цикл

for с помощью функции `pg_fetch_array()` или `mysql_fetch_array()` обеспечивает способ получения отдельных строк (записей) из переменной `$res`.

Таким образом, можно выполнить SELECT-запрос для базы данных, содержащей информацию о клиентах (включая e-mail-адреса), а затем, в случае если в результате имеется несколько записей, использовать for-цикл для выборки каждого e-mail-адреса и передачи его функции, осуществляющей рассылку писем (код такой функции с некоторой реализацией проверки достоверности адресов показан ниже).

```
$query = "SELECT email FROM customers;";
$res = pg_exec($conn,$query);
if(pg_numrows($res) >= 1) {
    for ($i=0; $i<=pg_numrows($res)-1; $i++) {
        $rec = pg_fetch_array($res,$i);
        if ($rec[email] != "") {
            if (strpos($rec[email],"@") >= 0) {
                $body = "Это тело сообщения"\n\n";
                //Заголовок From:
                $headers .= "From: Example <info@example.com>\n";
                //Отправка почты
                $to = "$rec[email]";
                $subject = "Example Email";
                mail($to, $subject, $body, $headers);
            } else {
                echo = "Адрес $i неправильный.<br>";
            }
        }
    }
}
```

Глава 16

Упражнение 1

Создайте PHP-сценарий, который открывает файл изображения и добавляет в изображение черную рамку толщиной в один пиксель.

Решение

Единственная хитрость, необходимая для успешного выполнения упражнения, заключается в том, что левый верхний угол изображения имеет координаты $x = 0$ и $y = 0$. Это означает, что для получения координат самой нижней правой точки из величин ширины и высоты изображения необходимо отнять 1. Ниже приведен сценарий-решение (`exercisel.php`) этого упражнения:

```
<?php
$myImage = imagecreatefromjpeg('cape_point.jpg');
$black = imagecolorallocate($myImage,0,0,0);
$width = imagesx($myImage);
$height = imagesy($myImage);
imagerectangle($myImage, 0, 0, $width-1, $height-1, $black);
header("Content-type: image/jpeg");
imagejpeg($myImage);
imagedestroy($myImage);
?>
```

Этот сценарий (или подобный ему) должен формировать красивую рамку вокруг изображения, как показано на рис. А.1.

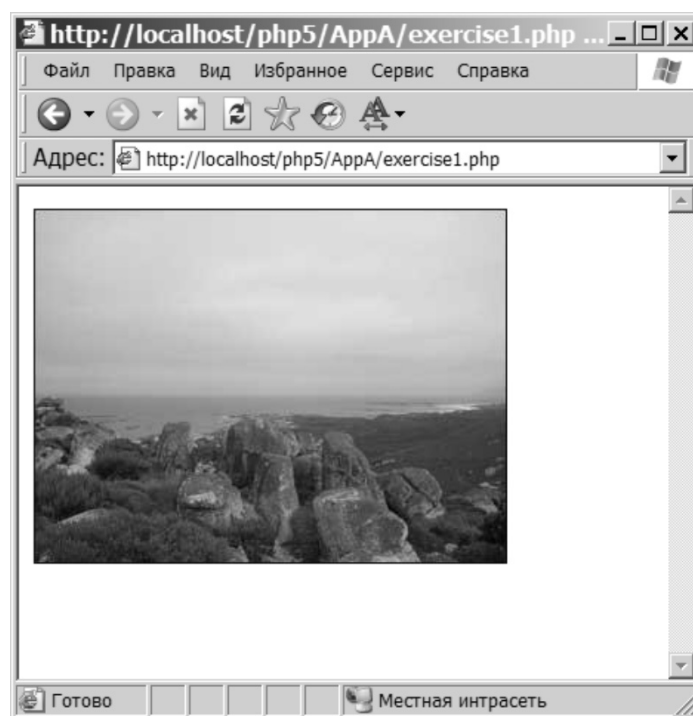


Рис. А.1.

Упражнение 2

Используя функции `disk_total_space()` и `disk_free_space()`, покажите графическим способом объем используемого пространства на жестком диске.

Решение

Существует два очевидных способа графического отображения результатов этого упражнения. Самый простой и быстрый способ — создать скользящую шкалу: прямоугольник, состоящий из двух цветных блоков, один из которых представляет используемое пространство на диске, а второй — свободное пространство. Это позволит создать компактную диаграмму использования дискового пространства. Ниже подробно разбирается код (файл `exercise2a.php`) для решения этой задачи.

Хорошей методикой всегда считалось хранение значений ширины и высоты изображений в сценарии. Изменить впоследствии размер изображения можно будет в начале сценария, в результате чего все вычисления ширины и высоты будут автоматически обновлены.

```
<?php
$iWidth = 500;
$iHeight = 50;
```

Создайте изображение и добавьте в него белый и черный цвет. Так как белый в данном случае добавляется первым, он будет фоновым цветом изображения. Черный будет использоваться для создания рамки изображения.

```
$myImage = imagecreate($iWidth, $iHeight);
$white = imagecolorallocate($myImage, 255, 255, 255);
$black = imagecolorallocate($myImage, 0, 0, 0);
```

В приведенном здесь решении красный цвет представляет занятое пространство на диске, а зеленый — свободное пространство, хотя можно использовать любые другие цвета.

```
$red = imagecolorallocate($myImage, 255, 0, 0);
$green = imagecolorallocate($myImage, 0, 255, 0);
```

Необходимо получить общий объем дискового пространства и объем свободного пространства. Оба значения возвращаются в байтах. Фактические значения несущественны — они требуются лишь для вычисления пропорции, которая будет воспроизведена в изображении.

```
$diskTotal = disk_total_space('/');
$diskFree = disk_free_space('/');
```

Затем вокруг изображения необходимо нарисовать черную рамку толщиной в один пиксель.

```
imagerectangle($myImage, 0, 0, $iWidth - 1, $iHeight - 1, $black);
```

Переменная `$threshold` используется для отметки x-координаты той точки диаграммы, где происходит переход от занятого пространства к свободному. Так как на подобных диаграммах занятое пространство обычно представлено слева, сначала необходимо вычислить площадь левой (красной) области прямоугольника — она пропорциональна общему объему дискового пространства минус объем свободного пространства. Полученное значение следует разделить на общий объем дискового пространства, чтобы получить число в диапазоне от 0 до 1, которое затем нужно умножить на ширину изображения (перед этим необходимо вычесть 2 из ширины изображения, поскольку вокруг изображения уже есть рамка). После этого нужно прибавить 1, так как отсчет пикселей начинается с 0.

```
$threshold = intval((( $diskTotal - $diskFree ) / $diskTotal) * ($iWidth-2)) + 1;
```

Например, предположим, что объем диска равен 400 байтам, из которых 200 занято, тогда $200/400 = 0,5$. Если ширина изображения равна 500 пикселей, то умножаем 498 (ширина минус 2 пикселя для рамки) на 0,5 и получаем 249. Затем прибавляем 1 и получаем 250 — позицию перехода одной области изображения в другую.

Чтобы представить занятое пространство, создаем в изображении красный прямоугольник, достигающий точки перехода:

```
imagefilledrectangle($myImage, 1, 1, $threshold, ($iHeight-2), $red);
```

Затем создаем зеленый прямоугольник, начиная от точки перехода — для формирования области свободного пространства:

```
imagefilledrectangle($myImage, ($threshold + 1), 1, ($iWidth - 2),
    $iHeight-2, $green);
```

Завершаем сценарий как обычно:

```
header("Content-type: image/png");
imagepng($myImage);
imagedestroy($myImage);
?>
```

Результат выполнения сценария показан на рис. А.2.

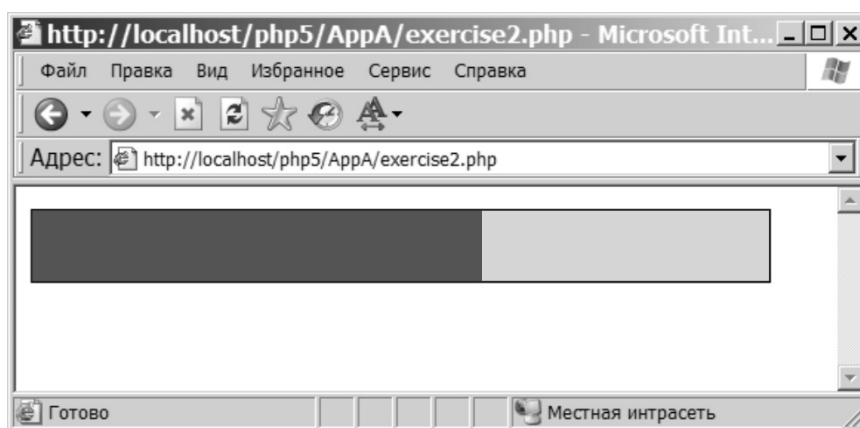


Рис. А.2.

Можно выполнить это упражнение иначе — построить круговую диаграмму использования дискового пространства. Файл `exercise2b.php` содержит код соответствующего сценария. Рассмотрим его подробнее.

Вместо определения точки перехода вычисляется угол сектора. Методика вычислений остается прежней, кроме того, что относительное значение дискового объема умножается не на ширину изображения, а на 360 — количество градусов в полном круге.

```
<?php
$iWidth = 200;
$iHeight = 200;
$myImage = imagecreate($iWidth, $iHeight);
$white = imagecolorallocate($myImage, 255, 255, 255);
$red = imagecolorallocate($myImage, 255, 0, 0);
$green = imagecolorallocate($myImage, 0, 255, 0);
$diskTotal = disk_total_space('/');
$diskFree = disk_free_space('/');
$usedDegrees = intval((($diskTotal - $diskFree) / $diskTotal) * 360);
```

Функция `imagefilledarc()` работает так же, как и `imagearc()`, кроме того, что она принимает дополнительный аргумент, определяющий способ заполнения дуги. Использование в качестве значения этого параметра PHP-константы `IMG_ARC_EDGED` приводит к тому, что две конечные точки дуги соединяются с центром и полученный контур заполняется заданным цветом. Отсчет начинается с 0 градусов, дуга рисуется до вычисленного угла, `$usedDegrees`.

```
imagefilledarc($myImage, ($iWidth/2), ($iHeight/2), $iWidth - 2, $iHeight - 2,
0, $usedDegrees, $red, IMG_ARC_EDGED);
```

Чтобы нарисовать дугу, представляющую свободное пространство, начните с того места, где заканчивается область занятого пространства, т.е. с `$usedDegree`, и продолжайте рисовать дугу до 360 градусов, до конца круга:

```
imagefilledarc($myImage, ($iWidth/2), ($iHeight/2), $iWidth - 2, $iHeight - 2,
$usedDegrees, 360, $green, IMG_ARC_EDGED);
```

Завершаем сценарий как обычно:

```
header("Content-type: image/png");
imagepng($myImage);
imagedestroy($myImage);
?>
```

Результат выполнения сценария показан на рис. А.3.

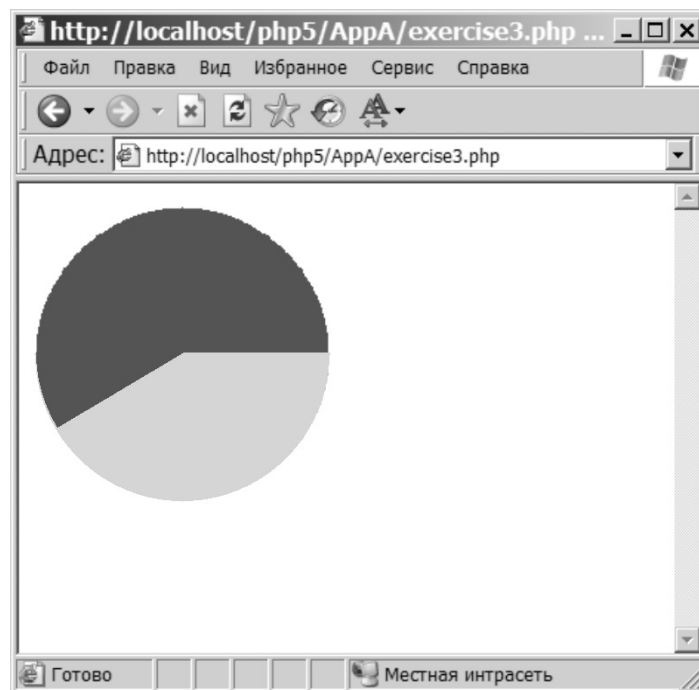


Рис. А.3.



Справочник по РНР-функциям

Это приложение не может содержать исчерпывающего перечня всех РНР-функций (иначе оно заняло бы сотни страниц) — здесь представлено подмножество функций (и их описаний), с которыми, по мнению авторов, разработчики РНР-сценариев сталкиваются в повседневной работе. В перечень включены основные функции языка, а также функции часто используемых РНР-расширений.

Приложение задумывалось как краткий справочник или памятка по входным параметрам и возвращаемым типам данных, а не как полное описание функций и методики их использования в сценариях. Чтобы узнать, как использовать ту или иную функцию или найти информацию по неописанным здесь функциям, обратитесь к странице РНР-документации www.php.net/docs.php.

Web-сервер Apache

Функция	Тип возвращаемых данных	Описание
<code>apache_child_terminate(void)</code>	Bool	Завершает apache-процесс после данного запроса
<code>apache_note(string note_name [, string note_value])</code>	String	Получает и устанавливает замечания apache-запроса
<code>virtual(string filename)</code>	Bool	Выполняет подзапрос apache
<code>getallheaders(void)</code>	Array	Псевдоним для функции <code>apache_request_headers()</code>
<code>apache_request_headers(void)</code>	Array	Возвращает все заголовки http-запроса

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>apache_response_headers(void)</code>	Array	Возвращает все заголовки http-ответа
<code>apache_setenv(string variable, string value [, bool walk_to_top])</code>	Bool	Устанавливает apache-переменную <code>subprocess_env</code>
<code>apache_lookup_uri(string URI)</code>	Object	Выполняет частичный запрос по заданному URI и возвращает всю информацию о нем
<code>apache_get_version(void)</code>	String	Возвращает версию Apache
<code>apache_get_modules(void)</code>	Array	Возвращает список загруженных модулей apache

Массивы

Функция	Тип возвращаемых данных	Описание
<code>krsort(array array_arg [, int sort_flags])</code>	Bool	Сортирует массив по ключам в обратном порядке
<code>ksort(array array_arg [, int sort_flags])</code>	Bool	Сортирует массив по ключам
<code>count(mixed var [, int mode])</code>	Int	Считает элементы в переменной (обычно в массиве)
<code>natsort(array array_arg)</code>	Void	Сортирует массив с использованием алгоритма "естественный порядок"
<code>natcasesort(array array_arg)</code>	Void	Сортирует массив с использованием нечувствительного к регистру символов алгоритма "естественный порядок"
<code>asort(array array_arg [, int sort_flags])</code>	Bool	Сортирует массив и сохраняет связь индексов и значений
<code>arsort(array array_arg [, int sort_flags])</code>	Bool	Сортирует массив в обратном порядке и сохраняет связь индексов и значений
<code>sort(array array_arg [, int sort_flags])</code>	Bool	Сортирует массив
<code>rsort(array array_arg [, int sort_flags])</code>	Bool	Сортирует массив в обратном порядке
<code>usort(array array_arg, string cmp_function)</code>	Bool	Сортирует массив по значениям с помощью пользовательской функции сравнения
<code>uasort(array array_arg, string cmp_function)</code>	Bool	Сортирует массив по значениям с помощью пользовательской функции сравнения и сохраняет связь индексов и значений

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>uksort(array array_arg, string cmp_function)</code>	Bool	Сортирует массив с помощью пользовательской функции сравнения
<code>end(array array_arg)</code>	Mixed	Устанавливает внутренний указатель массива на последний элемент и возвращает значение этого элемента
<code>prev(array array_arg)</code>	Mixed	Перемещает внутренний указатель массива к предыдущему элементу и возвращает этот элемент
<code>next(array array_arg)</code>	Mixed	Перемещает внутренний указатель массива к следующему элементу и возвращает этот элемент
<code>reset(array array_arg)</code>	Mixed	Устанавливает внутренний указатель массива на первый элемент и возвращает этот элемент
<code>current(array array_arg)</code>	Mixed	Возвращает текущий элемент массива
<code>key(array array_arg)</code>	Mixed	Возвращает ключ текущего элемента массива
<code>min(mixed arg1 [, mixed arg2 [, mixed. . .]])</code>	Mixed	Находит наименьшее значение в массиве или последовательности аргументов
<code>max(mixed arg1 [, mixed arg2 [, mixed. . .]])</code>	Mixed	Находит наибольшее значение в массиве или последовательности аргументов
<code>array_walk(array input, string funcname [, mixed userdata])</code>	Bool	Применяет пользовательскую функцию к каждому члену массива
<code>array_walk_recursive(array input, string funcname [, mixed userdata])</code>	Bool	Рекурсивно применяет пользовательскую функцию к каждому члену массива
<code>in_array(mixed needle, array haystack [, bool strict])</code>	Bool	Проверяет существование заданного значения в массиве
<code>array_search(mixed needle, array haystack [, bool strict])</code>	Mixed	Ищет в массиве заданное значение и возвращает соответствующий ключ в случае успеха
<code>extract(array var_array [, int extract_type [, string prefix]])</code>	Int	Импортирует переменные в таблицу имен из массива
<code>compact(mixed var_names [, mixed. . .])</code>	Array	Создает массив, содержащий переменные и их значения
<code>array_fill(int start_key, int num, mixed val)</code>	Array	Создает массив с num элементов, начи- ная с индекса start_key и присваивая каждому элементу значение val
<code>range(mixed low, mixed high[, int step])</code>	Array	Создает массив, содержащий диапазон целых чисел или символов от low до high (включительно)

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>shuffle(array array_arg)</code>	Bool	Случайным образом перемешивает содержимое массива
<code>array_push(array stack, mixed var [, mixed. . .])</code>	Int	Вставляет один или более элементов в конец массива
<code>array_pop(array stack)</code>	Mixed	Вытаскивает последний элемент массива
<code>array_shift(array stack)</code>	Mixed	Удаляет элемент в начале массива
<code>array_unshift(array stack, mixed var [, mixed. . .])</code>	Int	Вставляет один или более элементов в начало массива
<code>array_splice(array input, int offset [, int length [, array replacement]])</code>	Array	Удаляет последовательность элементов массива, заданную смещением и длиной, и замещает ее другой последовательностью
<code>array_slice(array input, int offset [, int length])</code>	Array	Возвращает последовательность элементов массива, заданную смещением и длиной
<code>array_merge(array arr1, array arr2 [, array. . .])</code>	Array	Объединяет элементы заданных массивов в один массив
<code>array_merge_recursive(array arr1, array arr2 [, array. . .])</code>	Array	Рекурсивно объединяет элементы двух или более заданных массивов в один массив
<code>array_keys(array input [, mixed search_value])</code>	Array	Возвращает все ключи массива; если задан необязательный параметр <code>search_value</code> , то возвращаются ключи только тех элементов, которые содержат заданное значение
<code>array_values(array input)</code>	Array	Возвращает все значения массива
<code>array_count_values(array input)</code>	Array	Возвращает массив, ключами которого являются значения массива <code>input</code> , а значениями - частота повторения этих значений
<code>array_reverse(array input [, bool preserve keys])</code>	Array	Возвращает новый массив, порядок элементов в котором обратный исходному
<code>array_pad(array input, int pad_size, mixed pad_value)</code>	Array	Возвращает копию входного массива, дополненную значениями <code>pad_value</code> до величины <code>pad_size</code>
<code>array_flip(array input)</code>	Array	Меняет местами ключи и значения массива
<code>array_change_key_case(array input [, int case=CASE_LOWER])</code>	Array	Возвращает массив со всеми строковыми ключами в верхнем (или в нижнем) регистре
<code>array_unique(array input)</code>	Array	Удаляет из массива дублирующиеся значения

Функция	Тип возвращаемых данных	Описание
<code>array_intersect(array arr1, array arr2 [, array. . .])</code>	Array	Возвращает массив, содержащий значения массива <code>arr1</code> , присутствующие во всех остальных аргументах
<code>array_uintersect(array arr1, array arr2 [, array. . .], callback data_compare_func)</code>	Array	Возвращает массив, содержащий значения массива <code>arr1</code> , присутствующие во всех остальных аргументах; данные сравниваются с помощью пользовательской функции обратного вызова
<code>array_intersect_assoc(array arr1, array arr2 [, array. . .])</code>	Array	Возвращает массив, содержащий значения массива <code>arr1</code> , присутствующие во всех остальных аргументах; ключи используются для более избирательной проверки
<code>array_uintersect_assoc(array arr1, array arr2 [, array. . . .], callback data_compare_func)</code>	Array	Возвращает массив, содержащий значения массива <code>arr1</code> , присутствующие во всех остальных аргументах; ключи используются для более избирательной проверки; данные сравниваются с помощью пользовательской функции обратного вызова
<code>array_intersect_uassoc(array arr1, array arr2 [, array. . .], callback key_compare_func)</code>	Array	Возвращает массив, содержащий значения массива <code>arr1</code> , присутствующие во всех остальных аргументах; ключи используются для более избирательной проверки и сравниваются с помощью пользовательской функции обратного вызова
<code>array_uintersect_uassoc(array arr1, array arr2 [, array. . .], callback data_compare_func, callback key_compare_func)</code>	Array	Возвращает массив, содержащий значения массива <code>arr1</code> , присутствующие во всех остальных аргументах; ключи используются для более избирательной проверки; и ключи и данные сравниваются с помощью пользовательской функции обратного вызова
<code>array_diff(array arr1, array arr2 [, array. . .])</code>	Array	Возвращает массив, состоящий из значений массива <code>arr1</code> , которые отсутствуют во всех последующих аргументах
<code>array_udiff(array arr1, array arr2 [, array. . .], callback data_comp_func)</code>	Array	Возвращает массив, состоящий из значений массива <code>arr1</code> , которые отсутствуют во всех последующих аргументах; элементы сравниваются с помощью пользовательской функции
<code>array_diff_assoc(array arr1, array arr2 [, array. . .])</code>	Array	Возвращает массив, состоящий из элементов массива <code>arr1</code> , пары ключ-значение которых отсутствуют во всех последующих аргументах

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>array_diff_uassoc(array arr1, array arr2 [, array. . .], callback data_comp_func)</code>	Array	Возвращает массив, состоящий из элементов массива <code>arr1</code> , пары ключ-значение которых отсутствуют во всех последующих аргументах; элементы сравниваются с помощью пользовательской функции
<code>array_udiff_assoc(array arr1, array arr2 [, array. . .], callback key_comp_func)</code>	Array	Возвращает массив, состоящий из элементов массива <code>arr1</code> , пары ключ-значение которых отсутствуют во всех последующих аргументах; ключи сравниваются с помощью пользовательской функции
<code>array_udiff_uassoc(array arr1, array arr2 [, array. . .], callback data_comp_func, callback key_comp_func)</code>	Array	Возвращает массив, состоящий из элементов массива <code>arr1</code> , пары ключ-значение которых отсутствуют во всех последующих аргументах; и ключи и элементы сравниваются с помощью пользовательской функции
<code>array_multisort(array arr1 [, SORT_ASC SORT_DESC [, SORT_REGULAR SORT_NUMERIC SORT_STRING]] [, array arr2 [, SORT_ASC SORT_DESC [, SORT_REGULAR SORT_NUMERIC SORT_STRING]], . . .])</code>	Bool	Одновременно сортирует несколько массивов (аналогична работе sql-предложения <code>order by</code>)
<code>array_rand(array input [, int num_req])</code>	Mixed	Возвращает один или несколько случайных ключей или значений из массива
<code>array_sum(array input)</code>	Mixed	Высчитывает сумму значений массива
<code>array_reduce(array input, mixed callback [, int initial])</code>	Mixed	Итеративно уменьшает массив до единственного значения, используя функцию обратного вызова
<code>array_filter(array input [, mixed callback])</code>	Array	Фильтрует элементы массива с помощью функции обратного вызова
<code>array_map(mixed callback, array input1 [, array input2, . . .])</code>	Array	Применяется обратный вызов к элементам заданного массива (или массивов)
<code>array_key_exists(mixed key, array search)</code>	Bool	Проверяет, существует ли в массиве данный индекс или ключ
<code>array_chunk(array input, int size [, bool preserve_keys])</code>	Array	Делит массив на отрезки
<code>array_combine(array keys, array values)</code>	Array	Создает новый массив, используя элементы одного массива в качестве ключей, а элементы другого в качестве соответствующих значений

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>each(array arr)</code>	Array	Возвращает текущую пару ключ/значение из массива и перемещает указатель массива к следующему элементу
<code>error_reporting(int new_error_level=null)</code>	Int	Возвращает текущий уровень отображения ошибок и устанавливает новый уровень, если он задан

BCMath

Функция	Тип возвращаемых данных	Описание
<code>bcadd(string left_operand, string right_operand [, int scale])</code>	String	Складывает два числа произвольной точности
<code>bcsb(string left_operand, string right_operand [, int scale])</code>	String	Вычитает одно число произвольной точности из другого
<code>bcmul(string left_operand, string right_operand [, int scale])</code>	String	Перемножает два числа произвольной точности
<code>bcdiv(string left_operand, string right_operand [, int scale])</code>	String	Делит два числа произвольной точности
<code>bcmod(string left_operand, string right_operand)</code>	String	Возвращает остаток от целочисленного деления операндов произвольной точности
<code>bcpowmod(string x, string y, string mod [, int scale])</code>	String	Возводит число произвольной точности в степень (обозначенную другим числом)
<code>bcpow(string x, string y [, int scale])</code>	String	Возводит число произвольной точности в степень (обозначенную другим числом)
<code>bcsqrt(string operand [, int scale])</code>	String	Извлекает квадратный корень из числа произвольной точности
<code>bccomp(string left_operand, string right_operand [, int scale])</code>	Int	Сравнивает два числа произвольной точности
<code>bcscale(int scale)</code>	Bool	Устанавливает точность вычислений по умолчанию для всех bcmath-функций

BZip2

Функция	Тип возвращаемых данных	Описание
<code>bzopen(string \$int file fp, string mode)</code>	resource	Открывает новый bzip2-поток
<code>bzread(int bz[, int length]</code>	String	Считывает из bzip2-потока length байт или 1024 байт, если параметр length не задан
<code>bzwrite(int bz, string data [, int length]</code>	Int	Записывает содержимое строки в bzip2-поток
<code>bzerrno(resource bz)</code>	Int	Возвращает номер ошибки bzip2
<code>bzerrstr(resource bz)</code>	String	Возвращает строку ошибки bzip2
<code>bzerror(resource bz)</code>	Array	Возвращает в ассоциативном массиве номер ошибки bzip2 и строку ошибки
<code>bzcompress(string source [, int blocksize100k [, int workfactor]])</code>	String	Сжимает строку в bzip2-кодированные данные
<code>bzdecompress(string source [, int small])</code>	String	Выполняет декомпрессию bzip2-кодированных данных
<code>bzclose((resource bz)</code>	Int	Закрывает bzip2-указатель на файл
<code>bzflush(resource bz)</code>	Int	Форсирует запись всех буферизованных bzip2-данных

Календарь

Функция	Тип возвращаемых данных	Описание
<code>unixtojd([int timestamp])</code>	Int	Конвертирует временную метку unix в дату по юлианскому календарю
<code>jdtounix(int jday)</code>	Int	Конвертирует юлианскую дату в временную метку unix
<code>cal_info(int calendar)</code>	Array	Возвращает информацию о заданном календаре
<code>cal_days_in_month(int calendar, int month, int year)</code>	Int	Возвращает количество дней в месяце для заданного года и календаря
<code>cal_to_jd(int calendar, int month, int day, int year)</code>	Int	Конвертирует дату из поддерживаемого календаря в юлианский
<code>cal_from_jd(int jd, int calendar)</code>	Array	Конвертирует из юлианского календаря в поддерживаемый календарь и возвращает расширенную информацию
<code>jdtogregorian(int juliandaycount)</code>	String	Конвертирует дату из юлианского календаря в григорианский

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>gregoriantojd(int month, int day, int year)</code>	Int	Конвертирует григорианскую дату в юлианскую
<code>jdtojulian(int juliandaycount)</code>	String	Конвертирует количество дней в юлианском календаре в дату юлианского календаря
<code>juliantojd(int month, int day, int year)</code>	Int	Конвертирует дату по юлианскому календарю в количество юлианских дней
<code>jdtojewish(int juliandaycount [, bool hebrew [, int fl]])</code>	String	Конвертирует юлианское количество дней в иудейскую дату
<code>jewishtojd(int month, int day, int year)</code>	Int	Конвертирует дату иудейского календаря в количество дней по юлианскому календарю
<code>jdtofrench(int juliandaycount)</code>	String	Конвертирует юлианское летоисчисление в дату по французскому республиканскому календарю
<code>frenchtojd(int month, int day, int year)</code>	Int	Конвертирует данные из французского республиканского календаря в юлианское летоисчисление
<code>jddayofweek(int juliandaycount [, int mode])</code>	Mixed	Возвращает день недели или его порядковый номер по юлианскому летоисчислению
<code>jdmonthname(int juliandaycount, int mode)</code>	String	Возвращает название месяца юлианского календаря
<code>easter_date([int year])</code>	Int	Возвращает временную метку полуночи пасхи для заданного года (по умолчанию для текущего)
<code>easter_days([int year, [int method]])</code>	Int	Возвращает дату — количество дней от 21 марта до пасхи в заданном году (по умолчанию в текущем)

Классы и объекты

Функция	Тип возвращаемых данных	Описание
<code>class_exists (string classname)</code>	Bool	Проверяет существование класса
<code>get_class(object object)</code>	String	Возвращает имя класса объекта
<code>get_parent_class (mixed object)</code>	String	Возвращает имя родительского класса для объекта или класса
<code>is_subclass_of(object object, string class_name)</code>	Bool	Возвращает true, если заданный класс является одним из предков объекта

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>is_a(object object, string class_name)</code>	Bool	Возвращает true, если это объект данного класса или этот класс является одним из предков объекта
<code>get_class_vars(string class_name)</code>	Array	Возвращает массив свойств класса и их значения по умолчанию
<code>get_object_vars(object obj)</code>	Array	Возвращает массив свойств объекта
<code>get_class_methods(mixed class)</code>	Array	Возвращает массив имен методов класса или экземпляра этого класса
<code>method_exists(object object, string method)</code>	Bool	Проверяет существование метода класса
<code>Get_class_vars(string class_name)</code>	Array	Возвращает массив установленных по умолчанию свойств класса
<code>get_declared_classes(string class_name)</code>	Array	Возвращает массив с именами объявленных в текущем сценарии классов

Типы символов

Функция	Тип возвращаемых данных	Описание
<code>ctype_alnum(string text)</code>	Bool	Проверяет наличие алфавитно-цифрового символа
<code>ctype_alpha(string text)</code>	Bool	Проверяет наличие алфавитного символа
<code>ctype_cntrl(string text)</code>	Bool	Проверяет наличие управляющего символа
<code>ctype_digit(string text)</code>	Bool	Проверяет наличие цифр
<code>ctype_graph(string text)</code>	Bool	Проверяет наличие печатаемых символов, кроме пробела
<code>ctype_lower(string text)</code>	Bool	Проверяет наличие символа в нижнем регистре
<code>ctype_print(string text)</code>	Bool	Проверяет наличие печатаемых символов
<code>ctype_punct(string text)</code>	Bool	Проверяет наличие печатаемых символов, которые не являются пробельными или алфавитно-цифровыми символами
<code>ctype_space(string text)</code>	Bool	Проверяет наличие пробельных символов
<code>ctype_upper(string text)</code>	Bool	Проверяет наличие символов верхнего регистра
<code>ctype_xdigit(string text)</code>	Bool	Проверяет наличие символов, представляющих шестнадцатеричные цифры

Curl

<i>Функция</i>	<i>Тип возвращаемых данных</i>	<i>Описание</i>
<code>curl_version([int version])</code>	Array	Возвращает версию текущей CURL
<code>curl_init([string url])</code>	resource	Инициализирует CURL-сеанс
<code>curl_setopt(resource ch, string option, mixed value)</code>	Bool	Устанавливает параметры CURL-операции
<code>curl_exec(resource ch)</code>	Bool	Выполняет CURL-сеанс
<code>curl_getinfo(resource ch, int opt)</code>	Mixed	Возвращает информацию об определенной операции
<code>curl_error(resource ch)</code>	String	Возвращает строку с описанием последней ошибки для текущего сеанса
<code>curl_errno(resource ch)</code>	Int	Возвращает целочисленный номер последней ошибки
<code>curl_close(resource ch)</code>	Void	Закрывает CURL-сеанс
<code>curl_multi_init(void)</code>	resource	Возвращает набор CURL-дескрипторов
<code>curl_multi_add_handle (resource multi, resource ch)</code>	Int	Добавляет обычный CURL-дескриптор в набор CURL-дескрипторов
<code>curl_multi_remove_handle (resource mh, resource ch)</code>	Int	Удаляет CURL-дескриптор из набора curl-дескрипторов
<code>curl_multi_select(resource mh[, double timeout])</code>	Int	Возвращает все сокеты, связанные с расширением CURL, которые затем можно выбрать
<code>curl_multi_exec(resource mh)</code>	Int	Выполняет операции текущего CURL-дескриптора
<code>curl_multi_getcontent (resource ch)</code>	String	Возвращает результат операции, если был установлен параметр CURLOPT_RETURNTRANSFER
<code>curl_multi_info_read (resource mh)</code>	Array	Возвращает информацию о текущих операциях
<code>curl_multi_close (resource mh)</code>	Void	Закрывает набор CURL-дескрипторов

Дата и время

<i>Функция</i>	<i>Тип возвращаемых данных</i>	<i>Описание</i>
<code>time(void)</code>	Int	Возвращает текущую временную метку UNIX
<code>mktime(int hour, int min, int sec, int mon, int day, int year)</code>	Int	Возвращает временную метку UNIX для даты

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>gmkmtime(int hour, int min, int sec, int mon, int day, int year)</code>	Int	Возвращает временную метку UNIX для GMT-даты
<code>date(string format [, int timestamp])</code>	String	Форматирует локальное время/даты
<code>gmdate(string format [, int timestamp])</code>	String	Форматирует GMT/CUT дату/время
<code>idate(string format [, int timestamp])</code>	Int	Форматирует локальное время/дату в виде целого числа
<code>localtime([int timestamp [, bool associative_array]])</code>	Array	Возвращает результаты вызова соответствующей C-функции в виде ассоциативного массива, если аргумент <code>associative_array</code> равен 1 (иначе возвращается обычный массив)
<code>getdate([int timestamp])</code>	Array	Возвращает информацию о дате/времени
<code>checkdate(int month, int day, int year)</code>	Bool	Возвращает true (1), если переданная дата является достоверной датой григорианского календаря
<code>strftime(string format [, int timestamp])</code>	String	Форматирует локальное время/дату в соответствии с локальными настройками
<code>gmstrftime(string format [, int timestamp])</code>	String	Форматирует локальное GMT/UTC-время/дату в соответствии с локальными настройками
<code>strtotime(string time, int now)</code>	Int	Конвертирует строковое представление даты и времени во временную метку
<code>microtime(void)</code>	String	Возвращает текущее время с секундами и микросекундами
<code>gettimeofday(void)</code>	Array	Возвращает текущее время в виде массива
<code>getrusage([int who])</code>	Array	Возвращает массив статистических данных об использовании системы
<code>date_sunrise(mixed time [, int format [, float latitude [, float longitude [, float zenith [, float gmt_offset]]]])</code>	Mixed	Возвращает время восхода солнца для заданной даты и места
<code>date_sunset(mixed time [, int format [, float latitude [, float longitude [, float zenith [, float gmt_offset]]]])</code>	Mixed	Возвращает время заката солнца для заданной даты и места

Каталоги

Функция	Тип возвращаемых данных	Описание
<code>opendir(string path)</code>	Mixed	Открывает каталог и возвращает его дескриптор <code>dir_handle</code>
<code>dir(string directory)</code>	Object	Класс <code>directory</code> , имеющий свойства <code>handle</code> и <code>path</code> , а также методы <code>read</code> , <code>rewind</code> и <code>close</code>
<code>closedir([resource dir_handle])</code>	Void	Закрывает дескриптор каталога, заданный параметром <code>dir_handle</code>
<code>chroot(string directory)</code>	Bool	Изменяет корневой каталог
<code>chdir(string directory)</code>	Bool	Смена текущего каталога
<code>getcwd(void)</code>	Mixed	Возвращает путь к текущему рабочему каталогу
<code>rewinddir([resource dir_handle])</code>	Void	Возвращает в начало дескриптор каталога
<code>readdir([resource dir_handle])</code>	String	Читает запись из дескриптора каталога <code>dir_handle</code>
<code>glob(string pattern [, int flags])</code>	Array	Находит пути, совпадающие с образцом <code>pattern</code>
<code>scandir(string dir [, int sorting_order])</code>	Array	Возвращает список файлов и каталогов, расположенных по указанному пути
<code>dl(string extension_filename)</code>	Int	Загружает РНР-расширение во время выполнения сценария

Обработка ошибок

Функция	Тип возвращаемых данных	Описание
<code>error_log(string message [, int message_type [, string destination [, string extra_headers]])</code>	Bool	Отправляет сообщение об ошибке
<code>debug_print_backtrace(void)</code>	Void	Распечатывает отладочную трассировку
<code>debug_backtrace(void)</code>	Array	Возвращает отладочную трассировку в виде массива
<code>restore_error_handler(void)</code>	Void	Восстанавливает предыдущий обработчик ошибок
<code>set_exception_handler(string exception_handler)</code>	String	Устанавливает пользовательский обработчик исключений; возвращает предыдущий обработчик или <code>false</code> в случае ошибки
<code>restore_exception_handler(void)</code>	Void	Восстанавливает предыдущий обработчик исключений

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>trigger_error(string message [, int error_type])</code>	Void	Генерирует ошибку/предупреждение/замечание пользовательского уровня
<code>set_error_handler(string error_handler)</code>	String	Устанавливает пользовательский обработчик ошибок; возвращает предыдущий обработчик или <code>false</code> в случае ошибки
<code>leak(int num_bytes=3)</code>	Void	Вызывает преднамеренную утечку памяти в целях отладки или тестирования

Файловая система

Функция	Тип возвращаемых данных	Описание
<code>flock(resource fp, int operation [, int &wouldblock])</code>	Bool	Кроссплатформенная блокировка файлов
<code>file_get_contents(string filename [, bool use_include_path [, resource context]])</code>	String	Считывает весь файл в строку
<code>file_put_contents(string file, mixed data [, int flags [, resource context]])</code>	Int	Записывает/создает файл с содержимым <code>data</code> и возвращает количество записанных байт
<code>file(string filename [, int flags [, resource context]])</code>	Array	Читает весь файл в массив
<code>tempnam(string dir, string prefix)</code>	String	Создает временный файл с уникальным именем в указанном каталоге
<code>tmpfile(void)</code>	resource	Создает временный файл, который будет автоматически удален после использования
<code>fopen(string filename, string mode [, bool use_include_path [, resource context]])</code>	resource	Открывает файл или URL и возвращает указатель
<code>fclose(resource fp)</code>	Bool	Закрывает открытый указатель файла
<code>popen(string command, string mode)</code>	resource	Выполняет команду и отрывает канал для чтения или записи в него
<code>pclose(resource fp)</code>	Int	Закрывает файловый указатель, созданный функцией <code>popen()</code>
<code>feof(resource fp)</code>	Bool	Проверяет конец файла для заданного указателя файла
<code>fgets(resource fp[, int length])</code>	String	Получает строку из указателя файла

Функция	Тип возвращаемых данных	Описание
<code>fgetc(resource fp)</code>	String	Получает символ из указателя файла
<code>fgetss(resource fp [, int length, string allowable_tags])</code>	String	Получает строку из указателя файла и удаляет HTML-теги
<code>fscanf(resource stream, string format [, string. . .])</code>	Mixed	Реализует обычно ANSI-совместимую функцию <code>fscanf()</code>
<code>fwrite(resource fp, string str [, int length])</code>	Int	Запись файла безопасная для двоичных данных
<code>fflush(resource fp)</code>	Bool	Сбрасывает буфер в файл
<code>rewind(resource fp)</code>	Bool	Возвращает указатель позиции в файле в начало
<code>ftell(resource fp)</code>	Int	Сообщает позицию указателя в файле
<code>fseek(resource fp, int offset [, int whence])</code>	Int	Ищет позицию указателя в файле
<code>mkdir(string pathname [, int mode [, bool recursive [, resource context]])</code>	Bool	Создает каталог
<code>rmdir(string dirname[, resource context])</code>	Bool	Удаляет каталог
<code>readfile(string filename [, bool use_include_path [, resource context]])</code>	Int	Читает файл или URL и записывает его в буфер вывода
<code>umask([int mask])</code>	Int	Возвращает или изменяет текущие <code>umask</code> -настройки
<code>fpassthru(resource fp)</code>	Int	Выводит все оставшиеся данные из файлового указателя
<code>rename(string old_name, string new_name[, resource context])</code>	Bool	Переименовывает файл
<code>unlink(string filename[, context context])</code>	Bool	Удаляет файл
<code>ftruncate(resource fp, int size)</code>	Bool	Усекает файл до заданного размера
<code>fstat(resource fp)</code>	Int	Получает <code>stat()</code> -информацию о файле, используя открытый файловый указатель
<code>copy(string source_file, string destination_file)</code>	Bool	Копирует файл
<code>fread(resource fp, int length)</code>	String	Безопасное для двоичных данных чтение файла
<code>fgetcsv(resource fp [,int length [, string delimiter [, string enclosure]])</code>	Array	Извлекает строку из файлового указателя и разбирает ее на CSV-поля
<code>realpath(string path)</code>	String	Возвращает канонический абсолютный путь

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>fnmatch(string pattern, string filename [, int flags])</code>	Bool	Сравнивает имя файла с образцом
<code>disk_total_space(string path)</code>	Float	Возвращает общий объем диска для указанной файловой системы
<code>disk_free_space(string path)</code>	Float	Возвращает объем свободного пространства диска для указанной файловой системы
<code>chgrp(string filename, mixed group)</code>	Bool	Изменяет группу владельцев файла
<code>chown (string filename, mixed user)</code>	Bool	Изменяет владельца файла
<code>chmod(string filename, int mode)</code>	Bool	Изменяет права доступа к файлу
<code>touch(string filename [, int time [, int atime]])</code>	Bool	Устанавливает время модификации файла
<code>clearstatcache(void)</code>	Void	Очищает stat-кэш
<code>fileperms(string filename)</code>	Int	Возвращает информацию о правах доступа к файлу
<code>fileinode(string filename)</code>	Int	Возвращает индексный узел файла
<code>filesize(string filename)</code>	Int	Определяет размер файла
<code>fileowner(string filename)</code>	Int	Получает идентификатор владельца файла
<code>filegroup(string filename)</code>	Int	Получает идентификатор группы владельцев файла
<code>fileatime(string filename)</code>	Int	Получает время последнего доступа к файлу
<code>filemtime(string filename)</code>	Int	Получает время последнего изменения файла
<code>filectime(string filename)</code>	Int	Получает время изменения индексного узла
<code>filetype(string filename)</code>	String	Получает тип файла
<code>is_writable(string filename)</code>	Bool	Сообщает, можно ли записывать данные в указанный файл
<code>is_readable(string filename)</code>	Bool	Сообщает, является ли заданный файл читабельным
<code>is_executable(string filename)</code>	Bool	Сообщает, является ли указанный файл исполняемым
<code>is_file(string filename)</code>	Bool	Сообщает, является ли указанный файл обычным файлом
<code>is_dir(string filename)</code>	Bool	Сообщает, является ли значение параметра <code>filename</code> каталогом

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>is_link(string filename)</code>	Bool	Сообщает, является ли значение параметра <code>filename</code> символической ссылкой
<code>file_exists(string filename)</code>	Bool	Проверяет, существует ли файл
<code>lstat(string filename)</code>	Array	Возвращает информацию о файле или символической ссылке
<code>stat(string filename)</code>	Array	Возвращает информацию о файле
<code>readlink(string filename)</code>	String	Возвращает имя файла, на который указывает символическая ссылка
<code>linkinfo(string filename)</code>	Int	Возвращает значение поля <code>st_dev</code> из описывающей ссылку структуры, возвращаемой Unix C-функцией <code>lstat</code>
<code>symlink(string target, string link)</code>	Int	Создает символическую ссылку
<code>link(string target, string link)</code>	Int	Создает жесткую ссылку
<code>is_uploaded_file(string path)</code>	Bool	Определяет, был ли файл загружен при помощи HTTP POST (RFC1867)
<code>move_uploaded_file(string path, string new_path)</code>	Bool	Перемещает загруженный файл в новое место
<code>parse_ini_file(string filename [, bool process_sections])</code>	Array	Разбирает файл конфигурации

FTP-функции

Функция	Тип возвращаемых данных	Описание
<code>ftp_connect(string host [, int port [, int timeout]])</code>	resource	Открывает FTP-соединение
<code>ftp_ssl_connect(string host [, int port [, int timeout]])</code>	resource	Открывает FTP-SSL-соединение
<code>ftp_login(resource stream, string username, string password)</code>	Bool	Регистрируется на FTP-сервере
<code>ftp_pwd(resource stream)</code>	String	Возвращает имя текущего рабочего каталога
<code>ftp_cdup(resource stream)</code>	Bool	Переходит в родительский каталог
<code>ftp_chdir(resource stream, string directory)</code>	Bool	Меняет каталог
<code>ftp_exec(resource stream, string command)</code>	Bool	Запрашивает выполнение программы на FTP-сервере

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>ftp_raw(resource stream, string command)</code>	Array	Отправляет заданную команду FTP-серверу
<code>ftp_mkdir(resource stream, string directory)</code>	String	Создает каталог и возвращает абсолютный путь к нему или false в случае ошибки
<code>ftp_rmdir(resource stream, string directory)</code>	Bool	Удаляет каталог
<code>ftp_chmod(resource stream, int mode, string filename)</code>	Int	Устанавливает права доступа к файлу
<code>ftp_alloc(resource stream, int size[, &response])</code>	Bool	Пытается зарезервировать для закачиваемого файла место на диске удаленного FTP-сервера
<code>ftp_nlist(resource stream, string directory)</code>	Array	Возвращает массив имен файлов в заданном каталоге
<code>ftp_rawlist(resource stream, string directory [, bool recursive])</code>	Array	Возвращает в виде массива подробный список файлов в заданном каталоге
<code>ftp_systype(resource stream)</code>	String	Возвращает тип операционной системы FTP сервера
<code>ftp_fget(resource stream, resource fp, string remote_file, int mode[, int resumepos])</code>	Bool	Загружает файл с FTP-сервера и сохраняет его в предварительно открытом файле
<code>ftp_nb_fget(resource stream, resource fp, string remote_file, int mode[, int resumepos])</code>	Int	Загружает файл с FTP сервера в асинхронном режиме и сохраняет его в предварительно открытом файле
<code>ftp_pasv(resource stream, bool pasv)</code>	Bool	Включает или выключает пассивный режим
<code>ftp_get(resource stream, string local_file, string remote_file, int mode[, int resume_pos])</code>	Bool	Загружает файл с FTP-сервера и записывает его в локальный файл
<code>ftp_nb_get(resource stream, string local_file, string remote_file, int mode[, int resume_pos])</code>	Int	Загружает файл с FTP-сервера в асинхронном режиме и сохраняет его в локальный файл
<code>ftp_nb_continue(resource stream)</code>	Int	Продолжает отправку или получение файла в асинхронном режиме
<code>ftp_fput(resource stream, string remote_file, resource fp, int mode[, int startpos])</code>	Bool	Загружает данные из открытого файла на FTP-сервер
<code>ftp_nb_fput(resource stream, string remote_file, resource fp, int mode[, int startpos])</code>	Int	Загружает данные из открытого файла на FTP сервер в асинхронном режиме

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>ftp_put(resource stream, string remote_file, string local_file, int mode[, int startpos])</code>	Bool	Загружает файл на FTP-сервер
<code>ftp_nb_put(resource stream, string remote_file, string local_file, int mode[, int startpos])</code>	Int	Загружает файл на FTP-сервер в асинхронном режиме
<code>ftp_size(resource stream, string filename)</code>	Int	Возвращает размер заданного файла или -1 в случае ошибки
<code>ftp_mdtm(resource stream, string filename)</code>	Int	Возвращает время последней модификации заданного файла или -1 в случае ошибки
<code>ftp_rename(resource stream, string src, string dest)</code>	Bool	Переименовывает файл на FTP-сервере
<code>ftp_delete(resource stream, string file)</code>	Bool	Удаляет файл на FTP-сервере
<code>ftp_site(resource stream, string cmd)</code>	Bool	Отправляет команду SITE на сервер
<code>ftp_close(resource stream)</code>	Bool	Закрывает FTP-соединение
<code>ftp_set_option(resource stream, int option, mixed value)</code>	Bool	Устанавливает параметры соединения с FTP-сервером
<code>ftp_get_option(resource stream, int option)</code>	Mixed	Получает текущие параметры FTP-соединения

Вызов функций

Функция	Тип возвращаемых данных	Описание
<code>call_user_func(string function_name [, mixed parameter] [, mixed. . .])</code>	Mixed	Вызывает пользовательскую функцию, заданную первым параметром
<code>call_user_func_array(string function_name, array parameters)</code>	Mixed	Вызывает заданную первым параметром пользовательскую функцию и передает ей массив параметров
<code>call_user_method(string method_name, mixed object [, mixed parameter] [, mixed. . .])</code>	Mixed	Вызывает пользовательский метод на заданном объекте или классе
<code>call_user_method_array(string method_name, mixed object, array params)</code>	Mixed	Вызывает пользовательский метод на заданном объекте или классе и передает ему массив параметров

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>register_shutdown_function</code> (string function_name)	Void	Регистрирует пользовательскую функцию, которая выполняется по завершению работы сценария
<code>register_tick_function</code> (string function_name [, mixed arg [, mixed. . .]])	Bool	Регистрирует функцию обратного вызова для выполнения при каждом такте
<code>unregister_tick_function</code> (string function_name)	Void	Разрегистраует функцию обратного вызова для выполнения при каждом такте
<code>create_function</code> (string args, string code)	String	Создает анонимную функцию и возвращает ее имя
<code>function_exists</code> (string function_name)	Bool	Проверяет существование функции
<code>func_num_args</code> (void)	Int	Возвращает количество переданных функции аргументов
<code>func_get_arg</code> (int arg_num)	Mixed	Возвращает элемент с индексом \$arg_num из списка переданных функции аргументов
<code>func_get_args</code> ()	Array	Возвращает массив переданных функции аргументов

HTTP-функции

Функция	Тип возвращаемых данных	Описание
<code>header</code> (string header [, bool replace, [int http_response_code]])	Void	Отправляет необработанный HTTP-заголовок
<code>setcookie</code> (string name [, string value [, int expires [, string path [, string domain [, bool secure]]]])	Bool	Отправляет cookie-файл
<code>setrawcookie</code> (string name [, string value [, int expires [, string path [, string domain [, bool secure]]]])	Bool	Устанавливает cookie-файл без url-кодирования его значения
<code>headers_sent</code> ([string \$file [, int \$line]])	Bool	Возвращает true, если заголовки уже были отправлены, и false в противном случае
<code>headers_list</code> (void)	String	Возвращает список отправленных (или готовых к отправке) HTTP-заголовков

Библиотека iconv

Функция	Тип возвращаемых данных	Описание
iconv(string in_charset, string out_charset, string str)	String	Конвертирует строку в заданную кодировку out_charset символов
ob_iconv_handler (string contents, int status)		Возвращает строку в выходном буфере, преобразованную в кодировку iconvoutput_encoding
iconv_get_encoding ([string type])	Mixed	Возвращает внутреннюю и внешнюю кодировки для обработчика ob_iconv_handler()
iconv_set_encoding (string type, string charset)	Bool	Устанавливает внутреннюю и внешнюю кодировки для обработчика ob_iconv_handler()

Функции для работы с изображениями

Функция	Тип возвращаемых данных	Описание
exif_tagname (index)	String	Возвращает имя заголовка для индекса или false, если заголовок не определен
exif_read_data (string filename [, sections_needed [, sub_arrays [, read_thumbnail]]])	Array	Считывает данные заголовков из JPEG/TIFF-изображений и внутренние пиктограммы (если требуется)
exif_thumbnail (string filename [, &width, &height [, imagetype]])	String	Считывает внедренную пиктограмму TIFF- или JPEG-изображения
exif_imagetype (string imagefile)	Int	Определяет тип изображения
gd_info ()	Array	Возвращает информацию об установленной библиотеке GD
imageloadfont (string filename)	Int	Загружает новый шрифт
imagesetstyle (resource im, array styles)	Bool	Устанавливает стиль рисования линии для использования в функции imageline с указанным цветом IMG_COLOR_STYLED
imagecreatetruecolor (int x_size, int y_size)	resource	Создает новое полноцветное изображение true color
imageistruecolor (resource im)	Bool	Возвращает true, если заданное изображение полноцветное
imagetruecolortopalette (resource im, bool ditherFlag, int colorsWanted)	Void	Конвертирует полноцветное изображение в индексированное изображение с определенным количеством цветов и сглаживанием (необязательно)

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>imagecolormatch(resource im1, resource im2)</code>	Bool	Устанавливает цвета индексированной версии более близкими к цветам полноцветной версии изображения
<code>imagesetthickness(resource im, int thickness)</code>	Bool	Устанавливает толщину рисования линий, эллипсов, прямоугольников, многоугольников и т.д.
<code>imagefilledellipse(resource im, int cx, int cy, int w, int h, int color)</code>	Bool	Рисует эллипс
<code>imagefilledarc(resource im, int cx, int cy, int w, int h, int s, int e, int col, int style)</code>	Bool	Рисует часть эллипса и заполняет ее
<code>imagealphablending(resource im, bool on)</code>	Bool	Включает или выключает режим альфа-сопряжения для изображения
<code>imagesavealpha(resource im, bool on)</code>	Bool	Включает информацию об альфа-каналах в сохраняемое изображение
<code>imagelayereffect(resource im, int effect)</code>	Bool	Устанавливает флаг альфа-сопряжения для использования встроенных libgd-эффектов слоев
<code>imagecolorallocatealpha(resource im, int red, int green, int blue, int alpha)</code>	Int	Выделяет цвет с альфа-слоем; работает для полноцветных и индексированных изображений
<code>imagecolorresolvealpha(resource im, int red, int green, int blue, int alpha)</code>	Int	Определяет цвет и альфа-слой; работает для полноцветных и индексированных изображений
<code>imagecolorclosestalpha(resource im, int red, int green, int blue, int alpha)</code>	Int	Определяет ближайший к заданному цвет с альфа-прозрачностью
<code>imagecolorexactalpha(resource im, int red, int green, int blue, int alpha)</code>	Int	Определяет точный индекс заданного цвета с прозрачностью
<code>imagecopyresampled(resource dst_im, resource src_im, int dst_x, int dst_y, int src_x, int src_y, int dst_w, int dst_h, int src_w, int src_h)</code>	Bool	Копирует и изменяет размеры части изображения, изменяя разрешение для обеспечения четкости
<code>imagerotate(resource src_im, float angle, int bgdcolor)</code>	resource	Поворачивает изображение на заданный угол
<code>imagesettile(resource image, resource tile)</code>	Bool	Устанавливает изображение \$tile для фонового заполнения изображения \$image цветом IMG_COLOR_TILED
<code>imagesetbrush(resource image, resource brush)</code>	Bool	Устанавливает изображение кисти для прорисовки линий цветом IMG_COLOR_BRUSHED

Функция	Тип возвращаемых данных	Описание
<code>imagecreate(int x_size, int y_size)</code>	resource	Создает новое индексированное изображение
<code>imagetypes(void)</code>	Int	Возвращает в битовом поле типы поддерживаемых изображений: 1=GIF, 2=JPEG, 4=PNG, 8=WBMP, 16=XPM
<code>imagecreatefromstring(string image)</code>	resource	Создает новое изображение из потока, заданного строкой
<code>imagecreatefromgif(string filename)</code>	resource	Создает новое изображение из GIF-файла или URL
<code>imagecreatefromjpeg(string filename)</code>	resource	Создает новое изображение из файла или URL
<code>imagecreatefrompng(string filename)</code>	resource	Создает новое изображение из PNG-файла или URL
<code>imagecreatefromxbm(string filename)</code>	resource	Создает новое изображение из XBM-файла или URL
<code>imagecreatefromxpm(string filename)</code>	resource	Создает новое изображение из XPM-файла или URL
<code>imagecreatefromwbmp(string filename)</code>	resource	Создает новое изображение из WBMP-файла или URL
<code>imagecreatefromgd(string filename)</code>	resource	Создает новое изображение из GD-файла или URL
<code>imagecreatefromgd2(string filename)</code>	resource	Создает новое изображение из GD2-файла или URL
<code>imagecreatefromgd2part(string filename, int srcX, int srcY, int width, int height)</code>	resource	Создает новое изображение из заданной части GD2-файла или URL
<code>imagexbm(int im, string filename [, int foreground])</code>	Int	Выводит XBM-изображение в браузер или файл
<code>imagegif(resource im [, string filename])</code>	Bool	Выводит GIF-изображение в браузер или файл
<code>imagepng(resource im [, string filename])</code>	Bool	Выводит PNG-изображение в браузер или файл
<code>imagejpeg(resource im [, string filename [, int quality]])</code>	Bool	Выводит JPEG-изображение в браузер или файл
<code>imagewbmp(resource im [, string filename [, int foreground]])</code>	Bool	Выводит WBMP-изображение в браузер или файл
<code>imagegd(resource im [, string filename])</code>	Bool	Выводит GD-изображение в браузер или файл
<code>imagegd2(resource im [, string filename [, int chunk_size [, int type]]])</code>	Bool	Выводит GD2-изображение в браузер или файл
<code>imagedestroy(resource im)</code>	Bool	Уничтожает изображение

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
imagecolorallocate(resource im, int red, int green, int blue)	Int	Выделяет цвет для изображения
imagepalettecopy(resource dst, resource src)	Void	Копирует палитру одного изображения в другое
imagecolorat(resource im, int x, int y)	Int	Получает индекс цвета пикселя
imagecolorclosest(resource im, int red, int green, int blue)	Int	Получает индекс цвета, ближайшего к заданному
imagecolorclosesthw (resource im, int red, int green, int blue)	Int	Получает индекс цвета, имеющего оттенок, значения белого и черного цветов, ближайшие к заданному цвету
imagecolordeallocate(resource im, int index)	Bool	Отменяет выделение цвета для изображения
imagecolorresolve(resource im, int red, int green, int blue)	Int	Получает индекс заданного цвета или его ближайшей возможной альтернативы
imagecolorexact(resource im, int red, int green, int blue)	Int	Получает индекс заданного цвета
imagecolorset(resource im, int col, int red, int green, int blue)	Void	Устанавливает цвет для заданного индекса палитры
imagecolorsforindex(resource im, int col)	Array	Получает цвет для индекса
imagegammacorrect(resource im, float inputgamma, float outputgamma)	Bool	Применяет гамма-коррекцию к GD-изображению
imagegetpixel(resource im, int x, int y, int col)	Bool	Устанавливает одиночный пиксель
imageline(resource im, int x1, int y1, int x2, int y2, int col)	Bool	Рисует линию
imagedashedline(resource im, int x1, int y1, int x2, int y2, int col)	Bool	Рисует пунктирную линию
imagerectangle(resource im, int x1, int y1, int x2, int y2, int col)	Bool	Рисует прямоугольник
imagefilledrectangle(resource im, int x1, int y1, int x2, int y2, int col)	Bool	Рисует заполненный прямоугольник
imagearc(resource im, int cx, int cy, int w, int h, int s, int e, int col)	Bool	Рисует дугу эллипса
imageellipse(resource im, int cx, int cy, int w, int h, int color)	Bool	Рисует эллипс

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>imagefilltoborder(resource im, int x, int y, int border, int col)</code>	Bool	Заливает изображение заданным цветом
<code>imagefill(resource im, int x, int y, int col)</code>	Bool	Заливает изображение заданным цветом
<code>imagecolorstotal(resource im)</code>	Int	Определяет количество цветов палитры изображения
<code>imagecolortransparent(resource im [, int col])</code>	Int	Определяет цвет как прозрачный
<code>imageinterlace(resource im [, int interlace])</code>	Int	Включает/отключает чередование
<code>imagepolygon(resource im, array point, int num_points, int col)</code>	Bool	Рисует многоугольник
<code>imagefilledpolygon(resource im, array point, int num_points, int col)</code>	Bool	Рисует заполненный многоугольник
<code>imagefontwidth(int font)</code>	Int	Получает ширину шрифта
<code>imagefontheight(int font)</code>	Int	Получает высоту шрифта
<code>imagechar(resource im, int font, int x, int y, string c, int col)</code>	Bool	Рисует символ горизонтально
<code>imagecharup(resource im, int font, int x, int y, string c, int col)</code>	Bool	Рисует символ вертикально под углом 90 градусов против часовой стрелки
<code>imagestring(resource im, int font, int x, int y, string str, int col)</code>	Bool	Рисует строку горизонтально
<code>imagestringup(resource im, int font, int x, int y, string str, int col)</code>	Bool	Рисует строку вертикально, под углом 90 градусов против часовой стрелки
<code>imagecopy(resource dst_im, resource src_im, int dst_x, int dst_y, int src_x, int src_y, int src_w, int src_h)</code>	Bool	Копирует часть изображения
<code>imagecopymerge(resource src_im, resource dst_im, int dst_x, int dst_y, int src_x, int src_y, int src_w, int src_h, int pct)</code>	Bool	Объединяет части изображений
<code>imagecopymergegray(resource src_im, resource dst_im, int dst_x, int dst_y, int src_x, int src_y, int src_w, int src_h, int pct)</code>	Bool	Соединяет части изображения с сохранением оттенка исходного изображения

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>imagecopyresized(resource dst_im, resource src_im, int dst_x, int dst_y, int src_x, int src_y, int dst_w, int dst_h, int src_w, int src_h)</code>	Bool	Копирует и изменяет размеры части изображения
<code>imagesx(resource im)</code>	Int	Получает ширину изображения
<code>imagesy(resource im)</code>	Int	Получает высоту изображения
<code>imageftbbox(int size, int angle, string font_file, string text[, array extrainfo])</code>	Array	Вычисляет обрамление для текста с использованием шрифтов freetype2
<code>imagefttext(resource im, int size, int angle, int x, int y, int col, string font_file, string text, [array extrainfo])</code>	Array	Записывает текст в изображение с использованием шрифтов freetype2
<code>imaggettfbbox(int size, int angle, string font_file, string text)</code>	Array	Вычисляет обрамление для текста с использованием truetype-шрифтов
<code>imaggettfttext(resource im, int size, int angle, int x, int y, int col, string font_file, string text)</code>	Array	Записывает текст в изображение с использованием truetype-шрифта
<code>imagepsloadfont(string pathname)</code>	Resource	Загружает новый шрифт из указанного файла
<code>imagepscopyfont(int font_index)</code>	Int	Создает копию шрифта с целью расширения или перекодировки
<code>imagepsfreefont(resource font_index)</code>	Bool	Высвобождает память, используемую шрифтом
<code>imagepsencodefont(resource font_index, string filename)</code>	Bool	Изменяет вектор кодировки символов шрифта
<code>imagepsextendfont(resource font_index, float extend)</code>	Bool	Расширяет или сжимает (если <code>extend < 1</code>) шрифт
<code>imagepsslantfont(resource font_index, float slant)</code>	Bool	Наклоняет шрифт
<code>imagepstext(resource image, string text, resource font, int size, int xcoord, int ycoord[, int space, int tightness, float angle, int antialias])</code>	Array	Прорисовывает текстовую строку поверх изображения
<code>imagepsbbox(string text, resource font, int size[, int space, int tightness, int angle])</code>	Array	Вычисляет обрамление необходимое для прорисованной строки
<code>image2wbmp(resource im[, string filename[, int threshold]])</code>	Bool	Выводит WBMP-изображение в браузер или в файл

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>jpeg2wbmp (string f_org, string f_dest, int d_height, int d_width, int threshold)</code>	Bool	Конвертирует JPEG-изображение в WBMP-изображение
<code>png2wbmp (string f_org, string f_dest, int d_height, int d_width, int threshold)</code>	Bool	Конвертирует PNG-изображение в WBMP-изображение
<code>imagefilter(resource src_im, int filtertype, [args])</code>	Bool	Применяет фильтр к изображению, используя заданный угол
<code>imageantialias(resource im, bool on)</code>	Bool	Определяет, следует ли использовать функции сглаживания
<code>image_type_to_mime_type (int imagetype)</code>	String	Определяет MIME-тип для типа изображения, возвращаемого функциями <code>getimagesize</code> , <code>exif_read_data</code> , <code>exif_thumbnail</code> , <code>exif_imagetype</code>
<code>image_type_to_extension (int imagetype [, bool include_dot])</code>	String	Определяет расширение для типа изображения, возвращаемого функциями <code>getimagesize</code> , <code>exif_read_data</code> , <code>exif_thumbnail</code> , <code>exif_imagetype</code>
<code>getimagesize(string imagefile [, array info])</code>	Array	Определяет размер изображения в виде четырехэлементного массива
<code>iptcembed(string iptcdata, string jpeg_file_name [, int spool])</code>	Array	Внедряет двоичные IPTC-данные в JPEG-изображение
<code>iptcparse(string iptcdata)</code>	Array	Разбирает двоичный IPTC-блок в ассоциативный массив

IMAP-функции

Функция	Тип возвращаемых данных	Описание
<code>imap_open(string mailbox, string user, string password [, int options])</code>	resource	Открывает IMAP-поток к почтовому ящику
<code>imap_reopen(resource stream_id, string mailbox [, int options])</code>	Bool	Повторно открывает IMAP-поток к новому почтовому ящику
<code>imap_append(resource stream_id, string folder, string message [, string options])</code>	Bool	Добавляет новое сообщение в заданный почтовый ящик
<code>imap_num_msg(resource stream_id)</code>	Int	Выдает количество сообщений в текущем почтовом ящике
<code>imap_ping(resource stream_id)</code>	Bool	Проверяет, активен ли IMAP-поток
<code>imap_num_recent(resource stream_id)</code>	Int	Возвращает количество свежих сообщений в текущем почтовом ящике

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>imap_get_quota(resource stream_id, string qroot)</code>	Array	Возвращает квоту для почтового ящика <code>qroot</code>
<code>imap_get_quotaroot(resource stream_id, string mbox)</code>	Array	Возвращает квоту для почтового ящика <code>mbox</code>
<code>imap_set_quota(resource stream_id, string qroot, int mailbox_size)</code>	Bool	Устанавливает квоту для почтового ящика <code>qroot</code>
<code>imap_setacl(resource stream_id, string mailbox, string id, string rights)</code>	Bool	Устанавливает ACL для заданного почтового ящика
<code>imap_getacl(resource stream_id, string mailbox)</code>	Array	Возвращает ACL для заданного почтового ящика
<code>imap_expunge(resource stream_id)</code>	Bool	Удаляет все сообщения, помеченные для удаления
<code>imap_close(resource stream_id [, int options])</code>	Bool	Закрывает IMAP-поток
<code>imap_headers(resource stream_id)</code>	Array	Возвращает заголовки всех сообщений в почтовом ящике
<code>imap_body(resource stream_id, int msg_no [, int options])</code>	String	Читает тело сообщения
<code>imap_mail_copy(resource stream_id, int msg_no, string mailbox [, int options])</code>	Bool	Копирует заданное сообщение в почтовый ящик
<code>imap_mail_move(resource stream_id, int msg_no, string mailbox [, int options])</code>	Bool	Перемещает заданные сообщения в почтовый ящик
<code>imap_createmailbox(resource stream_id, string mailbox)</code>	Bool	Создает новый почтовый ящик
<code>imap_renamemailbox(resource stream_id, string old_name, string new_name)</code>	Bool	Переименовывает почтовый ящик
<code>imap_deletemailbox(resource stream_id, string mailbox)</code>	Bool	Удаляет почтовый ящик
<code>imap_list(resource stream_id, string ref, string pattern)</code>	Array	Считывает список почтовых ящиков
<code>imap_getmailboxes(resource stream_id, string ref, string pattern)</code>	Array	Читает список почтовых ящиков и возвращает массив с подробной информацией о каждом почтовом ящике (имя, атрибуты, разделитель)
<code>imap_scanmailbox (resource stream_id, string ref, string pattern, string content)</code>	Array	Считывает список почтовых ящиков, содержащих определенную строку
<code>imap_check(resource stream_id)</code>	Object	Возвращает свойства текущего почтового ящика

Функция	Тип возвращаемых данных	Описание
<code>imap_delete(resource stream_id, int msg_no [, int options])</code>	Bool	Отмечает сообщение для удаления
<code>imap_undelete(resource stream_id, int msg_no)</code>	Bool	Снимает с сообщения отметку на удаление
<code>imap_headerinfo(resource stream_id, int msg_no [, int from_length [, int subject_length [, string default_host]])</code>	Object	Читает заголовки сообщения
<code>imap_rfc822_parse_headers (string headers [, string default_host])</code>	Object	Анализирует почтовые заголовки, содержащиеся в строке, и возвращает объект, аналогичный объекту, возвращаемому функцией <code>imap_headerinfo()</code>
<code>imap_lsub(resource stream_id, string ref, string pattern)</code>	Array	Возвращает список подписанных почтовых ящиков
<code>imap_getsubscribed(resource stream_id, string ref, string pattern)</code>	Array	Возвращает список всех подписанных почтовых ящиков в том же формате, что и <code>imap_getmailboxes()</code>
<code>imap_subscribe(resource stream_id, string mailbox)</code>	Bool	Подписывает на почтовый ящик
<code>imap_unsubscribe(resource stream_id, string mailbox)</code>	Bool	Отменяет подписку на почтовый ящик
<code>imap_fetchstructure(resource stream_id, int msg_no [, int options])</code>	Object	Считывает всю структуру сообщения
<code>imap_fetchbody(resource stream_id, int msg_no, int section [, int options])</code>	String	Извлекает определенный раздел тела сообщения
<code>imap_base64(string text)</code>	String	Декодирует BASE64-кодированный текст
<code>imap_qprint(string text)</code>	String	Конвертирует строку в кавычках в 8-битовую строку
<code>imap_8bit(string text)</code>	String	Конвертирует 8-битовую строку в строку в кавычках
<code>imap_binary(string text)</code>	String	Конвертирует 8-битовую строку в строку base64
<code>imap_mailboxmsginfo(resource stream_id)</code>	Object	Возвращает информацию о текущем почтовом ящике
<code>imap_rfc822_write_address (string mailbox, string host, string personal)</code>	String	Возвращает правильно сформированный e-mail-адрес для заданного почтового ящика, узла и персональной информации
<code>imap_rfc822_parse_adrlist (string address_string, string default_host)</code>	Array	Разбирает строку адреса

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>imap_utf8(string mime_encoded_text)</code>	String	Конвертирует MIME-текст в UTF-8
<code>imap_utf7_decode(string buf)</code>	String	Декодирует модифицированную UTF-7-кодированную строку
<code>imap_utf7_encode(string buf)</code>	String	Кодирует строку в модифицированную UTF-7-строку
<code>imap_setflag_full(resource stream_id, string sequence, string flag [, int options])</code>	Bool	Устанавливает флаги сообщений
<code>imap_clearflag_full(resource stream_id, string sequence, string flag [, int options])</code>	Bool	Очищает флаги сообщений
<code>imap_sort(resource stream_id, int criteria, int reverse [, int options [, string search_criteria [, string charset]]])</code>	Array	Сортирует массив заголовков сообщения; если указать необязательный критерий, то будут использоваться только сообщения, соответствующие этому критерию
<code>imap_fetchheader(resource stream_id, int msg_no [, int options])</code>	String	Возвращает полный заголовок сообщения
<code>imap_uid(resource stream_id, int msg_no)</code>	Int	Возвращает уникальный идентификатор сообщения, связанный со стандартным последовательным номером сообщения
<code>imap_msgno(resource stream_id, int unique_msg_id)</code>	Int	Возвращает последовательный номер сообщения, связанного с заданным UID-идентификатором
<code>imap_status(resource stream_id, string mailbox, int options)</code>	Object	Возвращает информацию о состоянии почтового ящика
<code>imap_bodystruct(resource stream_id, int msg_no, int section)</code>	Object	Читает структуру заданного раздела тела заданного сообщения
<code>imap_fetch_overview (resource stream_id, int msg_no [, int options])</code>	Array	Возвращает общие сведения в заголовках заданной последовательности сообщений
<code>imap_mail_compose(array envelope, array body)</code>	String	Создает MIME-сообщение на основе заданного конверта и тела
<code>imap_mail(string to, string subject, string message [, string additional_headers [, string cc [, string bcc [, string rpath]]]])</code>	Bool	Отправляет email-сообщение
<code>imap_search(resource stream_id, string criteria [, int options [, string charset]])</code>	Array	Возвращает массив сообщений, совпадающих с заданным критерием поиска

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>imap_alerts(void)</code>	Array	Возвращает массив всех IMAP-предупреждений, которые были сгенерированы с момента последней загрузки страницы или с момента последнего вызова <code>imap_alerts()</code> (в зависимости от того, какое событие произошло позднее); стек предупреждений очищается после вызова функции <code>imap_alerts()</code>
<code>imap_errors(void)</code>	Array	Возвращает массив всех IMAP-ошибок, которые были сгенерированы с момента последней загрузки страницы или с момента последнего вызова <code>imap_errors()</code> (в зависимости от того, какое событие произошло позднее); стек ошибок очищается после вызова функции <code>imap_errors()</code>
<code>imap_last_error(void)</code>	String	Возвращает последнюю ошибку, сгенерированную IMAP-функцией; стек ошибок после вызова этой функции не очищается
<code>imap_mime_header_decode(string str)</code>	Array	Декодирует элемент MIME-заголовка в соответствии с RFC 2047 и возвращает массив объектов, содержащий кодированный и декодированный текст
<code>imap_thread(resource stream_id [, int options])</code>	Array	Возвращает дерево связанных сообщений
<code>imap_timeout(int timeout_type [, int timeout])</code>	Mixed	Устанавливает или возвращает IMAP-таймаут

Mail-функции

Функция	Тип возвращаемых данных	Описание
<code>ezmlm_hash(string addr)</code>	Int	Вычисляет значение хеша, нужное для EZMLM
<code>mail(string to, string subject, string message [, string additional_headers [, string additional_parameters]])</code>	Int	Отправляет почтовое сообщение

Math-функции

Функция	Тип возвращаемых данных	Описание
<code>abs(int number)</code>	Int	Возвращает абсолютную величину числа
<code>ceil(float number)</code>	Float	Округляет дробь в сторону увеличения
<code>floor(float number)</code>	Float	Округляет дробь в сторону уменьшения
<code>round(float number [, int precision])</code>	Float	Округляет число до заданной точности
<code>sin(float number)</code>	Float	Возвращает синус от number радиан
<code>cos(float number)</code>	Float	Возвращает косинус от number радиан
<code>tan(float number)</code>	Float	Возвращает тангенс от number радиан
<code>asin(float number)</code>	Float	Возвращает арксинус от number радиан
<code>acos(float number)</code>	Float	Возвращает арккосинус от number радиан
<code>atan(float number)</code>	Float	Возвращает арктангенс от number радиан
<code>atan2(float y, float x)</code>	Float	Возвращает арктангенс y/x с результирующим квадрантом, определяемым знаками y и x quadrant
<code>sinh(float number)</code>	Float	Возвращает гиперболический синус, определяемый как $(\exp(\text{number}) - \exp(-\text{number}))/2$
<code>cosh(float number)</code>	Float	Возвращает гиперболический косинус, определяемый как $(\exp(\text{number}) + \exp(-\text{number}))/2$
<code>tanh(float number)</code>	Float	Возвращает гиперболический тангенс, определяемый как $\sinh(\text{number})/\cosh(\text{number})$
<code>asinh(float number)</code>	Float	Возвращает инверсный гиперболический синус числа, т.е. значение, гиперболический синус которого равен заданному числу
<code>acosh(float number)</code>	Float	Возвращает инверсный гиперболический косинус числа, т.е. значение, гиперболический косинус которого равен заданному числу
<code>atanh(float number)</code>	Float	Возвращает инверсный гиперболический тангенс, т.е. значение, гиперболический тангенс которого равен заданному числу
<code>pi(void)</code>	Float	Возвращает число пи
<code>is_finite(float val)</code>	Bool	Возвращает true, если аргумент является конечным числом
<code>is_infinite(float val)</code>	Bool	Возвращает true, если аргумент является бесконечным числом
<code>is_nan(float val)</code>	Bool	Возвращает true, если аргумент не является числом
<code>pow(number base, number exponent)</code>	Number	Возвращает число base, возведенное в степень exp; если возможно, функция возвращает целое число
<code>exp(float number)</code>	Float	Возвращает число e, возведенное в степень number
<code>expm1(float number)</code>	Float	Возвращает $\exp(\text{number}) - 1$, вычисленное точно, даже если значение number близко к нулю

Функция	Тип возвращаемых данных	Описание
<code>log1p(float number)</code>	Float	Возвращает $\log(1+number)$, вычисленное точно, даже если значение <code>number</code> близко к нулю
<code>log(float number, [float base])</code>	Float	Возвращает натуральный логарифм, или логарифм по заданному основанию <code>base</code>
<code>log10(float number)</code>	Float	Возвращает логарифм <code>number</code> по основанию 10
<code>sqrt(float number)</code>	Float	Вычисляет квадратный корень <code>number</code>
<code>hypot(float num1, float num2)</code>	Float	Возвращает $\sqrt{num1^2 + num2^2}$
<code>deg2rad(float number)</code>	Float	Конвертирует градусы в радианы
<code>rad2deg(float number)</code>	Float	Конвертирует радианы в градусы
<code>bindec(string binary_number)</code>	Int	Конвертирует двоичное число в десятичное
<code>hexdec(string hexadecimal_number)</code>	Int	Конвертирует шестнадцатеричное число в десятичное
<code>octdec(string octal_number)</code>	Int	Конвертирует восьмиричное число в десятичное
<code>decbin(int decimal_number)</code>	String	Конвертирует десятичное число в двоичное
<code>decoct(int decimal_number)</code>	String	Конвертирует десятичное число в восьмиричное
<code>dechex(int decimal_number)</code>	String	Конвертирует десятичное число в шестнадцатеричное
<code>base_convert(string number, int frombase, int tobase)</code>	String	Конвертирует числа между разными основаниями (основания от 2 до 36 включительно)
<code>number_format(float number [, int num_decimal_places [, string dec_seperator, string thousands_seperator]])</code>	String	Форматирует число с группировкой по три разряда
<code>fmod(float x, float y)</code>	Float	Возвращает остаток от деления <code>x</code> на <code>y</code> как число с плавающей точкой
<code>srand([int seed])</code>	Void	Устанавливает начальное число генератора случайных чисел
<code>mt_srand([int seed])</code>	Void	Устанавливает начальное число генератора случайных чисел Mersenne Twister
<code>rand([int min, int max])</code>	Int	Генерирует случайное значение
<code>mt_rand([int min, int max])</code>	Int	Генерирует случайное значение Mersenne Twister
<code>getrandmax(void)</code>	Int	Показывает наибольшее возможное случайное значение
<code>mt_getrandmax(void)</code>	Int	Показывает наибольшее возможное из случайных значений Mersenne Twister

MIME-функции

Функция	Тип возвращаемых данных	Описание
<code>mime_content_type(string filename resource stream)</code>	String	Определяет MIME-тип содержимого файла

Разные функции

Функция	Тип возвращаемых данных	Описание
<code>get_browser([string browser_name [, bool return_array]])</code>	Mixed	Сообщает о возможностях браузера
<code>constant(string const_name)</code>	Mixed	Возвращает значение заданной константы
<code>getenv(string varname)</code>	String	Возвращает значение переменной окружения
<code>putenv(string setting)</code>	Bool	Устанавливает значение переменной окружения
<code>getopt(string options [, array longopts])</code>	Array	Возвращает параметры из списка аргументов командной строки
<code>flush(void)</code>	Void	Сбрасывает буфер вывода
<code>sleep(int seconds)</code>	Void	Замедляет выполнение на заданное количество секунд
<code>usleep(int micro_seconds)</code>	Void	Замедляет выполнение на заданное количество микросекунд
<code>time_nanosleep(long seconds, long nanoseconds)</code>	Mixed	Замедляет выполнение на заданное количество секунд и наносекунд
<code>highlight_file(string file_name [, bool return])</code>	Bool	Выделение синтаксиса файла
<code>php_strip_whitespace(string file_name)</code>	String	Возвращает исходный код без комментариев и пробельных символов
<code>php_check_syntax(string file_name [, &\$error_message])</code>	Bool	Проверяет синтаксис заданного файла
<code>highlight_string(string string [, bool return])</code>	Bool	Выделяет синтаксис строки или возвращает ее (если требуется)
<code>uniqid([string prefix , bool more_entropy])</code>	String	Генерирует уникальный идентификатор
<code>version_compare(string ver1, string ver2 [, string oper])</code>	Int	Сравнивает две стандартизированные строки, содержащие номера версий PHP
<code>connection_aborted(void)</code>	Int	Возвращает true, если клиент отсоединен
<code>connection_status(void)</code>	Int	Возвращает битовое поле состояния соединения

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>ignore_user_abort(bool value)</code>	Int	Устанавливает, должно ли отсоединение клиента прерывать выполнение сценария
<code>define(string constant_name, mixed value, case_sensitive=true)</code>	Bool	Определяет новую константу
<code>defined(string constant_name)</code>	Bool	Проверяет существование заданной константы

Функции MS SQL

Функция	Тип возвращаемых данных	Описание
<code>mssql_connect([string servername [, string username [, string password]])</code>	Int	Открывает соединение с MS SQL-сервером
<code>mssql_pconnect([string servername [, string username [, string password]])</code>	Int	Открывает постоянное MS SQL-соединение
<code>mssql_close([resource conn_id])</code>	Bool	Закрывает соединение с MS SQL-сервером
<code>mssql_select_db(string database_name [, resource conn_id])</code>	Bool	Выбирает базу данных MS-SQL
<code>mssql_fetch_batch(resource result_index)</code>	Int	Возвращает следующий пакет записей
<code>mssql_query(string query [, resource conn_id [, int batch_size]])</code>	resource	Отправляет SQL-запрос к MS-SQL-базе данных
<code>mssql_rows_affected(resource conn_id)</code>	Int	Возвращает количество записей, затронутых запросом
<code>mssql_free_result(resource result_index)</code>	Bool	Освобождает память, занятую результатом MS-SQL-запроса
<code>mssql_get_last_message(void)</code>	String	Возвращает последнее сообщение с сервера
<code>mssql_num_rows(resource mssql_result_index)</code>	Int	Получает количество строк в результате, заданном с помощью идентификатора
<code>mssql_num_fields(resource mssql_result_index)</code>	Int	Получает количество полей в результате, заданном с помощью идентификатора
<code>mssql_fetch_row(resource result_id)</code>	Array	Возвращает массив текущей строки результирующего множества, заданного параметром <code>result_id</code>

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>mssql_fetch_object(resource result_id [, int result_type])</code>	Object	Возвращает псевдо-объект текущей строки результирующего множества, заданного с помощью параметра <code>result_id</code>
<code>mssql_fetch_array(resource result_id [, int result_type])</code>	Array	Возвращает текущую строку результирующего множества, заданного <code>result_id</code> , в виде массива
<code>mssql_fetch_assoc(resource result_id)</code>	Array	Возвращает текущую строку результирующего множества, заданного <code>result_id</code> , в виде ассоциативного массива
<code>mssql_data_seek(resource result_id, int offset)</code>	Bool	Перемещает внутренний указатель строки в результирующем множестве, связанном с заданным идентификатором, к указанной строке
<code>mssql_fetch_field(resource result_id [, int offset])</code>	Object	Возвращает информацию об определенных полях в результате запроса
<code>mssql_field_length(resource result_id [, int offset])</code>	Int	Возвращает длину MS-SQL-поля
<code>mssql_field_name(resource result_id [, int offset])</code>	String	Возвращает имя поля, заданного смещением <code>offset</code> в результирующем множестве <code>result_id</code>
<code>mssql_field_type(resource result_id [, int offset])</code>	String	Возвращает тип поля
<code>mssql_field_seek(int result_id, int offset)</code>	Bool	Ищет поле с заданным смещением
<code>mssql_result(resource result_id, int row, mixed field)</code>	String	Возвращает содержимое одной ячейки из результирующего множества MS-SQL
<code>mssql_next_result(resource result_id)</code>	Bool	Перемещает внутренний указатель к следующему результату
<code>mssql_min_error_severity(int severity)</code>	Void	Устанавливает минимальный уровень серьезности ошибок
<code>mssql_min_message_severity(int severity)</code>	Void	Устанавливает минимальный уровень серьезности сообщений
<code>mssql_init(string sp_name [, resource conn_id])</code>	Int	Инициализирует хранимую процедуру или удаленную хранимую процедуру
<code>mssql_bind(resource stmt, string param_name, mixed var, int type [, int is_output[, int is_null[, int maxlen]])</code>	Bool	Добавляет параметр в хранимую процедуру или удаленную хранимую процедуру
<code>mssql_execute(resource stmt [, bool skip_results = false])</code>	Mixed	Выполняет хранимую процедуру над базой данных MS-SQL Server
<code>mssql_free_statement(resource result_index)</code>	Bool	Освобождает индекс MS-SQL-оператора
<code>mssql_guid_string(string binary [, int short_format])</code>	String	Конвертирует 16-байтовый двоичный GUID-идентификатор в строку

Функции MySQL

Функция	Тип возвращаемых данных	Описание
<code>mysql_connect([string hostname [:port] [:path/to/socket] [, string username [, string password [, bool new [, int flags]]]])</code>	resource	Открывает соединение с MySQL-сервером
<code>mysql_pconnect([string hostname [:port] [:path/to/socket] [, string username [, string password [, int flags]]])</code>	resource	Открывает постоянное соединение с MySQL-сервером
<code>mysql_close([int link_identifier])</code>	Bool	Закрывает MySQL-соединение
<code>mysql_select_db(string database_name [, int link_identifier])</code>	Bool	Выбирает базу данных MySQL
<code>mysql_get_client_info(void)</code>	String	Возвращает строку, представляющую описание версии клиентской библиотеки
<code>mysql_get_host_info([int link_identifier])</code>	String	Возвращает строку, описывающую тип используемого соединения, включая имя машины-сервера
<code>mysql_get_proto_info([int link_identifier])</code>	Int	Возвращает версию используемого в текущем соединении протокола
<code>mysql_get_server_info([int link_identifier])</code>	String	Возвращает номер версии MySQL-сервера
<code>mysql_info([int link_identifier])</code>	String	Возвращает информацию о самом последнем запросе
<code>mysql_thread_id([int link_identifier])</code>	Int	Возвращает идентификатор текущего потока
<code>mysql_stat([int link_identifier])</code>	String	Возвращает строку с информацией о состоянии системы
<code>mysql_client_encoding([int link_identifier])</code>	String	Возвращает кодировку символов по умолчанию для текущего соединения
<code>mysql_create_db(string database_name [, int link_identifier])</code>	Bool	Создает базу данных MySQL
<code>mysql_drop_db(string database_name [, int link_identifier])</code>	Bool	Удаляет базу данных MySQL
<code>mysql_query(string query [, int link_identifier])</code>	resource	Отправляет MySQL-запрос
<code>mysql_unbuffered_query(string query [, int link_identifier])</code>	resource	Отправляет SQL-запрос без автообработки результата и буферизации
<code>mysql_db_query(string database_name, string query [, int link_identifier])</code>	resource	Отправляет MySQL-запрос

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>mysql_list_dbs([int link_identifier])</code>	resource	Возвращает список баз данных, доступных на MySQL-сервере
<code>mysql_list_tables(string database_name [, int link_identifier])</code>	resource	Возвращает список таблиц в базе данных MySQL
<code>mysql_list_fields(string database_name, string table_name [, int link_identifier])</code>	resource	Возвращает список полей MySQL-результата
<code>mysql_list_processes([int link_identifier])</code>	resource	Возвращает результирующее множество, описывающее текущие процессы сервера
<code>mysql_error([int link_identifier])</code>	String	Возвращает текст сообщения об ошибке для предыдущей MySQL-операции
<code>mysql_errno([int link_identifier])</code>	Int	Возвращает номер сообщения об ошибке для предыдущей MySQL-операции
<code>mysql_affected_rows([int link_identifier])</code>	Int	Возвращает количество строк, затронутых в предыдущей MySQL-операции
<code>mysql_escape_string(string to_be_escaped)</code>	String	Экранирует строку для MySQL-запроса
<code>mysql_real_escape_string(string to_be_escaped [, int link_identifier])</code>	String	Экранирует специальные символы в строке для использования в SQL-операторе, учитывая текущую кодировку символов в соединении
<code>mysql_insert_id([int link_identifier])</code>	Int	Возвращает идентификатор, сгенерированный в результате предыдущей INSERT-операции
<code>mysql_result(resource result, int row [, mixed field])</code>	Mixed	Возвращает результирующие данные
<code>mysql_num_rows(resource result)</code>	Int	Возвращает количество строк в результате
<code>mysql_num_fields(resource result)</code>	Int	Возвращает количество полей в результате
<code>mysql_fetch_row(resource result)</code>	Array	Возвращает результирующую строку в виде массива
<code>mysql_fetch_object(resource result [, int result_type])</code>	Object	Возвращает результирующую строку как объект
<code>mysql_fetch_array(resource result [, int result_type])</code>	Array	Возвращает результирующую строку в виде массива, (ассоциативного, числового или и того и другого)
<code>mysql_fetch_assoc(resource result)</code>	Array	Возвращает результирующую строку в виде ассоциативного массива

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>mysql_data_seek(resource result, int row_number)</code>	Bool	Перемещает внутренний указатель результата
<code>mysql_fetch_lengths(resource result)</code>	Array	Возвращает максимальный размер данных для каждого поля результата
<code>mysql_fetch_field(resource result [, int field_offset])</code>	Object	Получает информацию о столбцах результата и возвращает ее в виде объекта
<code>mysql_field_seek(resource result, int field_offset)</code>	Bool	Устанавливает указатель на поле с заданным смещением
<code>mysql_field_name(resource result, int field_index)</code>	String	Возвращает имя заданного поля результата
<code>mysql_field_table(resource result, int field_offset)</code>	String	Возвращает имя таблицы, в которой находится указанное поле
<code>mysql_field_len(resource result, int field_offset)</code>	Int	Возвращает длину заданного поля
<code>mysql_field_type(resource result, int field_offset)</code>	String	Возвращает тип заданного поля результата
<code>mysql_field_flags(resource result, int field_offset)</code>	String	Возвращает флаги указанного поля результата
<code>mysql_free_result(resource result)</code>	Bool	Освобождает память, занятую результатом
<code>mysql_ping([int link_identifier])</code>	Bool	Проверяет соединение с сервером; если соединение разорвано, то устанавливает его заново

Сетевые функции

Функция	Тип возвращаемых данных	Описание
<code>define_syslog_variables (void)</code>	Void	Инициализирует все переменные, относящиеся к syslog
<code>openlog(string ident, int option, int facility)</code>	Bool	Открывает соединение с программой системного протоколирования
<code>closelog(void)</code>	Bool	Закрывает соединение с программой системного протоколирования
<code>syslog(int priority, string message)</code>	Bool	Генерирует сообщение для системного журнала
<code>ip2long(string ip_address)</code>	Int	Конвертирует строку, содержащую адрес (IPV4), в сетевой адрес
<code>long2ip(int proper_address)</code>	String	Конвертирует сетевой адрес (IPV4) в строку стандартного Internet-формата с точкой

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>getservbyname(string service, string protocol)</code>	Int	Возвращает номер порта, связанного со службой; протокол должен быть TCP или UDP
<code>getservbyport(int port, string protocol)</code>	String	Возвращает имя службы, связанной с портом; протокол должен быть TCP или UDP
<code>getprotobyname(string name)</code>	Int	Возвращает номер протокола, связанного с именем в файле
<code>getprotobyname(int proto)</code>	String	Возвращает имя протокола, связанное с номером протокола

ODBC-функции

Функция	Тип возвращаемых данных	Описание
<code>odbc_close_all(void)</code>	Void	Закрывает все ODBC-соединения
<code>odbc_binmode(int result_id, int mode)</code>	Bool	Обрабатывает данные бинарного столбца
<code>odbc_longreadlen(int result_id, int length)</code>	Bool	Обрабатывает LONG-столбцы
<code>odbc_prepare(resource connection_id, string query)</code>	resource	Готовит оператор для выполнения
<code>odbc_execute(resource result_id [, array parameters_array])</code>	Bool	Выполняет подготовленный оператор
<code>odbc_cursor(resource result_id)</code>	String	Получает имя курсора
<code>odbc_data_source(resource connection_id, int fetch_type)</code>	Array	Возвращает информацию о подсоединенном в настоящий момент источнике данных
<code>odbc_exec(resource connection_id, string query [, int flags])</code>	resource	Готовит и выполняет SQL-оператор
<code>odbc_fetch_object(int result [, int rownumber])</code>	Object	Возвращает результирующую строку в виде объекта
<code>odbc_fetch_array(int result [, int rownumber])</code>	Array	Возвращает результирующую строку в виде ассоциативного массива
<code>odbc_fetch_into(resource result_id, array result_array [, int rownumber])</code>	Int	Извлекает одну результирующую строку в массив
<code>odbc_fetch_row(resource result_id [, int row_number])</code>	Bool	Извлекает строку
<code>odbc_result(resource result_id, mixed field)</code>	Mixed	Получает результирующие данные
<code>odbc_result_all(resource result_id [, string format])</code>	Int	Печатает результат как HTML-таблицу

Функция	Тип возвращаемых данных	Описание
<code>odbc_free_result(resource result_id)</code>	Bool	Освобождает ресурсы, связанные с результатом
<code>odbc_connect(string DSN, string user, string password [, int cursor_option])</code>	resource	Подключается к источнику данных
<code>odbc_pconnect(string DSN, string user, string password [, int cursor_option])</code>	resource	Устанавливает постоянное соединение с источником данных
<code>odbc_close(resource connection_id)</code>	Void	Закрывает ODBC-соединение
<code>odbc_num_rows(resource result_id)</code>	Int	Возвращает количество строк в результате
<code>odbc_next_result(resource result_id)</code>	Bool	Проверяет, доступны ли множественные результаты
<code>odbc_num_fields(resource result_id)</code>	Int	Возвращает количество столбцов в результате
<code>odbc_field_name(resource result_id, int field_number)</code>	String	Возвращает имя столбца
<code>odbc_field_type(resource result_id, int field_number)</code>	String	Возвращает тип данных поля
<code>odbc_field_len(resource result_id, int field_number)</code>	Int	Возвращает размер (точность) поля
<code>odbc_field_scale(resource result_id, int field_number)</code>	Int	Возвращает точность чисел в поле
<code>odbc_field_num(resource result_id, string field_name)</code>	Int	Возвращает номер столбца
<code>odbc_autocommit(resource connection_id [, int OnOff])</code>	Mixed	Включает/выключает режим автоподтверждения или возвращает состояние автоподтверждения
<code>odbc_commit(resource connection_id)</code>	Bool	Подтверждает ODBC-транзакцию
<code>odbc_rollback(resource connection_id)</code>	Bool	Аннулирует транзакцию
<code>odbc_error([resource connection_id])</code>	String	Возвращает код последней ошибки
<code>odbc_errormsg([resource connection_id])</code>	String	Возвращает последнее сообщение об ошибке
<code>odbc_setoption(resource conn_id result_id, int which, int option, int value)</code>	Bool	Настраивает соединение или параметры оператора
<code>odbc_tables(resource connection_id [, string qualifier, string owner, string name, string table_types])</code>	resource	Вызывает функцию <code>sqltables</code>

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>odbc_columns(resource connection_id, string qualifier, string owner, string table_name, string column_name)</code>	resource	Возвращает идентификатор результата, который можно использовать для получения списка имен столбцов в заданной таблице
<code>odbc_columnprivileges (resource connection_id, string catalog, string schema, string table, string column)</code>	resource	Возвращает идентификатор результата, который можно использовать для получения списка столбцов в заданной таблице и связанных с ними привилегий
<code>odbc_foreignkeys(resource connection_id, string pk_qualifier, string pk_owner, string pk_table, string fk_qualifier, string fk_owner, string fk_table)</code>	resource	Возвращает идентификатор либо для списка внешних ключей в заданной таблице, либо для списка внешних ключей в других таблицах, которые ссылаются на первичный ключ заданной таблицы
<code>odbc_gettypeinfo(resource connection_id [, int data_type])</code>	resource	Возвращает результирующий идентификатор, содержащий информацию о типах данных, поддерживаемых этим источником данных
<code>odbc_primarykeys(resource connection_id, string qualifier, string owner, string table)</code>	resource	Возвращает идентификатор результата, который может использоваться для извлечения имен столбцов, образующих первичный ключ таблицы
<code>odbc_procedurecolumns (resource connection_id [, string qualifier, string owner, string proc, string column])</code>	resource	Возвращает список параметров ввода и вывода, а также столбцов, образующих результирующее множество для заданных процедур
<code>odbc_procedures(resource connection_id [, string qualifier, string owner, string name])</code>	resource	Возвращает идентификатор результата, содержащий список процедур, хранимых в источнике данных
<code>odbc_specialcolumns(resource connection_id, int type, string qualifier, string owner, string table, int scope, int nullable)</code>	resource	Возвращает идентификатор результата, содержащий либо оптимальный набор столбцов, который уникально идентифицирует строку в таблице, либо столбцы, которые автоматически обновляются при обновлении транзакцией любого значения строки
<code>odbc_statistics(resource connection_id, string qualifier, string owner, string name, int unique, int accuracy)</code>	resource	Возвращает идентификатор результата, содержащий статистические данные об одной таблице и связанных с ней индексах
<code>odbc_tableprivileges (resource connection_id, string qualifier, string owner, string name)</code>	resource	Возвращает идентификатор результата, содержащий список таблиц и привилегий, связанных с каждой таблицей

Буферизация вывода

Функция	Тип возвращаемых данных	Описание
<code>ob_list_handlers()</code>	false array	Возвращает массив, в котором содержатся все используемые обработчики вывода
<code>ob_start([string array user_function [, int chunk_size [, bool erase]]])</code>	Bool	Включает буферизацию вывода (можно задать необязательный обработчик вывода)
<code>ob_flush(void)</code>	Bool	Сбрасывает (отправляет) буфер вывода; текущее содержимое буфера отправляется в следующий буфер
<code>ob_clean(void)</code>	Bool	Очищает (удаляет содержимое) буфер вывода
<code>ob_end_flush(void)</code>	Bool	Сбрасывает (отправляет) буфер вывода и удаляет текущий буфер вывода
<code>ob_end_clean(void)</code>	Bool	Очищает буфер вывода и удаляет текущий буфер вывода
<code>ob_get_flush(void)</code>	Bool	Получает содержимое текущего буфера, сбрасывает (отправляет) буфер вывода и удаляет текущий буфер вывода
<code>ob_get_clean(void)</code>	Bool	Получает текущее содержимое буфера и удаляет текущий буфер вывода
<code>ob_get_contents(void)</code>	String	Возвращает содержимое буфера вывода
<code>ob_get_level(void)</code>	Int	Возвращает уровень вложенности буферов вывода
<code>ob_get_length(void)</code>	Int	Возвращает размер буфера вывода
<code>ob_get_status([bool full_status])</code>	false array	Возвращает состояние активного или всех буферов вывода
<code>ob_implicit_flush([int flag])</code>	Void	Включает/выключает неявную очистку; эквивалентна вызову <code>flush()</code> после каждого внешнего вызова
<code>output_reset_rewrite_vars(void)</code>	Bool	Переустанавливает (очищает) значения перезаписи URL
<code>output_add_rewrite_var(string name, string value)</code>	Bool	Добавляет значения перезаписи URL

Функции PCRE

Функция	Тип возвращаемых данных	Описание
<code>preg_match(string pattern, string subject [, array subpatterns [, int flags [, int offset]]])</code>	Int	Выполняет проверку на соответствие регулярному выражению в Perl-стиле

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>preg_match_all(string pattern, string subject, array subpatterns [, int flags [, int offset]])</code>	Int	Выполняет глобальную проверку на соответствие регулярному выражению в Perl-стиле
<code>preg_replace(mixed regex, mixed replace, mixed subject [, int limit])</code>	String	Выполняет поиск и замену регулярного выражения
<code>preg_replace_callback (mixed regex, mixed callback, mixed subject [, int limit])</code>	String	Выполняет поиск и замену регулярного выражения с использованием функции обратного вызова
<code>preg_split(string pattern, string subject [, int limit [, int flags]])</code>	Array	Разбивает строку на элементы массива, используя регулярное выражение в Perl-стиле в качестве разделителя
<code>preg_quote(string str, string delim_char)</code>	String	Экранирует символы регулярного выражения плюс необязательный символ
<code>preg_grep(string regex, array input)</code>	Array	Возвращает массив вхождений, совпадающих с регулярным выражением

Параметры PHP и информация о PHP

Функция	Тип возвращаемых данных	Описание
<code>assert(string bool assertion)</code>	Int	Проверяет, не является ли утверждение ложным
<code>assert_options(int what [, mixed value])</code>	Mixed	Устанавливает/получает различные флаги утверждения
<code>phpinfo([int what])</code>	Void	Выводит страницу полезной информации о PHP и текущем запросе
<code>phpversion([string extension])</code>	String	Возвращает текущую версию PHP
<code>phpcredits([int flag])</code>	Void	Выводит список людей, которые внесли свой вклад в развитие проекта PHP
<code>php_logo_guid(void)</code>	String	Возвращает специальный идентификатор, который используется для запроса логотипа PHP на странице <code>phpinfo</code>
<code>php_real_logo_guid(void)</code>	String	Возвращает специальный идентификатор, который используется для запроса логотипа PHP на странице <code>phpinfo</code>
<code>php_egg_logo_guid(void)</code>	String	Возвращает специальный идентификатор, который используется для запроса логотипа PHP на странице <code>phpinfo</code>

Функция	Тип возвращаемых данных	Описание
<code>zend_logo_guid(void)</code>	String	Возвращает специальный идентификатор, который используется для запроса логотипа Zend на странице <code>phpinfo</code>
<code>php_sapi_name(void)</code>	String	Возвращает тип текущего SAPI-интерфейса
<code>php_uname(void)</code>	String	Возвращает информацию об операционной системе, на которой был собран PHP
<code>php_ini_scanned_files(void)</code>	String	Возвращает список разделенных запятыми <code>ini</code> -файлов, считанных из дополнительного каталога
<code>getmyuid(void)</code>	Int	Возвращает UID владельца PHP-сценария
<code>getmygid(void)</code>	Int	Возвращает GID владельца PHP-сценария
<code>getmypid(void)</code>	Int	Возвращает текущий идентификатор PHP-процесса
<code>getmyinode(void)</code>	Int	Возвращает индексный узел текущего сценария
<code>getlastmod(void)</code>	Int	Возвращает время последней модификации страницы
<code>set_time_limit(int seconds)</code>	Bool	Ограничивает время выполнения сценария
<code>ini_get(string varname)</code>	String	Возвращает значение параметра конфигурации
<code>ini_get_all([string extension])</code>	Array	Возвращает значения всех конфигурационных параметров
<code>ini_set(string varname, string newvalue)</code>	String	Устанавливает значение параметра конфигурации, возвращает прежнее значение параметра или <code>false</code> в случае ошибки
<code>ini_restore(string varname)</code>	Void	Восстанавливает значение параметра конфигурации, заданного в аргументе <code>varname</code>
<code>set_include_path(string varname, string newvalue)</code>	String	Устанавливает значение параметра <code>include_path</code>
<code>get_include_path()</code>	String	Возвращает значение параметра <code>include_path</code>
<code>restore_include_path()</code>	Void	Восстанавливает значение параметра <code>include_path</code>
<code>get_current_user(void)</code>	String	Возвращает имя владельца текущего PHP-сценария
<code>get_cfg_var(string option_name)</code>	String	Возвращает значение параметра конфигурации PHP
<code>set_magic_quotes_runtime(int new_setting)</code>	Bool	Устанавливает значение параметра <code>magic_quotes_runtime</code> , и возвращает предыдущее значение

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>get_magic_quotes_runtime(void)</code>	Int	Возвращает текущее значение параметра <code>magic_quotes_runtime</code>
<code>get_magic_quotes_gpc(void)</code>	Int	Возвращает текущее значение параметра <code>magic_quotes_gpc</code>
<code>get_declared_classes()</code>	Array	Возвращает массив имен всех объявленных классов
<code>get_declared_interfaces()</code>	Array	Возвращает массив имен всех объявленных интерфейсов
<code>get_defined_functions(void)</code>	Array	Возвращает массив имен всех объявленных функций
<code>get_defined_vars(void)</code>	Array	Возвращает массив имен и значений всех определенных переменных (переменных в текущей области видимости)
<code>get_resource_type(resource res)</code>	String	Возвращает тип заданного ресурса
<code>get_loaded_extensions(void)</code>	Array	Возвращает массив имен всех загруженных и расширений
<code>get_defined_constants(void)</code>	Array	Возвращает массив имен и значений всех определенных констант
<code>extension_loaded(string extension_name)</code>	Bool	Определяет, загружено ли заданное расширение
<code>get_extension_funcs(string extension_name)</code>	Array	Возвращает массив, содержащий имена функций заданного расширения
<code>get_included_files(void)</code>	Array	Возвращает массив с именами подключенных (с помощью функций <code>include</code> или <code>require</code>) файлов
<code>zend_version(void)</code>	String	Возвращает текущую версию Zend Engine

Выполнение программ

Функция	Тип возвращаемых данных	Описание
<code>exec(string command [, array &output [, int &return_value]])</code>	String	Выполняет внешнюю программу
<code>system(string command [, int &return_value])</code>	Int	Выполняет внешнюю программу и отображает вывод
<code>passthru(string command [, int &return_value])</code>	Void	Выполняет внешнюю программу и выводит необработанный вывод
<code>escapeshellcmd(string command)</code>	String	Экранирует метасимволы оболочки
<code>escapeshellarg(string arg)</code>	String	Экранирует аргумент для использования в shell-команде

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
shell_exec(string cmd)	String	Выполняет команду посредством shell и возвращает весь вывод как строку
proc_nice(int priority)	Bool	Изменяет приоритет текущего процесса
proc_terminate(resource process [, long signal])	Int	Уничтожает процесс, созданный функцией proc_open
proc_close(resource process)	Int	Закрывает процесс, созданный функцией proc_open
proc_get_status(resource process)	Array	Возвращает информацию о процессе, созданном с помощью функции proc_open
proc_open(string command, array descriptorspec, array &pipes [, string cwd [, array env [, array other_options]]])	Resource	Запускает процесс и предоставляет большие возможности управления им посредством файловых дескрипторов

Регулярные выражения

Функция	Тип возвращаемых данных	Описание
ereg(string pattern, string string [, array registers])	Int	Ищет совпадение с регулярным выражением
eregi(string pattern, string string [, array registers])	Int	Ищет совпадение с регулярным выражением (без учета регистра символов)
ereg_replace(string pattern, string replacement, string string)	String	Заменяет подстроку, совпадающую с регулярным выражением
eregi_replace(string pattern, string replacement, string string)	String	Заменяет подстроку, совпадающую с регулярным выражением (без учета регистра символов)
split(string pattern, string string [, int limit])	Array	Разбивает строку на элементы массива с помощью регулярного выражения
spliti(string pattern, string string [, int limit])	Array	Разбивает строку на элементы массива с помощью регулярного выражения (без учета регистра символов)
sql_regcase(string string)	String	Создает регулярное выражение для поиска совпадений без учета регистра символов

Сеансы

Функция	Тип возвращаемых данных	Описание
<code>session_set_cookie_params</code> (int lifetime [, string path [, string domain [, bool secure]]])	Void	Устанавливает параметры сеансовых cookie-файлов
<code>session_get_cookie_params</code> (void)	Array	Возвращает параметры сеансовых cookie-файлов
<code>session_name</code> ([string newname])	String	Возвращает и/или устанавливает имя текущего сеанса; если задан параметр newname, то имя сеанса заменяется значением этого параметра
<code>session_module_name</code> ([string newname])	String	Возвращает имя текущего расширения, используемое для доступа к данным сеанса; если задан параметр newname, то имя расширения заменяется значением этого параметра
<code>session_set_save_handler</code> (string open, string close, string read, string write, string destroy, string gc)	Void	Устанавливает пользовательские функции
<code>session_save_path</code> ([string newname])	String	Возвращает путь к каталогу, в котором сохраняются данные сеанса; если задан параметр newname, то путь будет изменен
<code>session_id</code> ([string newid])	String	Возвращает и/или устанавливает идентификатор текущего сеанса; если задан новый идентификатор (newid), то идентификатор сеанса будет изменен
<code>session_regenerate_id</code> ()	Bool	Заменяет текущий идентификатор сеанса только что сгенерированным идентификатором
<code>session_cache_limiter</code> ([string new_cache_limiter])	String	Возвращает и/или устанавливает текущий ограничитель кэша; если задан параметр new_cache_limiter, то имя текущего ограничителя кэша заменяется новым именем
<code>session_cache_expire</code> ([int new_cache_expire])	Int	Возвращает дату окончания действия текущего кэша; если задан параметр new_cache_expire, то дата окончания действия текущего кэша заменяется новой датой
<code>session_register</code> (mixed var_names [, mixed. . .])	Bool	Добавляет имена переменных к списку переменных, которые будут зарегистрированы в конце сеанса
<code>session_unregister</code> (string varname)	Bool	Удаляет имя переменной из списка переменных, которые будут зарегистрированы в конце сеанса
<code>session_is_registered</code> (string varname)	Bool	Проверяет существование переменной сеанса

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>session_encode(void)</code>	String	Сериализует данные текущего сеанса и возвращает сериализованное представление этих данных
<code>session_decode(string data)</code>	Bool	Десериализует данные сеанса из строки и заново инициализирует переменные
<code>session_start(void)</code>	Bool	Начинает сеанс — заново инициализирует переменные, регистрирует браузер и т.д.
<code>session_destroy(void)</code>	Bool	Уничтожает текущий сеанс и все данные, связанные с ним
<code>session_unset(void)</code>	Void	Удаляет все зарегистрированные переменные сеанса
<code>session_write_close(void)</code>	Void	Записывает данные сеанса и закрывает сеанс

Simple XML

Функция	Тип возвращаемых данных	Описание
<code>simplexml_load_file(string filename)</code>	<code>simplexml_element</code>	Интерпретирует заданный файл как объект <code>simplexml_element</code> для дальнейшей обработки
<code>simplexml_load_string(string data)</code>	<code>simplexml_element</code>	Интерпретирует строку как <code>simplexml_element</code> для дальнейшей обработки
<code>simplexml_import_dom(domNode node)</code>	<code>simplexml_element</code>	Получает <code>dom simplexml_element</code> -объект для дальнейшей обработки

Сокеты

Функция	Тип возвращаемых данных	Описание
<code>socket_select(array &read_fds, array &write_fds, &array except_fds, int tv_sec[, int tv_usec])</code>	Int	Запускает системный вызов <code>select()</code> на заданных массивах сокетов с таймаутом, заданным параметрами <code>tv_sec</code> и <code>tv_usec</code>
<code>socket_create_listen(int port[, int backlog])</code>	Resource	Открывает сокет на порте для приема соединений
<code>socket_accept(resource socket)</code>	Resource	Принимает соединение с сокетом
<code>socket_set_nonblock(resource socket)</code>	Bool	Устанавливает неблокирующий режим для сокета

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>socket_set_block(resource socket)</code>	Bool	Устанавливает блокирующий режим для сокета
<code>socket_listen(resource socket[, int backlog])</code>	Bool	Задаёт максимальное количество соединений, которые может ожидать заданный сокет
<code>socket_close(resource socket)</code>	Void	Закрывает ресурс сокета
<code>socket_write(resource socket, string buf[, int length])</code>	Int	Записывает содержимое буфера в сокет; параметр length не обязательный
<code>socket_read(resource socket, int length [, int type])</code>	String	Читает максимум length байтов из сокета
<code>socket_getsockname(resource socket, string &addr[, int &port])</code>	Bool	Запрашивает локальную сторону данного сокета, что может дать либо пару узел/порт, либо путь файловой системы UNIX, в зависимости от типа
<code>socket_getpeername(resource socket, string &addr[, int &port])</code>	Bool	Запрашивает удалённую сторону данного сокета, что может дать либо пару узел/порт, либо путь файловой системы UNIX, в зависимости от типа
<code>socket_create(int domain, int type, int protocol)</code>	Resource	Создаёт конечную точку соединения в заданном домене (domain) и с заданным типом (type)
<code>socket_connect(resource socket, string addr [, int port])</code>	Bool	Открывает соединение ресурсом addr:port на заданном сокете (socket)
<code>socket_strerror(int errno)</code>	String	Возвращает строку с описанием ошибки сокета
<code>socket_bind(resource socket, string addr [, int port])</code>	Bool	Связывает открытый сокет с прослушивающим портом; порт указывается только для AF_INET-сокеты
<code>socket_recv(resource socket, string &buf, int len, int flags)</code>	Int	Получает данные из подключённого сокета
<code>socket_send(resource socket, string buf, int len, int flags)</code>	Int	Отправляет данные в подключённый сокет
<code>socket_recvfrom(resource socket, string &buf, int len, int flags, string &name [, int &port])</code>	Int	Получает данные из подключённого или не подключённого сокета
<code>socket_sendto(resource socket, string buf, int len, int flags, string addr [, int port])</code>	Int	Отправляет сообщение в подключённый или не подключённый сокет
<code>socket_get_option(resource socket, int level, int optname)</code>	Mixed	Возвращает параметры сокета
<code>socket_set_option(resource socket, int level, int optname, int array optval)</code>	Bool	Устанавливает параметры сокета

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>socket_create_pair(int domain, int type, int protocol, array &fd)</code>	Bool	Создает пару одинаковых сокетов и сохраняет их в массиве <code>indistinguishable</code>
<code>socket_shutdown(resource socket[, int how])</code>	Bool	Выключает получение или отправку данных с сокета или и то, и другое
<code>socket_last_error([resource socket])</code>	Int	Возвращает последнюю ошибку сокета (либо для последнего использованного, либо для заданного сокета)
<code>socket_clear_error([resource socket])</code>	Void	Очищает ошибку в сокете или код последней ошибки
<code>fsockopen(string hostname, int port [, int errno [, string errstr [, float timeout]]])</code>	Int	Открывает соединение по сокету Internet или Unix-домена
<code>pfsockopen(string hostname, int port [, int errno [, string errstr [, float timeout]]])</code>	Int	Открывает постоянное соединение по сокету Internet или Unix-домена

Функции SQLite

Функция	Тип возвращаемых данных	Описание
<code>sqlite_popen(string filename [, int mode [, string &error_message]])</code>	Resource	Открывает или создает базу данных <code>sqlite</code> , делая подключение постоянным
<code>sqlite_open(string filename [, int mode [, string &error_message]])</code>	Resource	Открывает или создает базу данных <code>sqlite</code>
<code>sqlite_factory(string filename [, int mode [, string &error_message]])</code>	Object	Открывает или создает базу данных <code>sqlite</code> и возвращает объект для нее
<code>sqlite_busy_timeout(resource db, int ms)</code>	Void	Устанавливает время ожидания; если значение параметра <code>ms</code> ≤ 0 , то в случае, когда файл базы данных заблокирован другим процессом, большинство функций <code>sqlite</code> будут немедленно возвращать значение <code>SQLITE_BUSY</code>
<code>sqlite_close(resource db)</code>	Void	Закрывает открытую базу данных <code>sqlite</code>
<code>sqlite_unbuffered_query(string query, resource db [, int result_type])</code>	Resource	Выполняет запрос без буферизации результатов
<code>sqlite_query(string query, resource db [, int result_type])</code>	Resource	Выполняет запрос к заданной базе данных и возвращает дескриптор результата

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>sqlite_fetch_all(resource result [, int result_type [, bool decode_binary]])</code>	Array	Возвращает все строки результирующего множества в виде многомерного массива
<code>sqlite_fetch_array(resource result [, int result_type [, bool decode_binary]])</code>	Array	Возвращает следующую строку результирующего множества в виде массива
<code>sqlite_fetch_object(resource result [, string class_name [, NULL array ctor_params [, bool decode_binary]])</code>	Object	Возвращает следующую строку результирующего множества в виде объекта
<code>sqlite_array_query(resource db, string query [, int result_type [, bool decode_binary]])</code>	Array	Выполняет запрос к заданной базе данных и возвращает массив массивов
<code>sqlite_single_query(resource db, string query [, bool first_row_only [, bool decode_binary]])</code>	Array	Выполняет запрос и возвращает либо массив для одного столбца либо значение его первой строки
<code>sqlite_fetch_single(resource result [, bool decode_binary])</code>	String	Возвращает первое поле результирующего множества в виде строки
<code>sqlite_current(resource result [, int result_type [, bool decode_binary]])</code>	Array	Возвращает текущую строку результирующего множества в виде массива
<code>sqlite_column(resource result, mixed index_or_name [, bool decode_binary])</code>	Mixed	Возвращает одно поле из текущей записи результирующего множества
<code>sqlite_libversion()</code>	String	Возвращает версию связанной sqlite-библиотеки
<code>sqlite_libencoding()</code>	String	Возвращает кодировку (iso8859 или UTF-8) связанной sqlite-библиотеки
<code>sqlite_changes(resource db)</code>	Int	Возвращает количество измененных последним SQL-оператором строк
<code>sqlite_last_insert_rowid(resource db)</code>	Int	Возвращает идентификатор последней вставленной строки
<code>sqlite_num_rows(resource result)</code>	Int	Возвращает количество строк в буферизированном результирующем множестве
<code>sqlite_has_more(resource result)</code>	Bool	Проверяет, существуют ли еще доступные записи
<code>sqlite_has_prev(resource result)</code>	Bool	Проверяет доступность предыдущей записи
<code>sqlite_num_fields(resource result)</code>	Int	Возвращает количество полей в результирующем множестве
<code>sqlite_field_name(resource result, int field_index)</code>	String	Возвращает имя определенного поля в результирующем множестве

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>sqlite_seek(resource result, int row)</code>	Bool	Переходит к определенному номеру записи в буферизированном результирующем множестве
<code>sqlite_rewind(resource result)</code>	Bool	Переходит к первой записи в буферизированном результирующем множестве
<code>sqlite_next(resource result)</code>	Bool	Переходит к следующей записи в буферизированном результирующем множестве
<code>sqlite_prev(resource result)</code>	Bool	Переходит к предыдущей записи результирующего множества
<code>sqlite_escape_string(string item)</code>	String	Экранирует строку для использования в качестве параметра запроса
<code>sqlite_last_error(resource db)</code>	Int	Возвращает код последней ошибки базы данных
<code>sqlite_error_string(int error_code)</code>	String	Возвращает текстовое описание кода ошибки
<code>sqlite_create_aggregate(resource db, string funcname, mixed step_func, mixed finalize_func[, long num_args])</code>	Bool	Регистрирует функцию группировки для использования в запросах
<code>sqlite_create_function(resource db, string funcname, mixed callback[, long num_args])</code>	Bool	Регистрирует обычную функцию для использования в запросах
<code>sqlite_udf_encode_binary(string data)</code>	String	Кодирует бинарные данные, возвращаемые пользовательской функцией
<code>sqlite_udf_decode_binary(string data)</code>	String	Декодирует бинарные данные, переданные в пользовательскую функцию

Расширение Streams

Функция	Тип возвращаемых данных	Описание
<code>stream_socket_client(string remoteaddress[, long &errcode, string &errstring, double timeout, long flags, resource context])</code>	Resource	Открывает клиентское соединение
<code>stream_socket_server(string localaddress[, long &errcode, string &errstring, long flags, resource context])</code>	Resource	Создает серверный сокет, связанный с localaddress

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>stream_socket_accept(resource serverstream, [double timeout, string &peername])</code>	Resource	Принимает клиентское соединение от серверного сокета
<code>stream_socket_get_name(resource stream, bool want_peer)</code>	String	Возвращает имя локального или удаленного сокета
<code>stream_socket_sendto(resource stream, string data [, long flags [, string target_addr]])</code>	Long	Отправляет данные в поток сокета; если указывается параметр <code>target_addr</code> , то он должен быть задан в виде четырех чисел, разделенных точками (или в формате ipv6)
<code>stream_socket_recvfrom(resource stream, long amount [, long flags [, string &remote_addr]])</code>	String	Получает данные из потока сокета
<code>stream_get_contents(resource source [, long maxlen])</code>	Long	Читает все оставшиеся байты (до <code>maxlen</code> байтов) из потока и возвращает их в виде строки
<code>stream_copy_to_stream(resource source, resource dest [, long maxlen])</code>	Long	Считывает до <code>maxlen</code> байт из потока-отправителя и записывает их в поток-получатель
<code>stream_get_meta_data(resource fp)</code>	Resource	Извлекает заголовки или метаданные из заданного потока/файла
<code>stream_get_transports()</code>	Array	Возвращает список зарегистрированных протоколов для транспортировки данных сокета
<code>stream_get_wrappers()</code>	Array	Возвращает список зарегистрированных обработчиков потока
<code>stream_select(array &read_streams, array &write_streams, array &except_streams, int tv_sec[, int tv_usec])</code>	Int	Запускает системный вызов <code>select()</code> на заданных массивах потоков с таймаутом, заданным параметрами <code>tv_sec</code> и <code>tv_usec</code>
<code>stream_context_get_options(resource context resource stream)</code>	Array	Возвращает параметры потока/обработчика/контекста
<code>stream_context_set_option(resource context resource stream, string wrappername, string optionname, mixed value)</code>	Bool	Устанавливает параметр для обработчика
<code>stream_context_set_params(resource context resource stream, array options)</code>	Bool	Устанавливает параметры для файлового контекста
<code>stream_context_create([array options])</code>	Resource	Создает файловый контекст и устанавливает параметры (если они заданы)
<code>stream_filter_prepend(resource stream, string filtername[, int read write[, string filterparams]])</code>	Bool	Добавляет фильтр в начало списка фильтров потока

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>stream_filter_append(resource stream, string filtername[, int read_write[, string filterparams]])</code>	Bool	Добавляет фильтр в конец списка фильтров потока
<code>stream_get_line(resource stream, int maxlen, string ending)</code>	String	Считывает до maxlen байт из потока или до символа конца строки
<code>stream_set_blocking(resource socket, int mode)</code>	Bool	Устанавливает блокирующий/неблокирующий режим для сокета или потока
<code>set_socket_blocking(resource socket, int mode)</code>	Bool	Устанавливает блокирующий/неблокирующий режим для сокета
<code>stream_set_timeout(resource stream, int seconds, int microseconds)</code>	Bool	Устанавливает время ожидания чтения потока (seconds + microseconds)
<code>stream_set_write_buffer(resource fp, int buffer)</code>	Int	Устанавливает буфер записи в файл
<code>stream_wrapper_register(string protocol, string classname)</code>	Bool	Регистрирует нестандартный класс-обработчик протокола URL
<code>stream_bucket_make_writeable(resource brigade)</code>	Object	Возвращает из группы (brigade) объект ячейки (bucket) для дальнейшего использования
<code>stream_bucket_prepend(resource brigade, resource bucket)</code>	Void	Добавляет ячейку в начало группы
<code>stream_bucket_append(resource brigade, resource bucket)</code>	Void	Добавляет ячейку в конец группы
<code>stream_bucket_new(resource stream, string buffer)</code>	Resource	Создает новую ячейку для использования в текущем потоке
<code>stream_get_filters(void)</code>	Array	Возвращает список зарегистрированных фильтров
<code>stream_filter_register(string filtername, string classname)</code>	Bool	Регистрирует пользовательский класс обработки фильтра

Строки

Функция	Тип возвращаемых данных	Описание
<code>crc32(string str)</code>	String	Вычисляет crc32-полином строки
<code>crypt(string str [, string salt])</code>	String	Шифрует строку
<code>convert_cyr_string(string str, string from, string to)</code>	String	Преобразует строку из одной кириллической кодировки в другую

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>lcg_value()</code>	Float	Возвращает значение из комбинированного линейного конгруэнтного генератора
<code>levenshtein(string str1, string str2)</code>	Int	Вычисляет расстояние Левенштейна между двумя строками
<code>md5(string str, [bool raw_output])</code>	String	Вычисляет md5-хеш строки
<code>md5_file(string filename [, bool raw_output])</code>	String	Вычисляет md5-хеш заданного файла
<code>metaphone(string text, int phones)</code>	String	Разбивает английские фразы на фонемы
<code>pack(string format, mixed arg1 [, mixed arg2 [, mixed\$. . . \$]])</code>	String	Упаковывает данные в бинарную строку, согласно аргументу format
<code>unpack(string format, string input)</code>	Array	Распаковывает данные из бинарной строки в именованный массив согласно аргументу format
<code>sha1(string str [, bool raw_output])</code>	String	Вычисляет sha1-хеш строки
<code>sha1_file(string filename [, bool raw_output])</code>	String	Вычисляет sha1-хеш заданного файла
<code>soundex(string str)</code>	String	Вычисляет soundex-ключ строки
<code>bin2hex(string data)</code>	String	Конвертирует двоичные данные в шестнадцатеричное представление
<code>strspn(string str, string mask [, start [, len]])</code>	Int	Возвращает длину участка в начале строки, соответствующего маске; если задан параметр start и/или len, то функция работает как <code>strspn(substr(\$s, \$start,\$len), \$good_chars)</code>
<code>strcspn(string str, string mask [, start [, len]])</code>	Int	Возвращает длину участка в начале строки, не соответствующего маске; если задан параметр start и/или len, то функция работает как <code>strcspn(substr(\$s, \$start,\$len)\$bad_chars)</code>
<code>nl_langinfo(int item)</code>	String	Возвращает информацию о языке и локализации
<code>strcoll(string str1, string str2)</code>	Int	Сравнение строк на основе текущих настроек локализации
<code>trim(string str [, string character_mask])</code>	String	Удаляет пробельные символы в начале и в конце строки
<code>rtrim(string str [, string character_mask])</code>	String	Удаляет пробельные символы в конце строки
<code>ltrim(string str [, string character_mask])</code>	String	Удаляет пробельные символы в начале строки

Функция	Тип возвращаемых данных	Описание
<code>wordwrap(string str [, int width [, string break [, boolean cut]]])</code>	String	Делает перенос строки на данное количество символов с использованием символа разрыва строки
<code>explode(string separator, string str [, int limit])</code>	Array	Делит строку по строке-разделителю (separator) и возвращает массив компонентов
<code>join(array src, string glue)</code>	String	Псевдоним функции <code>implode</code>
<code>implode([string glue,] array pieces)</code>	String	Объединяет элементы массива, помещая между ними строку glue, и возвращает одну строку
<code>strtok([string str,] string token)</code>	String	Разбивает строку на лексемы
<code>strtoupper(string str)</code>	String	Переводит символы строки в верхний регистр
<code>strtolower(string str)</code>	String	Переводит символы строки в нижний регистр
<code>basename(string path [, string suffix])</code>	String	Возвращает имя файла для заданного пути
<code>dirname(string path)</code>	String	Возвращает имя каталога для заданного пути
<code>pathinfo(string path)</code>	Array	Возвращает информацию о пути к файлу
<code>stristr(string haystack, string needle)</code>	String	Возвращает первое вхождение одной строки в другую без учета регистра символов
<code>strstr(string haystack, string needle)</code>	String	Возвращает первое вхождение одной строки в другую
<code>strchr(string haystack, string needle)</code>	String	Псевдоним функции <code>strstr</code>
<code>strpos(string haystack, string needle [, int offset])</code>	Int	Находит позицию первого вхождения одной строки в другую
<code>stripos(string haystack, string needle [, int offset])</code>	Int	Находит позицию первого вхождения одной строки в другую без учета регистра символов
<code>strrpos(string haystack, string needle [, int offset])</code>	Int	Находит позицию последнего вхождения одной строки в другую
<code>strripos(string haystack, string needle [, int offset])</code>	Int	Находит позицию последнего вхождения одной строки в другую без учета регистра символов
<code>strrchr(string haystack, string needle)</code>	String	Находит последнее вхождение символа в строке
<code>chunk_split(string str [, int chunklen [, string ending]])</code>	String	Делит строку на небольшие фрагменты
<code>substr(string str, int start [, int length])</code>	String	Возвращает часть строки

Продолжение таблицы

Функция	Тип возвращаемых данных	Описание
<code>substr_replace(mixed str, mixed repl, mixed start [, mixed length])</code>	Mixed	Заменяет текст части строки
<code>quoted_printable_decode(string str)</code>	String	Раскодирует строку, закодированную методом <code>quoted-printable</code> , в 8-битовую строку
<code>quotemeta(string str)</code>	String	Экранирует специальные символы
<code>ord(string character)</code>	Int	Возвращает ASCII-значение символа
<code>chr(int ascii)</code>	String	Конвертирует ASCII-код в символ
<code>ucfirst(string str)</code>	String	Переводит первый символ строки в верхний регистр
<code>ucwords(string str)</code>	String	Переводит первый символ каждого слова строки в верхний регистр
<code>strtr(string str, string from, string to)</code>	String	Заменяет определенные символы строки <code>str</code> , используя заданную таблицу преобразования
<code>strrev(string str)</code>	String	Возвращает строку <code>str</code> , в которой порядок символов изменен на обратный
<code>similar_text(string str1, string str2 [, float percent])</code>	Int	Вычисляет сходство между двумя строками
<code>addslashes(string str, string charlist)</code>	String	Экранирует все символы, заданные в <code>charlist</code> , символом обратной косой черты; создает восьмиричное представление символов "\" с 8-битным набором или с ASCII < 32 (кроме '\n', '\r', '\t' и т.д.)
<code>addslashes(string str)</code>	String	Экранирует одинарные кавычки, двойные кавычки и символы "\" символами "\"
<code>stripslashes(string str)</code>	String	Удаляет экранирование в стиле C
<code>stripslashes(string str)</code>	String	Удаляет из строки символы обратной косой черты
<code>str_replace(mixed search, mixed replace, mixed subject [, int replace_count])</code>	Mixed	Замещает все вхождения строки (<code>search</code>) строкой замещения (<code>replace</code>)
<code>str_ireplace(mixed search, mixed replace, mixed subject [, int replace_count])</code>	Mixed	Замещает все вхождения строки (<code>search</code>) строкой замещения (<code>replace</code>) без учета регистра символов
<code>hebrew(string str [, int max_chars_per_line])</code>	String	Преобразует текст на иврите из логической кодировки в визуальную
<code>hebrevc(string str [, int max_chars_per_line])</code>	String	Преобразует текст на иврите из логической кодировки в визуальную с преобразованием перевода строки
<code>nl2br(string str)</code>	String	Конвертирует символы перевода строки в HTML-теги разрыва строки

Функция	Тип возвращаемых данных	Описание
<code>strip_tags(string str [, string allowable_tags])</code>	String	Вырезает из строки HTML- и PHP-теги
<code>setlocale(mixed category, string locale [, string . . .])</code>	String	Устанавливает параметры локализации
<code>parse_str(string encoded_string [, array result])</code>	Void	Разбирает строку на глобальные переменные
<code>str_repeat(string input, int mult)</code>	String	Повторяет строку заданное количество раз
<code>count_chars(string input [, int mode])</code>	Mixed	Возвращает информацию о символах, используемых в строке
<code>strnatcmp(string s1, string s2)</code>	Int	Возвращает результат сравнения строк с использованием "естественного" алгоритма
<code>localeconv(void)</code>	Array	Возвращает информацию о форматировании чисел на основе настроек локализации
<code>strnatcasecmp(string s1, string s2)</code>	Int	Возвращает результат регистро-независимого сравнения строк с использованием "естественного" алгоритма
<code>substr_count(string haystack, string needle)</code>	Int	Вычисляет количество вхождений подстроки в строку
<code>str_pad(string input, int pad_length [, string pad_string [, int pad_type]])</code>	String	Заполняет строку до определенной длины другой строкой(<code>pad_string</code>)
<code>sscanf(string str, string format [, string . . .])</code>	Mixed	Разбирает строку в соответствии с заданным форматом
<code>str_rot13(string str)</code>	String	Выполняет rot13-преобразование строки
<code>str_shuffle(string str)</code>	Void	Переставляет символы в строке; выбирается одна перестановка из всех возможных
<code>str_word_count(string str, [int format])</code>	Mixed	Подсчитывает количество слов в строке <code>str</code> ; если значение необязательного аргумента <code>format</code> равно 1, то функция возвращает массив, содержащий все слова в строке; если <code>format</code> равен 2, то возвращается ассоциативный массив, в котором ключами являются числовые позиции слов
<code>money_format(string format, float value)</code>	String	Форматирует число как денежную величину
<code>str_split(string str [, int split_length])</code>	Array	Преобразует строку в массив; если указан необязательный аргумент <code>split_length</code> , то строка разбивается на фрагменты длиной <code>split_length</code> символов каждый
<code>strpbrk(string haystack, string char_list)</code>	Array	Ищет любой символ из списка <code>char_list</code> в строке <code>haystack</code>

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>substr_compare(string main_str, string str, int offset [, int length [, bool case_sensitivity]])</code>	Int	Выполняет безопасное для бинарных данных (возможно регистро-независимое) сравнение 2 строк от смещения offset до length символов
<code>uuencode(string data)</code>	String	Кодирует строку
<code>uudecode(string data)</code>	String	Декодирует строку
<code>sprintf(string format [, mixed arg1 [, mixed. . .]])</code>	String	Возвращает отформатированную строку
<code>vsprintf(string format, array args)</code>	String	Возвращает отформатированную строку
<code>printf(string format [, mixed arg1 [, mixed. . .]])</code>	Int	Выводит отформатированную строку
<code>vprintf(string format, array args)</code>	Int	Выводит отформатированную строку
<code>fprintf(resource stream, string format [, mixed arg1 [, mixed. . .]])</code>	Int	Выводит отформатированную строку в поток
<code>fprintf(resource stream, string format, array args)</code>	Int	Выводит отформатированную строку в поток
<code>htmlspecialchars(string string [, int quote_style] [, string charset])</code>	String	Конвертирует специальные символы в HTML-последовательности
<code>html_entity_decode(string string [, int quote_style] [, string charset])</code>	String	Преобразует HTML-последовательности в соответствующие символы
<code>htmlentities(string string [, int quote_style] [, string charset])</code>	String	Конвертирует все возможные символы в HTML-последовательности
<code>get_html_translation_table([int table [, int quote_style]])</code>	Array	Возвращает внутреннюю таблицу преобразования, используемую функциями <code>htmlspecialchars</code> и <code>htmlentities</code>
<code>strlen(string str)</code>	Int	Возвращает длину строки
<code>strcmp(string str1, string str2)</code>	Int	Выполняет безопасное для двоичных данных сравнение строк
<code>strncmp(string str1, string str2, int len)</code>	Int	Выполняет безопасное для двоичных данных сравнение строк
<code>strcasecmp(string str1, string str2)</code>	Int	Выполняет безопасное для двоичных данных регистро-независимое сравнение строк
<code>strncasecmp(string str1, string str2, int len)</code>	Int	Выполняет безопасное для двоичных данных сравнение строк

URL-функции

Функция	Тип возвращаемых данных	Описание
<code>http_build_query(mixed formdata [, string prefix])</code>	String	Генерирует URL-кодированную строку запроса из ассоциативного массива или объекта
<code>parse_url(string url)</code>	Array	Разбирает URL и возвращает его компоненты
<code>get_headers(string url)</code>	Array	Возвращает все заголовки, отправленные сервером в ответ на HTTP-запрос
<code>urlencode(string str)</code>	String	URL-кодирует все не алфавитно-цифровые символы, кроме <code>-_</code>
<code>urldecode(string str)</code>	String	Декодирует URL-кодированную строку
<code>rawurlencode(string str)</code>	String	URL-кодирует все не алфавитно-цифровые символы
<code>rawurldecode(string str)</code>	String	Декодирует URL-кодированные строки
<code>base64_encode(string str)</code>	String	Кодирует данные, используя алгоритм MIME base64
<code>base64_decode(string str)</code>	String	Декодирует данные, используя алгоритм MIME base64
<code>get_meta_tags(string filename [, bool use_include_path])</code>	Array	Извлекает из файла все атрибуты тегов meta и возвращает их в виде массива

Функции переменных

Функция	Тип возвращаемых данных	Описание
<code>gettype(mixed var)</code>	String	Возвращает тип переменной
<code>settype(mixed var, string type)</code>	Bool	Устанавливает тип переменной
<code>intval(mixed var [, int base])</code>	Int	Возвращает целочисленное значение переменной, используя не обязательное основание для преобразования
<code>floatval(mixed var)</code>	Float	Возвращает значение переменной — число с плавающей точкой
<code>strval(mixed var)</code>	String	Возвращает строковое значение переменной
<code>is_null(mixed var)</code>	Bool	Возвращает true, если переменная имеет значение NULL
<code>is_resource(mixed var)</code>	Bool	Определяет, является ли переменная ресурсом
<code>is_bool(mixed var)</code>	Bool	Определяет, является ли переменная булевым значением

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>is_long(mixed var)</code>	Bool	Определяет, является ли переменная целочисленной
<code>is_float(mixed var)</code>	Bool	Определяет, является ли переменная числом с плавающей точкой
<code>is_string(mixed var)</code>	Bool	Определяет, является ли переменная строкой
<code>is_array(mixed var)</code>	Bool	Определяет, является ли переменная массивом
<code>is_object(mixed var)</code>	Bool	Определяет, является ли переменная объектом
<code>is_numeric(mixed value)</code>	Bool	Возвращает true, если значение переменной является числом или числовой строкой
<code>is_scalar(mixed value)</code>	Bool	Возвращает true, если значение скалярно
<code>is_callable(mixed var [, bool syntax_only [, string callable_name]])</code>	Bool	Возвращает true, если переменная var является правильной вызываемой конструкцией
<code>var_dump(mixed var)</code>	Void	Выводит строковое представление переменной
<code>debug_zval_dump(mixed var)</code>	Void	Выводит строковое представление внутреннего Zend-значения
<code>var_export(mixed var [, bool return])</code>	Mixed	Выводит или возвращает строковое представление переменной
<code>serialize(mixed variable)</code>	String	Возвращает строковое представление переменной (которое затем можно десериализовать)
<code>unserialize(string variable_representation)</code>	Mixed	Принимает строковое представление переменной и восстанавливает ее
<code>memory_get_usage()</code>	Int	Возвращает объем памяти, выделенный для PHP
<code>print_r(mixed var [, bool return])</code>	Mixed	Возвращает информацию о заданной переменной
<code>import_request_variables (string types [, string prefix])</code>	Bool	Импортирует GET/POST/Cookie-переменные в глобальную область видимости

XML-функции

Функция	Тип возвращаемых данных	Описание
<code>xml_parser_create([string encoding])</code>	Resource	Создает XML-анализатор
<code>xml_parser_create_ns([string encoding [, string sep]])</code>	Resource	Создает XML-анализатор
<code>xml_set_object(resource parser, object &obj)</code>	Int	Устанавливает объект, который следует использовать для обратных вызовов
<code>xml_set_element_handler(resource parser, string shdl, string ehdl)</code>	Int	Настраивает обработчики начального и конечного элементов
<code>xml_set_character_data_handler(resource parser, string hdl)</code>	Int	Настраивает обработчик символьных данных
<code>xml_set_processing_instruction_handler(resource parser, string hdl)</code>	Int	Настраивает обработчик инструкций
<code>xml_set_default_handler(resource parser, string hdl)</code>	Int	Настраивает обработчик по умолчанию
<code>xml_set_unparsed_entity_decl_handler(resource parser, string hdl)</code>	Int	Настраивает обработчик объявления неразобранного экземпляра
<code>xml_set_notation_decl_handler(resource parser, string hdl)</code>	Int	Настраивает обработчик объявления нотации
<code>xml_set_external_entity_ref_handler(resource parser, string hdl)</code>	Int	Настраивает внешний обработчик экземпляров
<code>xml_set_start_namespace_decl_handler(resource parser, string hdl)</code>	Int	Настраивает обработчик символьных данных
<code>xml_set_end_namespace_decl_handler(resource parser, string hdl)</code>	Int	Настраивает обработчик символьных данных
<code>xml_parse(resource parser, string data [, int isFinal])</code>	Int	Начинает разбор XML-документа
<code>xml_parse_into_struct(resource parser, string data, array &struct, array &index)</code>	Int	Разбирает XML-документ
<code>xml_get_error_code(resource parser)</code>	Int	Возвращает код ошибки XML-анализатора
<code>xml_error_string(int code)</code>	String	Возвращает строку ошибки XML-анализатора
<code>xml_get_current_line_number(resource parser)</code>	Int	Возвращает номер текущей строки для XML-анализатора
<code>xml_get_current_column_number(resource parser)</code>	Int	Возвращает номер текущего столбца для XML-анализатора
<code>xml_get_current_byte_index(resource parser)</code>	Int	Возвращает индекс текущего байта для XML-анализатора

Окончание таблицы

Функция	Тип возвращаемых данных	Описание
<code>xml_parser_free(resource parser)</code>	Int	Освобождает XML-анализатор
<code>xml_parser_set_option(resource parser, int option, mixed value)</code>	Int	Устанавливает параметры XML-анализатора
<code>xml_parser_get_option(resource parser, int option)</code>	Int	Возвращает параметры XML-анализатора
<code>utf8_encode(string data)</code>	String	Кодирует ISO-8859-1-строку в UTF-8
<code>utf8_decode(string data)</code>	String	Конвертирует UTF-8-строку в ISO-8859-1-строку

ZLib

Функция	Тип возвращаемых данных	Описание
<code>gzfile(string filename [, int use_include_path])</code>	Array	Читает весь gz-файл в массив
<code>gzopen(string filename, string mode [, int use_include_path])</code>	Resource	Открывает gz-файл и возвращает указатель
<code>readgzfile(string filename [, int use_include_path])</code>	Int	Выводит gz-файл
<code>gzcompress(string data [, int level])</code>	String	Сжимает строку
<code>gzuncompress(string data [, int length])</code>	String	Распаковывает сжатую gzip строку
<code>gzdeflate(string data [, int level])</code>	String	Сжимает (gzip) строку
<code>gzinflate(string data [, int length])</code>	String	Распаковывает сжатую gzip строку
<code>zlib_get_coding_type(void)</code>	String	Возвращает тип кодирования, используемый для сжатия вывода
<code>gzencode(string data [, int level [, int encoding_mode]])</code>	String	Сжимает строку в формате gzip
<code>ob_gzhandler(string str, int mode)</code>	String	Кодирует строку str на основании настроек accept-encoding браузера; функция должна использоваться как функция обратного вызова из ob_start()



Использование SQLite

В дистрибутив пятой версии PHP включена библиотека SQLite. Это, по сути, дает пользователям завершенную, поддерживающую языки сценариев серверную систему управления реляционными базами данных в виде одного бесплатного и простого в использовании инструмента. Как использовать этот инструмент и использовать ли его вообще — решать вам. В этом приложении представлена краткая информация, необходимая для того, чтобы включить SQLite в Web-проект. Однако, прежде всего, это приложение предназначено для того, чтобы дать читателю навыки использования инструментов SQLite.

Что такое SQLite?

SQLite представляет собой базовую библиотеку, написанную на C, которая делает встраиваемую машину SQL-баз данных доступной использующим ее приложениям. Конечно, SQLite можно использовать и как автономную систему баз данных — здесь такой вариант рассматривается, — но основная ценность технологии SQLite заключается в том, что ее можно использовать в качестве сервера баз данных для более функционально развитого приложения, написанного на PHP. SQLite в отличие от MySQL не является клиент-серверной СУБД, это даже не часть такой системы. SQLite считывает и записывает данные в локально сохраняемые дисковые файлы, в которых содержатся создаваемые и используемые базы данных. Такое определение может показаться очень похожим на определение системы баз данных на основе простых файлов, таких как DBM, однако это не так. SQLite — вполне завершенная СУБД со многими функциями, сопутствующими этому понятию, которая также содержит реальную реализацию SQL, основанную на стандарте SQL 92.

SQLite — детище Д. Ричарда Хиппа (D. Richard Hipp) из Wyrick & Company, Inc., а начиная с декабря 2000 года проект SQLite развивается благодаря усилиям многих энтузиастов. Несмотря на кажущуюся коммерческую направленность SQLite — бесплатный продукт, полностью являющийся достоянием общественности и освобождающий пользователей от ответственности, предполагаемой какой-либо лицензией, даже такой либеральной, как GPL.

Как получить SQLite?

Если при установке PHP SQLite-расширение не было явно отключено, то все, что нужно для написания приложений на основе этой СУБД, уже есть. Для независимых от PHP экспериментов с SQLite можно использовать утилиту командной строки, которая доступна на странице www.sqlite.org/download.html. Web-сайт SQLite www.sqlite.org/ содержит гораздо больше информации о продукте, чем можно было включить в это приложение; в то же время там не много сказано об использовании SQLite с PHP. Благодаря PHP проект SQLite в скором времени безусловно завоеует широкую популярность, пока же наилучшим источником информации по этой теме является сайт PHP www.php.net/.

Почему стоит (или не стоит) использовать SQLite?

Мнения программистов расходятся, поэтому приведенные ниже замечания по SQLite предлагаются не как преимущества или недостатки, а скорее как факты, которые можно оценить и на их основе принять собственное решение относительно требований разработки конкретного приложения.

1. SQLite — это, как уже отмечалось, “бессерверная” технология. В отличие от многих других баз данных, система SQLite не содержит клиентского и серверного компонентов. Вместо этого SQLite представляет собой C-библиотеку для обработки данных, которая предназначена для простой интеграции и разработки приложений. С другой стороны, удаленный доступ к базе данных реализован практически вне всякого сомнения.
2. SQLite не поддерживает типы данных. На Web-сайте SQLite опубликована интересная дискуссия по этой теме. Она сосредоточена вокруг последовательной защиты автором своей точки зрения по этому вопросу. Цитата: *“Это не дефект, а часть замысла. База данных предназначена для хранения и извлечения данных, и ей должно быть безразлично, в каком формате хранятся эти данные. Строгая типизация, используемая в большинстве других SQL-систем и закодированная в языке SQL, является вредным пережитком — это пример реализации, которая “проступает” через интерфейс. SQLite пытается преодолеть этот пережиток, позволяя хранить любые типы данных в любых столбцах, обеспечивая гибкость спецификации баз данных.”*

Не нужно быть пуристом в программировании, чтобы оценить влияние строгой типизации на производительность крупных реляционных баз данных. Однако “отсутствие типов” значительно упрощает проектирование и создание таблиц, а в случае SQLite таблицы действительно так малы, как это возможно. Это особенно важно в условиях ограниченности таких ресурсов, как память и дисковое пространство. И поскольку PHP также главным образом использует слабую типизацию, это не представляет реальной угрозы.

3. Неполный SQL: создатель SQLite Хипп подтверждает несколько недостающих частей SQL-стандарта: не реализованы ограничения внешнего ключа, триггеры поддерживаются не полностью, а только одиночные одновременные транзакции, не полностью реализованы многие типы JOIN-операций, в частности правое и полное внешнее объединение. Самый большой недостаток — недоступен синтаксис ALTER TABLE, хотя его появление ожидается в ближайшем будущем. В настоящее время для внесения изменений в структуру необходимо сохранить данные во временной таблице, удалить старую таблицу, создать ее заново с новой структурой, а затем импортировать данные из временной таблицы.

4. Схема привилегий: в списке недостающих элементов SQL также находятся команды GRANT и REVOKE. Хипп объясняет это тем, что привилегии бессмысленны во встраиваемой СУБД. Однако с этим можно не согласиться. Привилегии доступа, по существу, контролируются правами доступа к дисковым файлам, а это не легко реализовать в PHP, работающем в безопасном режиме (`safe_mode = on`). Если разработчику понадобится управлять привилегиями, то, по сути, ему придется разработать встроенную в приложение систему контроля привилегий.

Ниже перечислено еще несколько интересных моментов.

- ❑ Файлы данных легко распространяются независимо от платформы.
- ❑ Двоичные данные перед вставкой должны кодироваться в текстовом виде и декодироваться перед отображением. Для этого в PHP поддерживается внутренняя реализация SQLite, а, кроме того, можно использовать другие стандартные PHP-функции.
- ❑ Поддерживаются базы данных размером до 2 терабайт.
- ❑ Измерения скорости для небольших баз данных имеют подающие надежды результаты; общедоступной информации по тестированию крупных баз данных пока нет.

Если, изучив представленную выше информацию, вы решили, что SQLite хорошо подходит для ваших потребностей, можете переходить к чтению следующего раздела.

Использование SQLite в PHP

PHP предоставляет множество функций для работы с базами данных SQLite. Тот, кто уже использовал PHP с другой СУБД, может очень легко догадаться, как работать с SQLite-базами данных. Однако при работе с дисковыми файлами баз данных следует учитывать некоторые особенности этой системы.

Создание и поддержка соединений

В приведенной ниже таблице описаны общие функции для подключения и управления базами данных.

Функция	Синтаксис	Описание
<code>sqlite_open()</code>	<code>resource sqlite_open (string filename [, int mode [, string &error_message]])</code>	Открывает или создает базу данных SQLite. В качестве параметра принимает абсолютный или относительный путь и имя файла базы данных. Параметр <code>mode</code> (режим) в настоящее время не поддерживается библиотекой SQLite, но его планируется использовать для управления правами доступа к базе данных. Во время выполнения сценария для операций, не требующих записи в базу данных (например, <code>SELECT</code> -запрос), могут быть установлены права “только для чтения”. Сообщение об ошибке (<code>error_message</code>) представляет собой переданную по ссылке переменную, содержащую описательное сообщение об ошибке, переданное SQLite и поясняющее, почему невозможно было открыть базу данных

Окончание таблицы

Функция	Синтаксис	Описание
<code>sqlite_popen()</code>	<code>resource sqlite_popen (string filename [, int mode [, string &error_message]])</code>	Версия функции <code>sqlite_open</code> , создающая постоянное соединение. Функция ищет и возвращает существующий дескриптор, а если не находит его, то создает новое постоянное соединение
<code>sqlite_close()</code>	<code>void sqlite_close (resource dbhandle)</code>	Закрывает указанный дескриптор базы данных
<code>sqlite_libencoding()</code>	<code>string sqlite_libencoding (void)</code>	Возвращает кодировку, которая используется активной SQLite-библиотекой
<code>sqlite_libversion()</code>	<code>string sqlite_libversion (void)</code>	Возвращает номер версии активной SQLite-библиотеки
<code>sqlite_query()</code>	<code>resource sqlite_query (resource dbhandle, string query)</code>	Несмотря на то что имя функции подсказывает, что ее следует использовать для манипуляции данными и чтения информации из базы, эта функция оказывается полезной для создания таблиц данных. Она возвращает дескриптор результата для запросов на выборку данных и булево значение <code>true</code> или <code>false</code> для всех остальных типов запросов. В отличие от привычных для MySQL-функций, в данном случае дескриптор результата обязателен. Работа этой функции при манипуляции данными описывается в следующей таблице

Манипуляция данными

В следующей таблице описаны функции, предназначенные для чтения, изменения и сохранения данных.

Функция	Синтаксис	Описание
<code>sqlite_query()</code>	<code>resource sqlite_query (resource dbhandle, string query)</code>	Возвращает дескриптор результата с произвольным доступом для запроса <code>query</code> , направленного базе данных, заданной с помощью дескриптора <code>dbhandle</code>
<code>sqlite_unbuffered_query()</code>	<code>resource sqlite_unbuffered_query (resource dbhandle, string query)</code>	Аналогична функции <code>sqlite_query()</code> за исключением того, что результаты возвращаются в виде последовательного результирующего множества с однонаправленным доступом
<code>sqlite_seek()</code>	<code>bool sqlite_seek (resource result, int rownum)</code>	Ищет строку, заданную параметром <code>rownum</code> ; если находит, возвращает <code>True</code> , <code>False</code> — в противном случае
<code>sqlite_next()</code>	<code>bool sqlite_next (resource result)</code>	Передвигает указатель результата на следующую строку. Возвращает <code>False</code> , если следующей строки нет
<code>sqlite_rewind()</code>	<code>bool sqlite_rewind (resource result)</code>	Возвращается обратно к первой строке результирующего множества. Если результирующее множество пусто, возвращает <code>False</code>

Окончание таблицы

Функция	Синтаксис	Описание
<code>sqlite_current()</code>	<code>array sqlite_current (resource result [, int result_type [, bool decode_binary]])</code>	Возвращает текущую строку результирующего множества в виде массива, но в отличие от <code>sqlite_fetch_array()</code> не передвигает указатель на следующую строку
<code>sqlite_fetch_array()</code>	<code>array sqlite_fetch_array (resource result [, int result_type [, bool decode_binary]])</code>	Аналогична <code>sqlite_current()</code> , но после возвращения результата переходит на следующую строку. Если больше строк нет, возвращает <code>False</code>
<code>sqlite_has_more()</code>	<code>bool sqlite_has_more (resource result)</code>	Возвращает <code>True</code> , если в результирующем множестве еще есть доступные строки
<code>sqlite_fetch_single()</code>	<code>string sqlite_fetch_single (resource result [, int result_type [, bool decode_binary]])</code>	Возвращает первый столбец результирующего множества
<code>sqlite_column()</code>	<code>mixed sqlite_column (resource result, mixed index_or_name [, bool decode_binary])</code>	Возвращает столбец, заданный индексом или именем
<code>sqlite_array_query()</code>	<code>array sqlite_array_query (resource dbhandle, string query [, int result_type [, bool decode_binary]])</code>	Возвращает результаты запроса в виде нескольких массивов

Сведения об извлекаемых данных

Ниже приведены функции для работы с метаданными SQLite.

Функция	Синтаксис	Описание
<code>sqlite_changes()</code>	<code>int sqlite_changes (resource dbhandle)</code>	Возвращает количество строк, измененных последним SQL-оператором
<code>sqlite_field_name()</code>	<code>string sqlite_field_name (resource result, int field_index)</code>	Возвращает имя столбца, заданного параметром <code>field_index</code>
<code>sqlite_last_insert_rowid()</code>	<code>int sqlite_last_insert_rowid (resource dbhandle)</code>	Возвращает присвоенное значение первичного ключа для последней вставленной записи. Работает только для столбцов, определенных как <code>INTEGER PRIMARY KEY</code>
<code>sqlite_num_fields()</code>	<code>int sqlite_num_fields (resource result)</code>	Возвращает количество полей в результирующем множестве SQLite. Не работает с функцией <code>sqlite_unbuffered_query()</code>
<code>sqlite_num_rows()</code>		Возвращает количество строк в результирующем множестве. Не работает с небуферизированными результирующими множествами

Разные функции

В следующей таблице описаны функции, которые не попадают в другие категории, но являются весьма полезными в некоторых ситуациях.

Функция	Синтаксис	Описание
<code>sqlite_busy_timeout()</code>	<code>void sqlite_busy_timeout(resource dbhandle, int milliseconds)</code>	По умолчанию PHP ожидает освобождения SQLite-ресурса в течение 60 секунд. Время ожидания должно быть указано в миллисекундах; значение 0 приводит к тому, что PHP немедленно возвращает код состояния <code>SQLITE_BUSY</code> . Обычно стандартного значения более чем достаточно, и вместе с тем оно не настолько велико, чтобы сделать отладку утомительной
<code>sqlite_create_aggregate()</code>	<code>bool sqlite_create_aggregate(resource dbhandle, string function_name, mixed step_func, mixed finalize_func [, int num_args])</code>	Реализует вторичные функции, как правило, определяемые пользователем, для вычисления результатов группировки данных результирующего множества. Функция, указанная в параметре <code>step_func</code> , обрабатывает каждую строку результирующего множества, а затем после того, как все строки обработаны, <code>finalize_func</code> возвращает агрегированное значение. Параметр <code>func_name</code> определяет функцию, которая должна использоваться в SQL-операторе
<code>sqlite_create_function()</code>	<code>bool sqlite_create_function(resource dbhandle, string function_name, mixed callback [, int num_args])</code>	Параметр <code>function_name</code> определяет имя функции, которая будет использоваться в SQL-операторах; значением параметра <code>callback</code> может быть имя любой PHP-функции обратного вызова, которая обработает указанную SQL-функцию. Необязательный параметр <code>num_args</code> определяет количество параметров, которые необходимо передать функции
<code>sqlite_error_string()</code>	<code>string sqlite_error_string(int error_code)</code>	Возвращает читабельную строку, соответствующую переданному коду ошибки <code>error_code</code> , который обычно возвращается функцией <code>sqlite_last_error()</code>
<code>sqlite_escape_string()</code>	<code>string sqlite_escape_string(string item)</code>	Заключает в кавычки и экранирует специальные символы в строке, которая будет использоваться в SQL-операторе; может оказаться полезной даже для двоичных данных, хотя их лучше обрабатывать с помощью функции <code>sqlite_udf_encode_binary()</code>
<code>sqlite_last_error()</code>	<code>int sqlite_last_error(resource dbhandle)</code>	Возвращает код ошибки, возвращенный последней выполненной функцией для указанного дескриптора базы данных
<code>sqlite_udf_decode_binary()</code>	<code>string sqlite_udf_decode_binary(string data)</code>	Очень полезная, специфическая для SQLite функция декодирования двоичных данных, сохраненных в SQLite-таблице с помощью описанной ниже парной функции
<code>sqlite_udf_encode_binary()</code>	<code>string sqlite_udf_encode_binary(string data)</code>	Кодирует двоичные данные так, чтобы их можно было без искажений возвращать с помощью запросов

Практическое использование SQLite

Большинство ключевых PHP-функций для работы с другими базами данных дублированы для SQLite. Однако вряд ли разработчик захочет внедрять незнакомое средство, не прошедшее документированного крупномасштабного тестирования в реальных условиях, и это правильно. Подходящими приложениями для экспериментирования с SQLite могли бы стать гостевая книга, небольшая система складского учета, система управления пользователями сайта или диспетчер форм.

Далее приведен пример разработки на базе SQLite простого приложения для управления персональной библиотекой.

Приложение “Персональная библиотека”

Рамки книги не позволяют описать создание очень сложного приложения, поэтому такие моменты, как наличие нескольких авторов, нескольких изданий и т.д., не принимаются во внимание. С другой стороны, затронутые темы помогут разработчику обрести уверенность в своих силах. В этом разделе в качестве примеров приведены отдельные фрагменты кода, а весь код окончательного продукта доступен для загрузки с сайта издательства.

Для начала необходимо подготовить каталог для хранения SQLite-баз данных. Лучше всего использовать каталог, находящийся за пределами Web-дерева, но доступный пользователю, от имени которого работает Web-сервер, чтобы приложение могло считывать и записывать в него данные. Для целей данного примера в Unix можно использовать каталог `/var/sqlite` или создать подкаталог `sqlite` в соответствующем каталоге Windows-системы.

Затем следует определить структуру таблиц. По каждой книге желательно хранить следующую информацию:

- ☐ название;
- ☐ автор;
- ☐ год издания;
- ☐ издательство;
- ☐ дата прочтения;
- ☐ рейтинг.

После чего можно заставить SQLite создать соответствующие таблицы.

Создание базы данных и таблиц

Для создания необходимой структуры базы данных в этом случае используется короткий сценарий `books_table_create.php`, который очень похож на сценарий для работы с MySQL-базой данных, созданный в главах 9, 10 и 11:

```
<html>
<head>
<title>Создание базы данных и таблиц "Персональной библиотеки"</title>
</head>
<body>
<?php
echo ( '<p>Создание базы данных и таблиц с помощью SQLite версии: '
. sqlite_libversion() . '</p>' );
```

Очевидно, что первую строку PHP-кода в этом сценарии использовать не обязательно, но поскольку в данном случае SQLite запускается на машине впервые, эта строка определено может предоставить полезную информацию. Сбой в этой строке объяснит многие последующие проблемы. Корректное выполнение строки сообщает о том, насколько современной является библиотека; эта информация полезна для отладки.

Далее необходимо определить базу данных, которую будет использовать сценарий, а затем открыть дескриптор с помощью функции `sqlite_open()`. Следует отметить, что если указанная база данных еще не существует, то эта функция создает ее:

```
$libraryDB = "/var/sqlite/library";
if ($db = @sqlite_open($libraryDB, 0666, $error)) {
    print ( '<p>SQLite-база данных library успешно создана</p>' );
    $sql = "CREATE TABLE books (
        book_id INTEGER PRIMARY KEY,
        book_title VARCHAR,
        book_author VARCHAR,
        book_pub_year INTEGER,
        book_publisher VARCHAR,
        book_read VARCHAR,
        book_score VARCHAR,
        book_loan VARCHAR
    )";
```

Предположим, что база данных успешно создана и распечатаны некоторые сообщения о текущем состоянии дел:

```
if ( @sqlite_query($db, $sql) ) {
    print ( '<p>SQLite-таблица books успешно создана</p>' );
} else {
    print ( '<p>SQLite-таблица books не создана из-за ошибки:
    <br />' . sqlite_error_string(sqlite_last_error($db)) . '</p>' );
}
} else {
    print ( '<p>SQLite-база данных library не создана из-за ошибки:
    <br />' . $error . '</p>' );
}
?>
</body>
</html>
```

Здесь следует отметить несколько моментов. Во-первых, при перехвате ошибки результат функции `sqlite_last_error()` непосредственно передается функции `sqlite_error_string()`. Подобная краткая запись впоследствии может принести реальную экономию времени.

Во-вторых, несмотря на “отсутствие” типов в SQLite, в определении таблицы включены типы (хотя и очень общие). SQLite принимает (и возвращает) информацию о самых базовых SQL-типах данных, даже несмотря на то, что не использует ее. Впоследствии это позволит другим программистам быстрее понять замыслы создателя программы и может послужить хорошим напоминанием ему самому, когда понадобится перенести приложение на другую СУБД.

Наконец, необходимо отметить важное исключение в аспекте обычной работы SQLite с типами данных. Столбец `book_id` определен как `INTEGER PRIMARY KEY`. Это сообщает SQLite, что данное поле предназначено в качестве уникального индекса. Той же цели можно достичь, используя только SQL-слова `PRIMARY KEY` для нечислового первичного ключа или `UNIQUE` для того, чтобы записи в любом не ключевом поле не повторялись. Хотя чистый эффект любой из этих трех комбинаций один и тот

же, использование ключевых слов PRIMARY KEY для поля типа INTEGER имеет побочный эффект — это поле определяется как поле с автоинкрементными значениями.

Когда SQLite принимает SQL-оператор INSERT для таблицы и значение поля `book_id` равно NULL или не определено, в это поле вставляется следующее целое значение. Если автоинкрементные типы не подходят, то это препятствие можно легко обойти, вставляя собственные значения. SQLite свободно пропускает эти значения и переходит к следующей значимой инструкции.

Запустите сценарий в браузере. На рис. В.1 показаны сообщения, которые должны появиться на странице.

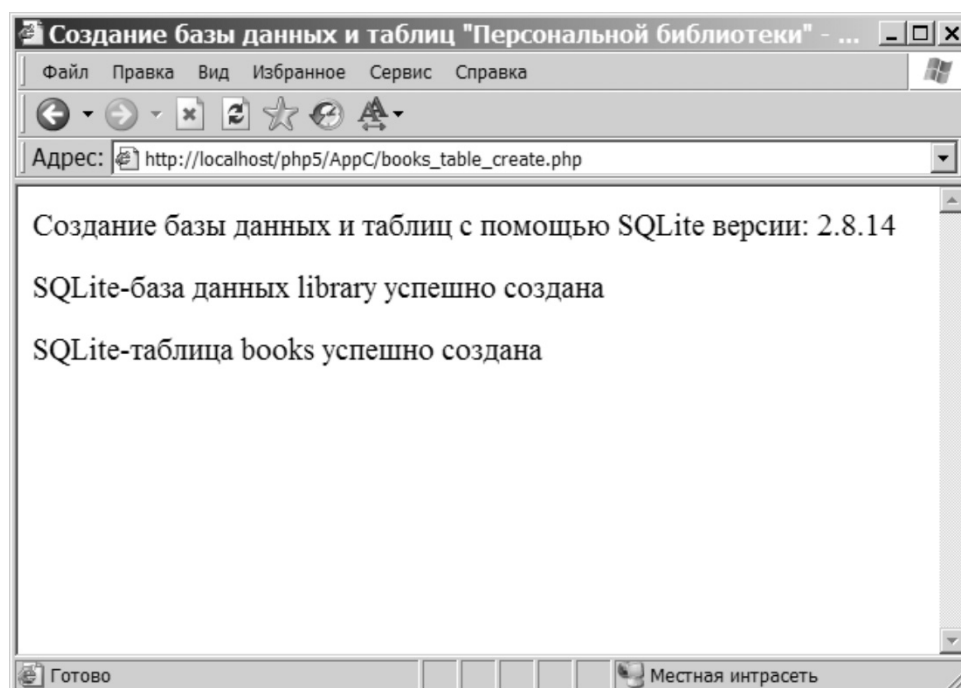


Рис. В.1.

Форма ввода данных

SQLite пока еще ничего “не знает” о персональной библиотеке. Поэтому первый повторно используемый блок кода получает от пользователей данные о книгах и вставляет эти данные в таблицу `books`.

Здесь нет ничего нового, если вы изучили главы о работе с базами данных. Главное отличие заключается в замене слова `mysql_` в именах функций на `sqlite_`. Сами SQL-операторы идентичны тем, которые использовались для MySQL. Это часть хорошо продуманного стандарта. Последствия такой гибкости просто удивительные. Конечно, существуют некоторые отличия в реализациях SQL этих СУБД, но изучив одну гибкую систему, вы сможете без труда разобраться с другими.

Ниже приведен код сценария `books_insert.php`:

```

<html>
<head>
<title>Персональная библиотека: добавление книги</title>
</head>
<body>
<h1 align="center">Добро пожаловать в библиотеку</h1>
<?php

```

Затем используется та же управляющая структура и метод сообщения об ошибках, который использовался при создании таблицы; фактически, эта часть кода просто скопирована:

```

if ($_POST['action'] == "Записать") {
    $libraryDB = "/var/sqlite/library";
    if ($db = @sqlite_open($libraryDB, 0666, $error)) {
        // Библиотека открыта. Вставляем данные с помощью SQL-оператора INSERT
        $sql = "INSERT INTO books(book_title, book_author,
            book_pub_year, book_publisher, book_read,
            book_score, book_loan)
            values('$ _POST[form_title]', '$ _POST[form_author]',
            '$ _POST[form_pub_year]', '$ _POST[form_publisher]',
            '$ _POST[form_read]',
            '$ _POST[form_score]', '$ _POST[form_loan]')";
        if ( @sqlite_query($db, $sql) ) {
            print ( '<p>Книга '. $_POST[form_title] . ' успешно
                добавлена в таблицу books.</p>' );
            print('<p>При желании можно добавить другие книги.</p>');
        } else {
            print ( '<p>Книга не была добавлена '.
                $_POST[form_title] . ' из-за ошибки:
                <br />' . sqlite_error_string(sqlite_last_error ($db)) . '</p>' );
        }
    } else {
        print ( '<p>SQLite-база данных library не была открыта
            из-за ошибки: <br />' . $error . '</p>' );
    }
} else {
    print ( '<p>Пожалуйста, введите информацию о книге. Поля название и автор
        обязательны</p>' );
}
?>

```

Затем выводится простая HTML-форма независимо от того, выполнялась ли вставка данных. Это дает пользователю возможность легко ввести последовательно несколько книг. Кроме того, добавляется ссылка на главную страницу, даже если она еще не создана, поэтому заново вводить адрес сценария не придется:

```

<p><form action="books_insert.php" method="post"><br />
Введите название книги:<br />
<input type="text" name="form_title"><br />
Введите автора книги (имя и фамилию):<br />
<input type="text" name="form_author"><br />
Введите год издания<br />
<input type="text" name="form_pub_year"><br />
Введите название издательства:<br />
<input type="text" name="form_publisher"><br />
Вы уже прочли эту книгу?<br />
<select name="form_read"><br />
<option value="Да" SELECTED>Да</option><br />
<option value="Нет">Нет</option><br />
</select><br />
Присвойте книге рейтинг (от 1 до 5)<br />

```

```
<input type="text" name="form_score"><br />
Книга выдана:<br />
<input type="text" name="form_loan"><br />
<input type="submit" name="action" value="Записать"><br />
</form></p>
<p><a href="index.php">На главную страницу библиотеки.</a></p>
</body>
</html>
```

На рис. В.2 показан вид начальной страницы до того, как пользователь ввел данные.

Теперь можно ввести некоторые данные по книгам, чтобы протестировать следующие сценарии.

Персональная библиотека: добавление книги - Microsoft Intern...

Файл Правка Вид Избранное Сервис Справка

Адрес: http://localhost/php5/AppC/books_insert.php

Добро пожаловать в библиотеку

Пожалуйста, введите информацию о книге. Поля название и автор обязательны

Введите название книги:
НР 5 для начинающих

Введите автора книги (имя и фамилию):
Дэйв Мерсер

Введите год издания
2005

Введите название издательства:
Издательский дом Ви

Вы уже прочли эту книгу?
Да

Присвойте книге рейтинг (от 1 до 5)
5

Книга выдана:
Дэйв Мерсер

Записать

На главную страницу библиотеки.

Готово Местная интрасеть

Рис. В.2.

Главная страница и листинг книг

Главная страница фактически решает две задачи. Во-первых, она предоставляет интерфейс к остальным функциям приложения, которые можно назвать панелью управления. Во-вторых, здесь перечислены книги, записанные в базу, а также предоставляется возможность редактировать или удалять сведения о них. Это очень важный этап, поскольку несмотря на отсутствие ошибок в предыдущем сценарии, убедиться в работоспособности приложения вы сможете, только получив информацию, которая возвращается базой данных. Назовем этот сценарий `index.php`:

```
<html>
<head>
<title>Персональная библиотека: панель управления</title>
</head>
<body>
<h1 align="center">Добро пожаловать в библиотеку</h1>
<p><a href="books_insert.php">Добавить книгу</a></p>
```

Итак, пользователю предоставляется возможность добавить книгу в базу данных.

Первые несколько строк PHP-кода как обычно создают подключение к базе данных и обрабатывают ошибки подключения:

```
<?php
$libraryDB = "/var/sqlite/library";
if ($db = @sqlite_open($libraryDB, 0666, $error)) {
```

Затем выполняется проверка того, как был вызван сценарий: с идентификационным номером книги (в этом случае сценарий удаляет книгу) или без. Если страница вызывается непосредственно, т.е. без GET-параметров, то этот блок кода не выполняется. Если в строке запроса был передан номер книги, то соответствующая запись удаляется из базы данных, а пользователю возвращается страница с номером удаленной книги.

```
if ($_GET['id']) {
    $sql = "DELETE FROM books where book_id = '$_GET[id]'" ;
    if ( @sqlite_query($db, $sql)) {
        print('<p>Запись ' . $_GET['id'] . ' успешно удалена</p>');
    } else {
        print ( '<p>Не удалось удалить запись ' . $_GET['id'] . ' из-за ошибки:
        <br />' . sqlite_error_string(sqlite_last_error($db)) . '</p>' );
    }
}
```

Предположим, что этот блок кода выполняется без ошибок, тогда следующим этапом будет вывод списка книг. SQL-оператор для SQLite такой же, как и для MySQL. Каждая последующая запись с помощью цикла `while` помещается в массив, а затем распечатывается список:

```
// Библиотека открыта с помощью INSERT-запроса
$sql = "SELECT * FROM books";
if ( $result = sqlite_query($db, $sql) ) {
    print('<p>В библиотеке ' . sqlite_num_rows($result) . ' книг.</p>');
    while ($book = sqlite_fetch_array($result)) {
        print ( '<p><strong>Книга:</strong> ' . $book[book_title] .
        ' <strong>Автор</strong>: ' . $book[book_author] . '<br />' .
        ' <strong>Издательство:</strong> ' . $book[book_publisher] . '; ' .
        $book[book_pub_year] . ' <strong>Прочитана:</strong> ' .
```

```

        $book[book_read] .
        ' <strong>Рейтинг:</strong> ' . $book[book_score] . '<br />' .
        '<strong>Принадлежит:</strong> ' . $book[book_loan] . '<br />' .

```

Теперь необходимо добавить ссылки для редактирования и удаления записей:

```

        ' <a href="books_edit.php?id=' . $book[book_id] . '">Редактировать</a> | ' .
        ' <a href="index.php?id=' . $book[book_id] . '">Удалить</a></p>' );
    }
} else {
    print ( '<p>Не удалось прочитать таблицу books из-за ошибки:
    <br />' . sqlite_error_string(sqlite_last_error($db)) . '</p>' );
}
} else {
    print ( '<p>Не удалось открыть SQLite-базу данных library из-за ошибки:
    <br />' . $error . '</p>' );
}
?>

```

Код страницы завершается ссылкой для добавления новой книги и закрывающими HTML-тегами:

```

<p><a href="books_insert.php">Добавить книгу</a></p>
</body>
</html>

```

Результат выполнения сценария показан на рис. В.3.

В следующем разделе описан код для редактирования записей.

Редактирование записи

Последний сценарий в приложении самый сложный, но несмотря на это в нем нет ничего нового. Ниже приведен код сценария для редактирования записей `books_edit.php`:

```

<html>
<head>
<title>Персональная библиотека: редактирование записи</title>
</head>
<body>
<h1 align="center">Добро пожаловать в библиотеку</h1>
<?php
$libraryDB = "/var/sqlite/library";
if ($db = @sqlite_open($libraryDB, 0666, $error)) {
    // Библиотека открыта, выбираем операцию
    if ($_GET['id']) {

```

Сначала выводится страница.

Страница `books_edit.php` всегда вызывается с идентификатором книги, который можно включить в SQL-запрос. После того, как запрос вернет необходимую информацию, можно распечатать форму:

```

$sql = "SELECT * FROM books where book_id = '$_GET[id]'";
if ( $result = sqlite_query($db, $sql) ) {
    $book = sqlite_fetch_array($result);
    print("<p><form action=\"books_edit.php\" method=\"post\"><br />");
    print("<input type=\"text\" name=\"form_id\" value=\"$_GET[id]\"><br />");
    print("Введите название книги:<br />");
    print("<input type=\"text\" name=\"form_title\"
    value=\"$book[book_title]\"><br />");

```

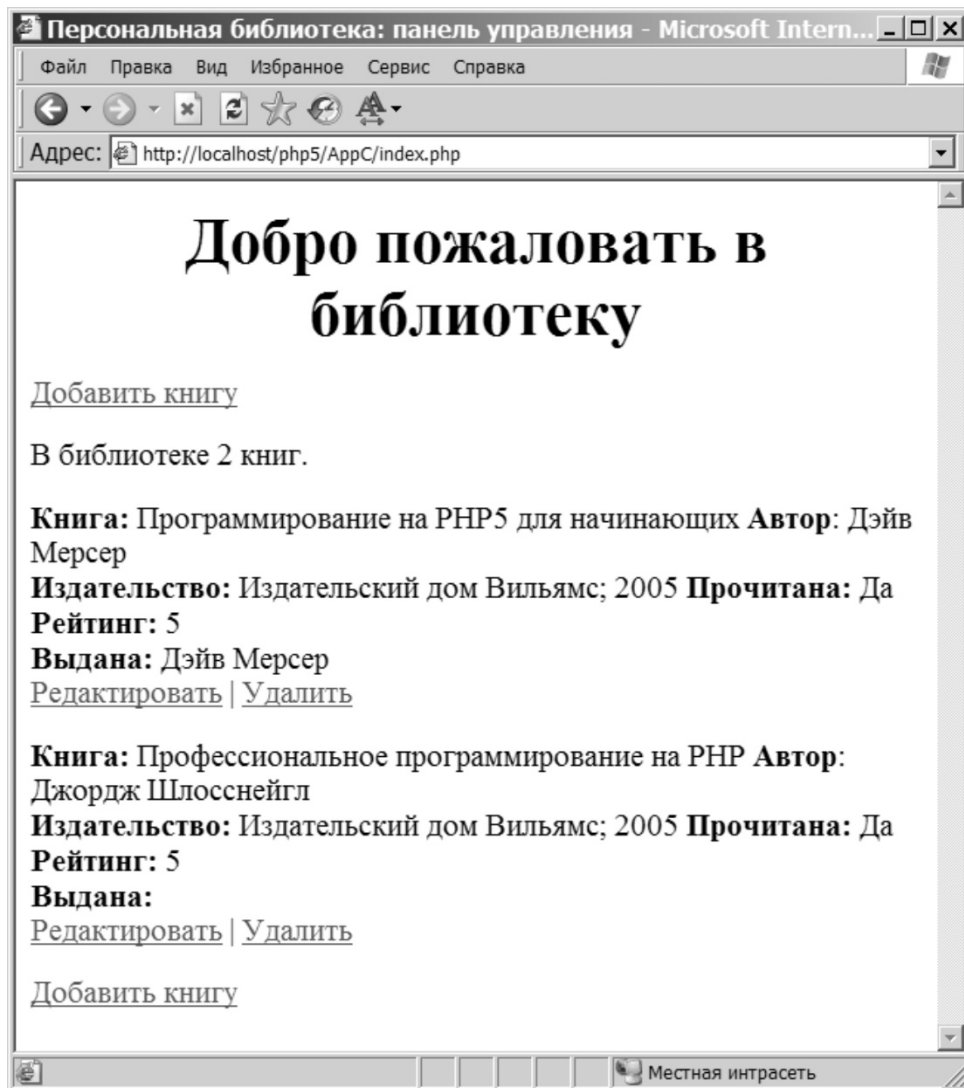


Рис. В.3.

```

print("Введите автора (имя и фамилию):<br />");
print("<input type=\"text\" name=\"form_author\"
value=\"$book[book_author]\"><br />");
print("Введите год издания<br />");
print("<input type=\"text\" name=\"form_pub_year\"
value=\"$book[book_pub_year]\"><br />");
print("Введите название издательства:<br />");
print("<input type=\"text\" name=\"form_publisher\"
value=\"$book[book_publisher]\"><br />");
print("Вы уже прочли эту книгу?<br />");
print("<select name=\"form_read\"><br />");
print("<option value=\"Да\">");

```

```

if ($book[book_read] == 'Да') { print("selected"); }
print(">Да</option><br />");
print("<option value=\"Нет\"");
if ($book[book_read] == 'Нет') { print("selected"); }
print(">Нет</option><br />");
print("</select><br />");
print("Назначьте книге рейтинг (от 1 до 5)<br />");
print("<input type=\"text\" name=\"form_score\" "
value=\"$book[book_score]\"><br />");
print("Книга выдана:<br />");
print("<input type=\"text\" name=\"form_loan\" "
value=\"$book[book_loan]\"><br />");
print("<input type=\"submit\" name=\"action\" "
value=\"Обновить\"><br />");
print("</form></p>");

} else {
    print ( '<p>Не удалось получить запись из-за ошибки:
    <br />' . sqlite_error_string(sqlite_last_error($db)) . '</p>' );
}

```

В этом примере вся форма распечатывается с помощью PHP, но можно было бы использовать HTML с включениями PHP-вызовов, как это делалось в других примерах.

После того как пользователь внесет необходимые изменения, сценарий получает значения из глобального массива \$_POST и использует их в UPDATE-операторе, который модифицирует соответствующие значения в базе данных:

```

} else if ($_POST[form_id]) {
    $sql = "update books set book title = '$_POST[form_title]',
    book_author = '$_POST[form_author]',
    book_pub_year = '$_POST[form_pub_year]',
    book_publisher = '$_POST[form_publisher]',
    book_read = '$_POST[form_read]',
    book_score = '$_POST[form_score]',
    book_loan = '$_POST[form_loan]'
    where book_id = '$_POST[form_id]';
    print $sql;
    if (sqlite_query($db, $sql)) {
        print ( '<p>Изменения были успешно внесены.</p>' );
    } else {
        print ( '<p>Не удалось обновить запись из-за ошибки:
        <br />' . sqlite_error_string(sqlite_last_error($db)) .
        '</p>' );
    }
}
} else {
    print ( '<p>Не удалось открыть SQLite-базу данных library из-за ошибки:
    <br />' . $error . '</p>' );
}
?>
<p><a href="index.php">На главную страницу библиотеки.</a></p>
</body>
</html>

```

Приложение “Персональная библиотека” можно считать функционально завершенным, хотя в нем отсутствуют некоторые функции, которые могли бы сделать его более удобным. Ниже перечислены возможные усовершенствования приложения в порядке от самого простого к самому сложному.

- ❑ Исправить SQL-код в сценарии index.php так, чтобы записи сортировались и чтобы пользователь получил возможность выбирать порядок сортировки.

- ❑ Используя функцию `sqlite_escape_string()`, сделайте операторы вставки и обновления более лояльными к специальным символам.
- ❑ Используя функцию `sqlite_num_rows()` и SQL-предложение `LIMIT`, напишите простую подпрограмму разбиения списка книг в `index.php` на страницы. Первая страница приложения может сбивать пользователей с толку, если на ней будет выводиться большое количество записей.

SQLite — отличный компактный инструмент. Несмотря на сравнительно молодой возраст, SQLite является довольно стабильной, мощной и функциональной технологией. Конечно, не все разработчики захотят переносить на нее свои приложения, но сам факт того, что поддержка SQLite включена в PHP, является надежным подтверждением качеств этой технологии. Теперь читатель имеет хорошие знания основ SQLite и может самостоятельно продолжать изучение этой технологии.



ODBC

ODBC, или Open DataBase Connectivity (открытый интерфейс доступа к базам данных), представляет собой разновидность API (application programming interface — программный интерфейс приложений), позволяющую использовать SQL (Structured Query Language — язык структурированных запросов) для работы с базами данных разных производителей. ODBC создает средний уровень (*драйвер базы данных*) между приложением и базой данных, позволяющий приложению отправлять базе данных SQL-команды (запросы), которые она может распознать и корректно обработать. Например, чтобы использовать в PHP-приложении базу данных Access или SQL Server в качестве хранилища информации, можно задействовать ODBC-драйвер для форматирования SQL-команд так, чтобы любая база данных распознавала эти команды и корректно отвечала на них. По существу, драйвер базы данных ODBC преобразовывает SQL-команды в формат, понятный той или иной базе данных.

В этом приложении представлен перечень PHP-функций, которые можно использовать в ODBC-совместимых базах данных (таких как Access и Microsoft SQL Server). Кроме того, здесь рассматривается пример создания имени источника данных (data source name, DSN) в Windows, а также пример PHP-приложения (`odbc.php`) для Windows, которое подключается к базе данных SQL Server. Эти примеры можно использовать также для подключения к базам данных Access.

Общие ODBC-функции

Независимо от того, как осуществляется доступ к базе данных, некоторые действия, такие как открытие соединения, запуск SQL-запроса и закрытие соединения, являются обязательными. В главах 9–11 рассматривались функции для подключения к базе данных (например, к базе данных MySQL или SQLite). В PHP (если расширение SQLite установлено статически или загружается динамически) имеются функции для открытия соединения с базой данных (`sqlite_open()`), запуска SQL-запроса (`sqlite_array_query()`) и закрытия соединения (`sqlite_close()`).

Эти функции наряду с аналогичными функциями для MySQL- и PostgreSQL-баз данных встроены в PHP и не являются ODBC-функциями. Однако они выполняют ту же работу, что и ODBC-функции (например, `odbc_connect()`, `odbc_exec()` и `odbc_close()`), хотя и несколько быстрее, чем ODBC-функции. Поэтому если в PHP имеются встроенные функции для запуска SQL в определенной базе данных, то применять следует именно их; они, скорее всего, будут быстрее и эффективнее, кроме того, они позволяют использовать специфические для базы данных возможности, доступа, которые ODBC не предоставляет. Если такие функции в PHP отсутствуют, то высока вероятность того, что существует ODBC-драйвер, который можно использовать в PHP-приложении.

ODBC-функции в PHP

В приведенной ниже таблице представлено описание распространенных ODBC-функций PHP.

ODBC-функции	Описание
<code>odbc_connect()</code> и <code>odbc_pconnect()</code>	Если предположить, что база данных уже создана, первое, что следует сделать, это создать подключение к этой базе данных, через которое можно отправлять ей команды и получать результаты. Эта функция создает подключение, используя переданное ей имя источника данных (DSN), а также имя пользователя, пароль и необязательный тип курсора. Если база данных не требует имени пользователя и пароля, то соответствующие параметры можно не указывать. Функция <code>odbc_pconnect()</code> работает аналогично <code>odbc_connect()</code> , за исключением того, что соединение остается постоянно открытым и впоследствии при необходимости его можно использовать снова; это обычно сокращает время доступа к базе данных
<code>odbc_close()</code> , <code>odbc_close_all()</code> и <code>odbc_free_result()</code>	Функции <code>odbc_close()</code> и <code>odbc_close_all()</code> закрывают соединение. Эти функции в качестве параметра принимают идентификатор соединения (полученный от <code>odbc_connect()</code>). Функция <code>odbc_free_result()</code> высвобождает все ресурсы, связанные с возвращенным результатом, и часто используется перед закрытием соединения, чтобы гарантировать, что для остальных частей сценария имеется достаточно доступной памяти
<code>odbc_exec()</code> , <code>odbc_execute</code> и <code>odbc_prepare</code>	Функция <code>odbc_exec()</code> принимает идентификатор соединения и SQL-запрос, отформатированный в виде строки, после чего подготавливает и выполняет этот запрос. Некоторые запросы, такие как INSERT, UPDATE и DELETE, не возвращают результирующих множеств, и поэтому могут запускать эту функцию самостоятельно. Запрос SELECT должен возвращать результаты, поэтому результирующее множество можно записать в переменную, чтобы использовать результаты для последующей обработки в PHP-приложении. Работу функции <code>odbc_exec()</code> можно разделить на две части, используя функцию <code>odbc_prepare()</code> перед функцией <code>odbc_execute()</code> . Примечание: функция <code>odbc_do()</code> аналогична функции <code>odbc_exec()</code>
<code>odbc_num_rows()</code>	Часто после получения результирующего множества необходимо узнать количество строк в нем. Например, если используется SELECT-запрос для выборки всех записей о заказах для определенного клиента, можно получить количество этих записей, просто передав результирующее множество в качестве аргумента функции <code>odbc_num_rows()</code>

ODBC-функции	Описание
odbc_fetch_row(), odbc_result() и odbc_fetch_array()	Функция <code>odbc_fetch_row()</code> возвращает строку данных из результирующего множества; затем можно передать эту строку функции <code>odbc_result()</code> , которая возвращает значение по имени поля в этой строке. Функция <code>odbc_fetch_array()</code> не документирована, но она возвращает строку из результирующего множества в виде ассоциативного массива, так что каждый элемент массива получает имя по имени поля и содержит соответствующее значение. Хотя функции <code>odbc_fetch_array()</code> необходимо передавать номер строки (возвращаемый массив индексируется, начиная с нуля), каждый вызов функции <code>odbc_fetch_row()</code> , если он выполняется без указания числа, просто возвращает следующую строку результирующего множества
odbc_commit(), odbc_autocommit() и odbc_rollback()	Чтобы убедиться, что все части определенной операции над базой данных, а не только один или несколько SQL-команд, выполнены полностью, применяют транзакции. Если весь набор команд выполнен успешно, то транзакция завершается, в противном случае база данных возвращается в исходное состояние. Отключение автозавершения транзакции путем передачи идентификатора результата (который получен во время создания подключения) и значения <code>FALSE</code> в функцию <code>odbc_autocommit()</code> дает тот же эффект, что и начало транзакции. С этого момента можно запускать SQL-команды до тех пор, пока транзакция не будет завершена (с помощью функции <code>odbc_commit()</code>); если возникает ошибка, то можно вызывать функцию <code>odbc_rollback()</code> , которая вернет базу данных в предыдущее состояние (аннулирует транзакцию)
odbc_error() и odbc_errormsg()	Функция <code>odbc_error()</code> возвращает код последней ошибки соединения. Функция <code>odbc_errormsg()</code> возвращает текст соответствующего сообщения об ошибке. Эти функции часто используются непосредственно после выполнения запроса и позволяют определить, был ли запрос правильно обработан

Другие ODBC-функции

Остальные ODBC-функции используются главным образом для получения более подробной информации о соединении или базе данных, а также для специфической обработки данных. Например:

- ☐ `odbc_tables()` возвращает список таблиц в базе данных;
- ☐ `odbc_tableprivileges()` возвращает таблицы и права доступа для каждой из них;
- ☐ `odbc_statistics()` возвращает имя владельца таблицы и индексы;
- ☐ `odbc_procedures()` возвращает хранимые процедуры базы данных;
- ☐ `odbc_field`-функции возвращают информацию о полях (длину, имя, тип данных и т.д.).

Использование ODBC в Windows и Linux

Поддержка ODBC входит в Windows-инсталляцию PHP. Для запуска ODBC-функция в Linux-инсталляции PHP необходимо либо скомпилировать PHP с поддержкой ODBC, либо загружать ODBC-расширение динамически. На сайте www.iodbc.org имеется подробная инструкция по использованию ODBC в Linux-инсталляции PHP.

ODBC-параметры в конфигурации PHP

Так как средства поддержки ODBC компилируются в PHP, для того чтобы заставить их работать, изменять настройки в `php.ini` не обязательно. Однако следует заметить, что несколько параметров влияют на некоторые ODBC-функции в PHP; эти параметры описываются ниже.

- ❑ `odbc.default_db`: задает строку, определяющую используемую по умолчанию базу данных на случай, если функция `odbc_connect()` или `odbc_pconnect()` вызывается без указания базы данных.
- ❑ `odbc.default_user`: задает строку, определяющую имя пользователя по умолчанию на случай, если функция `odbc_connect()` или `odbc_pconnect()` вызывается без указания имени пользователя.
- ❑ `odbc.default_pw`: задает строку, определяющую стандартный пароль, который используется, если функция `odbc_connect()` или `odbc_pconnect()` используется без указания пароля.

Конфигурационные параметры также можно использовать для того, чтобы разрешать или проверять постоянные соединения, задавать максимальное количество постоянных соединений (`odbc.allow_persistent`, `odbc.check_persistent` и `odbc.max_persistent`), задавать максимальное количество соединений, включая постоянные (`odbc.max_links`), а также настраивать обработку LONG-полей или бинарных данных (`odbc.defaultlrl` и `odbc.defaultbinmode`).

Пример использования ODBC и SQL Server в PHP-приложении для Windows

Для работы этого примера с использованием ODBC в PHP, необходима база данных SQL Server с минимум одной таблицей (в данном случае используется таблица “emails”), а также DSN — область для хранения необходимой информации о соединении (можно создать соединение без использования DSN, включив всю информацию о соединении в строку подключения; если настроить DSN невозможно, то это единственный способ заставить ODBC работать).

Создание базы данных Microsoft SQL Server

Прежде чем создавать базу данных, необходимо установить и настроить SQL Server. В этом примере используется SQL Server 2000 в Windows 2000. Для управления SQL Server используется программа Enterprise Manager. Ниже описан процесс подготовки примера.

1. Открыть диспетчер SQL Server (Пуск ⇒ Программы ⇒ Microsoft SQL Server ⇒ Enterprise Manager).
2. Открыть узел локального сервера, как показано на рис. Г.1.
3. Щелкнуть правой клавишей мыши на узле Databases и выбрать в контекстном меню создание новой базы данных (пункт New Database).
4. Присвоить создаваемой базе имя (в примере база данных называется BPHP5).

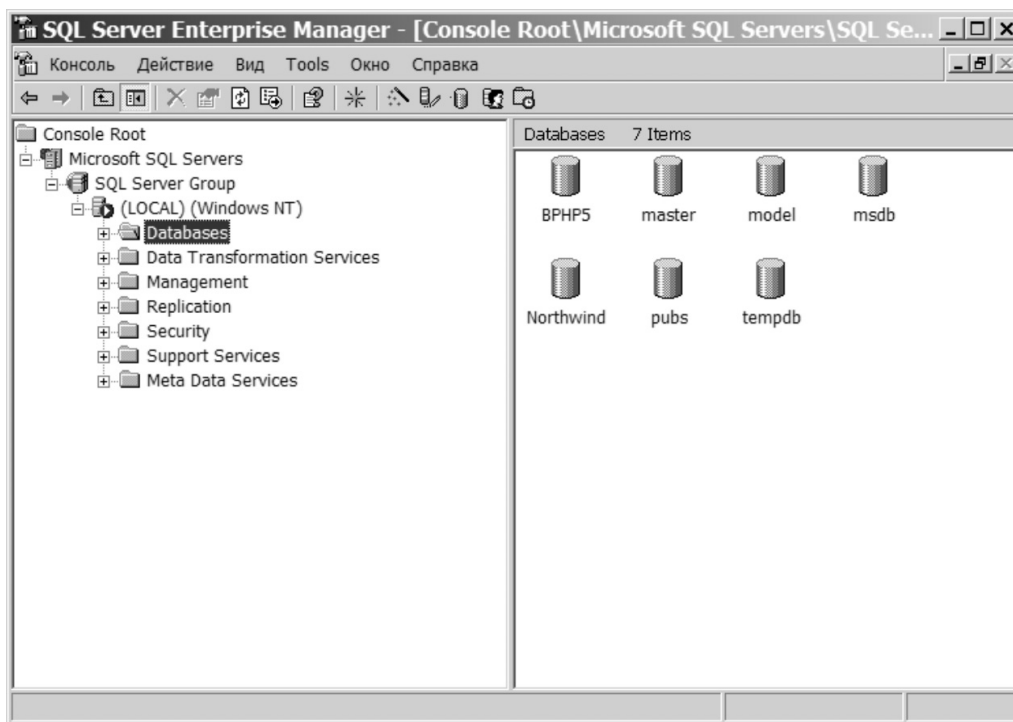


Рис. Г.1.

5. Открыть узел новой базы данных и создать в ней таблицу, щелкнув правой клавишей на узле Tables и выбрав в контекстном меню пункт New Table.
6. В режиме конструктора введите имя поля email_id, установите для него тип данных int и сделайте это поле уникальным (присвойте значение Yes параметру Identity, который находится внизу экрана), кроме этого, сделайте поле первичным ключом, выбрав все поле и щелкнув на кнопке Set primary key.
7. Создайте еще одно поле, назовите его emails и настройте тип данных varchar с длиной поля 100 символов.
8. Сохраните таблицу под именем emails. Эта таблица (в режиме конструктора) должна выглядеть так, как показано на рис. Г.2.
9. Закройте SQL Server.

Чтобы вместо базы данных SQL Server использовать базу данных Access, можно открыть приложение Microsoft Access, создать пустую базу данных и сохранить ее в доступном месте (впоследствии необходимо будет создать для нее DSN), добавить новую таблицу с именем emails и теми же типами данных полей, которые были описаны выше. База данных Access будет доступна с помощью тех же ODBC-функций, что и база данных SQL Server.

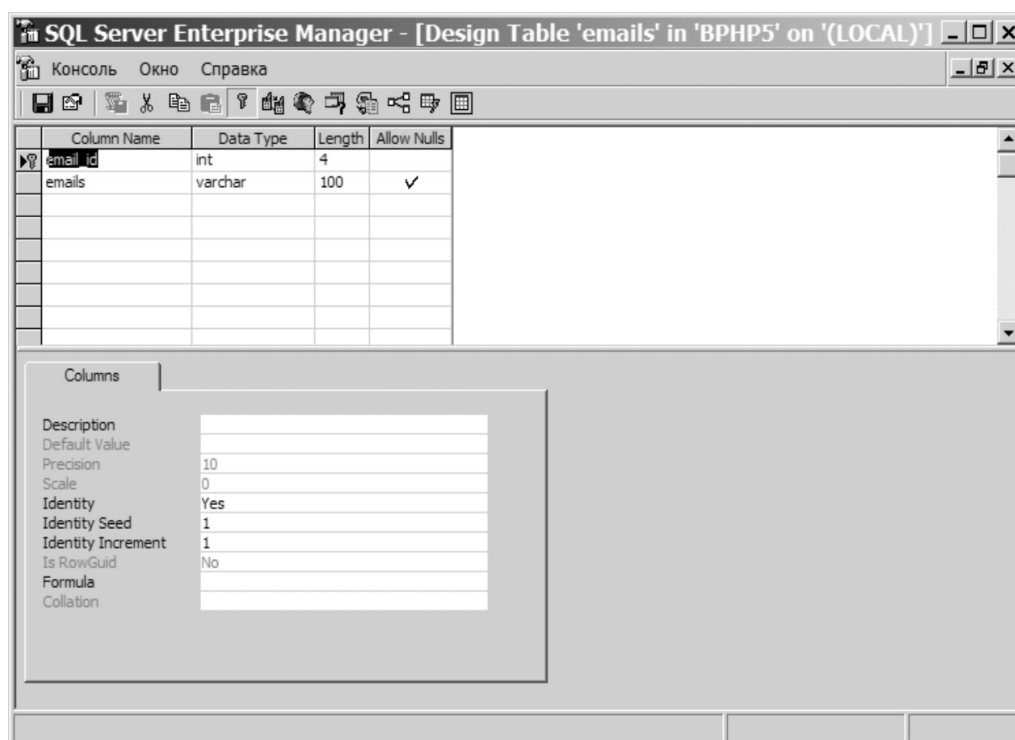


Рис. Г.2.

Создание системного DSN

Сценарий может подключиться к ODBC-базе данных, если будет доступна вся необходимая информация: имя сервера баз данных, имя базы данных (или файла базы данных), используемый драйвер, имя/пароль (если нужно) и т.д. Эта информация может присутствовать в строке подключения или храниться (в Windows-системах) в DSN (Data Source Name — имя источника данных). В приведенном ниже примере показано, как создать DSN в Windows 2000.

1. Откройте Источники данных (ODBC) (Пуск⇒Программы⇒Администрирование⇒Источники данных (ODBC)).
2. Выберите вкладку Системный DSN и нажмите кнопку Добавить. В появившемся диалоговом окне выберите соответствующий драйвер, как показано на рис. Г.3.
3. Нажмите кнопку Готово.
4. В следующем диалоговом окне введите имя DSN (оно впоследствии будет использоваться в PHP-коде для подключения к базе данных; в примере используется имя "brhp5"), краткое описание и SQL Server, к которому следует подключиться, см. рис. Г.4.

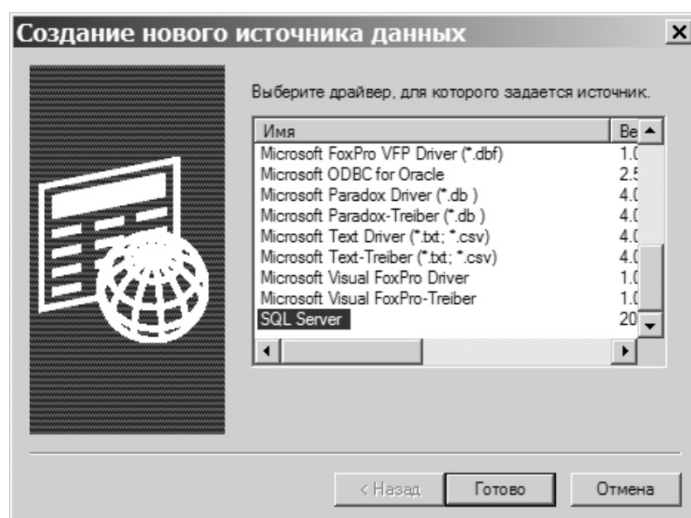


Рис. Г.3.

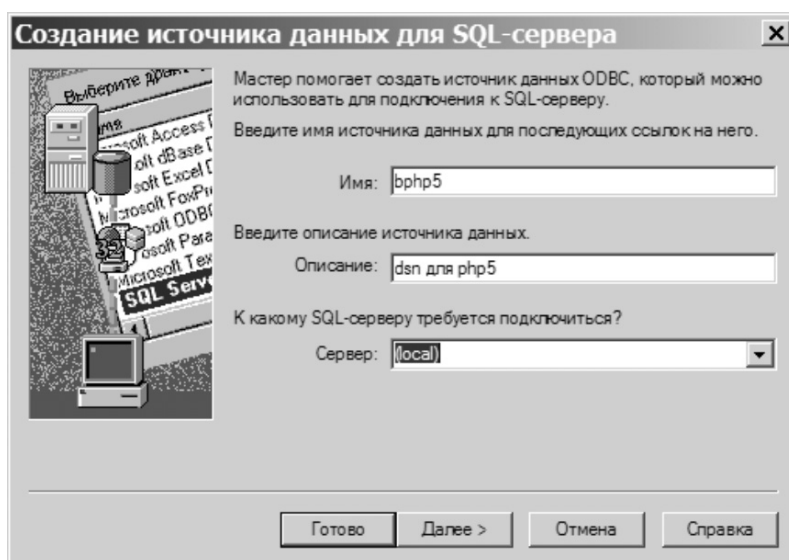


Рис. Г.4.

5. Нажмите кнопку Далее. В следующем диалоговом окне (рис. Г.5) оставьте настройки по умолчанию.

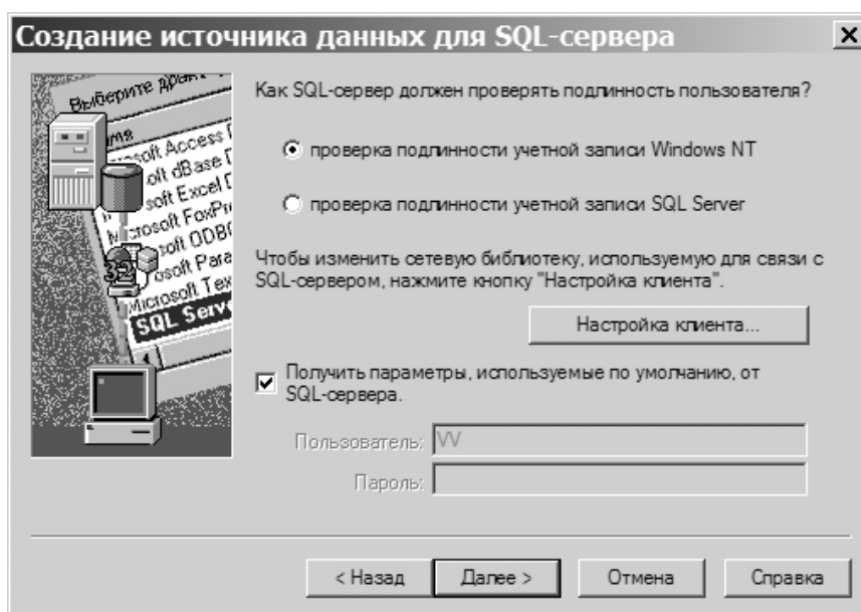


Рис. Г.5.

6. Нажмите кнопку Далее. Замените используемую по умолчанию базу данных на ВРНР5 (рис. Г.6).

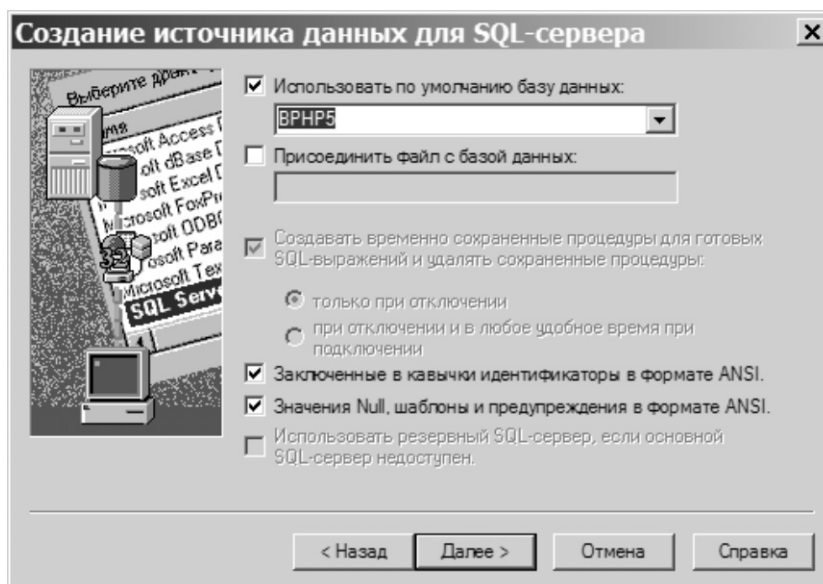


Рис. Г.6.

7. Нажмите кнопку Далее. В следующем окне оставьте настройки по умолчанию или измените язык системных сообщений SQL Server на русский (рис. Г.7).

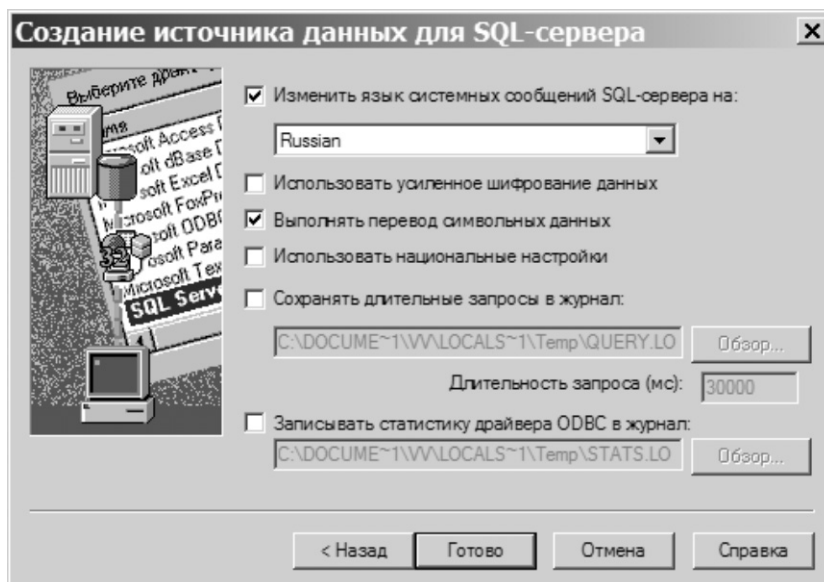


Рис. Г.7.

8. Нажмите кнопку Готово. Должно появиться окно с перечнем сделанных настроек (рис. Г.8).

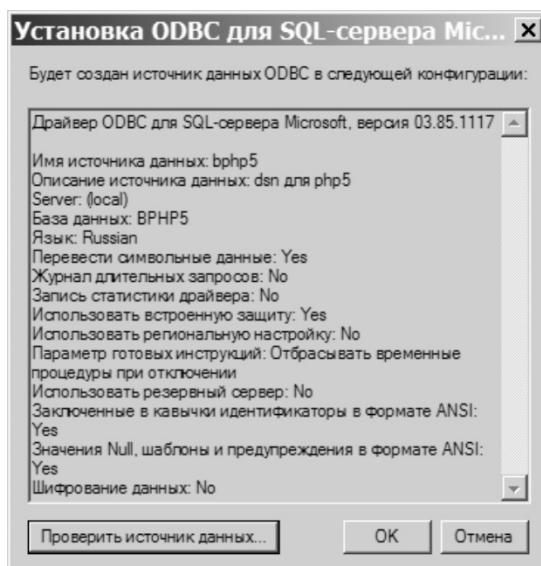


Рис. Г.8.

9. Нажмите кнопку Проверить источник данных. Должно появиться окно с результатами проверки (рис. Г.9).

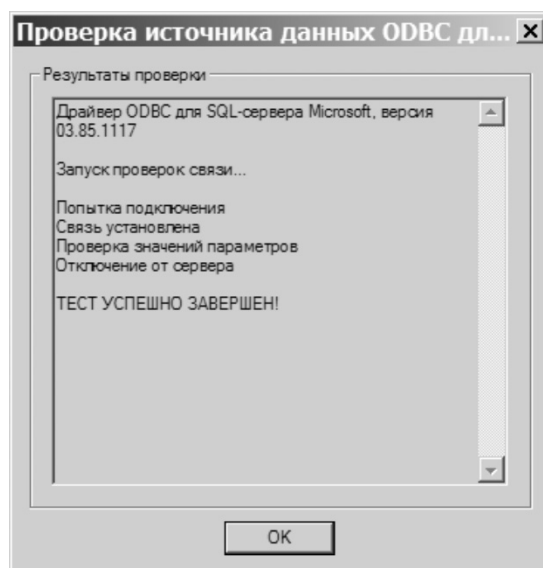


Рис. Г.9.

10. Если проверка прошла успешно, то источник данных готов к использованию.

Чтобы создать DSN для базы данных Access, необходимо выбрать драйвер Access вместо драйвера SQL Server на этапе 2. Затем появится возможность указать созданный ранее файл базы данных, а также имя и пароль, если они были установлены для этой базы данных. После этого созданный DSN можно использовать для подключения к указанной базе данных Access так же, как и для подключения к базе данных SQL Server.

Использование ODBC-функций PHP

В PHP включено более 40 ODBC-функций, позволяющих выполнять множество операций от создания соединения и выполнения запросов до определения имен полей и типов данных и возвращения ошибок. Ниже демонстрируется практическое использование некоторых ключевых ODBC-функций.

Сценарий `odbc.php` создает подключение к базе данных, выбирает записи (если они есть), а также предоставляет возможность добавлять, редактировать и удалять записи в таблице. Сценарий представлен единственной Web-страницей. Код сценария начинается с тела страницы, включающей HTML-таблицу для отображения данных:

```
<html>
<head>
  <title>PHP5 ODBC для начинающих</title>
  <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body bgcolor="#FFFFFF">
  <form action="odbc.php" method="POST">
    <table>
      <tr>
        <td>
```

Функция `odbc_connect()` записывает идентификатор соединения в переменную `$conn`. Следует отметить, что имя DSN ("bphp5") используется в качестве первого параметра функции, за которым следуют пустые строки для имени пользователя и пароля. При необходимости можно указать тип курсора. Для этого в качестве необязательного четвертого параметра необходимо передать одну из специальных констант (`SQL_CUR_DEFAULT`, `SQL_CUR_USE_DRIVER` и т.д.).

```
<?php
//создаем соединение
$conn = odbc_connect("bphp5", "", "");
```

Когда пользователь пытается добавить запись, нажимая кнопку **Добавить новый** Email-адрес, запрос, заданный в виде строковой переменной `$query`, отправляется базе данных с помощью функции `odbc_exec()`. Первым параметром этой функции является переменная `$conn`, а вторым — `$query`.

```
//добавляем запись
if(isset($_POST['add'])) {
    $query = "INSERT INTO emails (emails) values('".$_POST['email'].");";
    odbc_exec($conn,$query);
    echo "<h3>Email-адрес добавлен</h3><br><br>";
}
```

Хотя этот пример очень прост и в нем нет никакой проверки ошибок, можно использовать функцию `odbc_error()`, а затем получить сообщение об ошибке (если таковая возникла) с помощью функции `odbc_errormsg()`.

Следующий фрагмент кода предназначен для обновления данных таблицы `emails`. Запись, определенная пользователем, идентифицируется с помощью значения, вставленного в скрытое поле `$up_id` (это значение вставляется в поле в ответ на выбор пользователем e-mail-адреса из выпадающего списка, код которого формируется ниже):

```
//редактирование записи
if(isset($_POST['update'])) {
    $query = "UPDATE emails SET emails = '".$_POST['up_email']."' WHERE
        email_id=".$_POST['up_id'].";";
    odbc_exec($conn,$query);
    echo "<h3>Email-адрес изменен</h3><br><br>";
}
```

SELECT-запрос к таблице `emails` извлекает выбранную запись, а затем записывает значение `id` в скрытое поле с именем `"up_id"`:

```
//получаем запись, соответствующую выбору пользователя
if(isset($_POST['select'])) {
    $query = "SELECT * FROM emails WHERE email_id=".$_POST['select_id'].";";
    $res = odbc_exec($conn,$query);
    $rec = odbc_fetch_array($res,0);

    //вставляем значение поля email_id выбранной записи в скрытое поле
    ?>
    <input type=hidden name="up_id" value="<? echo $rec[email_id];?>">
```

Форма позволяет узнать текущее значение поля `emails` (email-адрес) выбранной записи. Кроме того, в форме имеются кнопки **Обновить** и **Удалить** для модификации или удаления записи. Приведенный ниже HTML-код используется исключительно для форматирования:

```

<table border="0" cellspacing="0" cellpadding="2">
  <tr valign="top">
    <td bgcolor="#003399">
      <table width="619" height="11" bgcolor="#FFFFFF" cellpadding="4"
        cellspacing="0">
        <tr>
          <td colspan="2" class="tablecell">
          </td>
        </tr>
        <tr>
          <td width="150" class="tablecell">Email-адрес</td>
          <td width="469" class="tablecell">
            <input type="text" name="up_email" value="<? echo
              $rec[email_address];?>" size="60">
          </td>
        </tr>
        <tr>
          <td width="150"></td>
          <td width="494">
            <table border="0" cellspacing="0" cellpadding="4">
              <tr>
                <td>
                  <input type="submit" value="Обновить" name="update">
                </td>
                <td>
                  <input type="submit" value="Удалить" name="delete">
                </td>
              </tr>
            </table>
          </td>
        </tr>
      </table>
    </td>
  </tr>
</table>
<table border="0" cellspacing="0" cellpadding="4" align="center">
  <tr>
    <td height="10"></td>
  </tr>
  <tr>
    <td align="center">
      <?
    </td>
  </tr>
</table>

```

Приведенный ниже PHP-код выполняет запрос на удаление записи. Удаляемая запись указывается таким же способом, как и запись для редактирования (в скрытом поле "up_id" указывается идентификатор):

```

//удаляем запись
if(isset($_POST['delete'])) {
    $query = "DELETE FROM emails WHERE email_id=$_POST[up_id]";
    odbc_exec($conn,$query);
    echo "<h3>Email-адрес удален</h3>";
}

```

Ниже приведен выполняемый по умолчанию код, который отображает адреса из таблицы emails при первоначальном открытии страницы. Он выполняется каждый раз при выводе страницы и показывает текущую запись в таблице. Функция odbc_exec() присваивает переменной \$res результат выполнения запроса; функция odbc_fetch_row() выбирает строку из \$res; функция odbc_result() определяет значение указанного поля (первым параметром функции odbc_result() является переменная \$res, а имя поля указывается в качестве второго параметра). Цикл while() используется для обработки

выбранных строк, и так как он выполняется внутри HTML-дескриптора <select>, здесь также выводятся HTML-теги <option> для всех имеющихся e-mail-адресов:

```
//извлекаем все записи и помещаем их в выпадающий список
$query = "SELECT * FROM emails ORDER BY email_id;";
$res = odbc_exec($conn,$query);
?>
<select name="select_id">
<?
while (odbc_fetch_row($res)) {
    $email_id = odbc_result($res,"email_id");
    $email = odbc_result($res,"email");
?>
    <option value="<? echo $email_id; ?>">
        <? echo "Идентификатор Email-адреса:" . $email_id . ":" . " Email-адрес: "
        . $email; ?>
    </option>
<?
} ?>
</select>
        <input type="submit" name="select" value="Выбрать Email-адрес">
    </td>
</tr>
</table>
```

Следующий HTML-код выводит форму для добавления новых e-mail-адресов:

```
<table border="0" cellspacing="0" cellpadding="2"
align="center">
<tr>
<td bgcolor="#003399">
<table width="619" cellpadding="4" cellspacing="0"
align="center"
        bgcolor="#FFFFFF">
<tr>
<td colspan="2" class="tablecell"> </td>
</tr>
<tr>
<td width="150" height="29">Email-адрес</td>
<td width="494" height="29">
<input type="text" name="email" size="60">
</td>
</tr>
<tr>
<td width="130"></td>
<td width="477">
<input type="submit" value="Добавить новый Email-адрес"
name="add">
</td>
</tr>
</table>
</td>
</tr>
</table>
</td>
</tr>
</table>
</form>
</body>
</html>
```

Откройте страницу `odbc.php` в браузере. Она должна выглядеть аналогично рис. Г.10. Добавьте несколько e-mail-адресов.

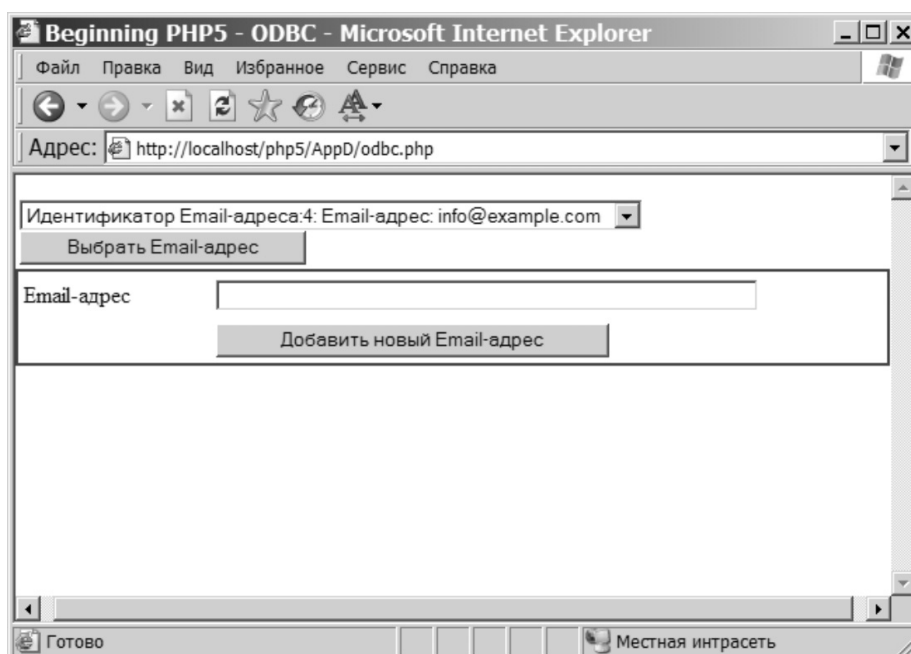


Рис. Г.10.

Можно выбрать адреса, добавленные в базу, и изменить или удалить их (рис. Г.11).

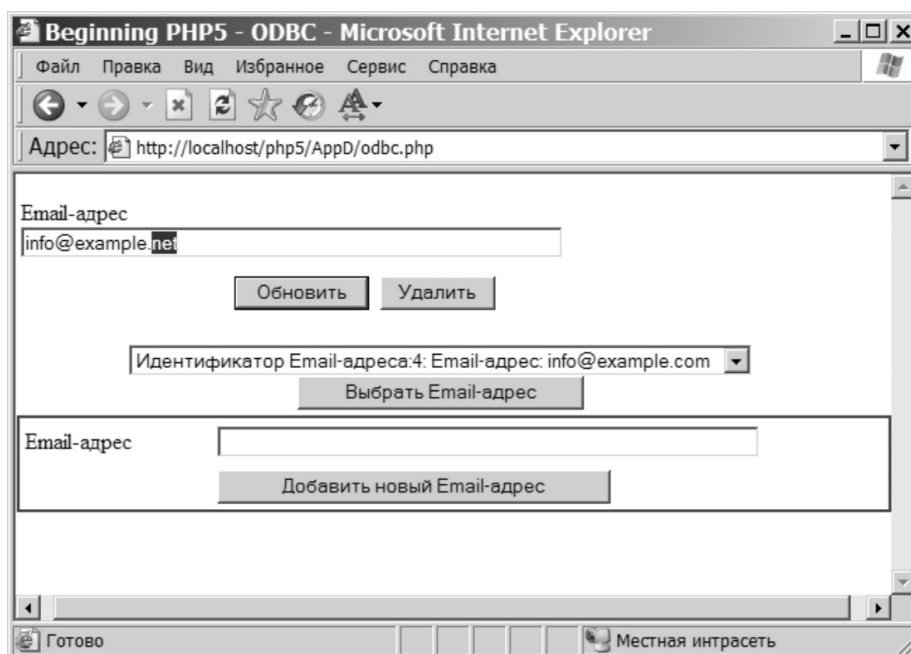


Рис. Г.11.



CLI-интерфейс PHP

Одним из самых приятных усовершенствований, внесенных в PHP за последние годы, является введение CLI-интерфейса (Command Line Interpreter — интерпретатор командной строки). CLI позволяет использовать PHP-сценарии в не-Web-среде, например, для решения задач системного администрирования или даже в качестве встраиваемого компонента для другого приложения. Это дает PHP возможность решать широкий круг задач, для которых обычно использовались такие средства, как Perl, Bash или DOS. Приложение Д знакомит читателя с CLI-интерфейсом и демонстрирует возможные варианты его использования.

Поскольку CLI-интерфейс в PHP представляет собой новый виток развития языка и заимствует некоторые более ранние технологии, стоит рассмотреть некоторые исторические вехи этого развития.

Начало

В течение очень долгого времени пользователям Unix и DOS приходилось использовать понятие сценариев оболочки — как правило, очень простых программ для выполнения рутинных задач системного администрирования, написанных на одном из нескольких доступных языков сценариев. Трудно представить себе работу на Linux-машине без некоторых shell-сценариев; многие стандартные задачи администрирования, например, ротация системных журналов и резервное копирование, часто решаются с помощью этих на первый взгляд простых программ. Вместе с тем эти сценарии сами по себе без интерпретатора, который выполняет содержащиеся в них инструкции, не представляют особого интереса.

Интерпретатор, как правило, представлен в одной из двух форм: интерактивная оболочка или интерпретируемый язык. Примерами первой формы является печально известный `command.com` в MS-DOS и множество оболочек, доступных большинству Linux/Unix-пользователей — Bash, Csh, Zsh и масса других. Примеры второй формы — такие широко известные языки программирования, как Perl, Python, Tcl, а с недавнего времени PHP. Оболочки обычно имеют несколько простых встроенных кон-

струкций наподобие управляющих и условных операторов. Кроме того, они часто занимают у системы средства для реализации большинства остальных возможностей. Однако для решения сложных и трудоемких задач необходимо использовать более гибкие инструменты; в качестве таких инструментов оказываются полезными различные интерпретаторы. Интерпретаторы по сравнению с оболочками обычно представляют более гибкую типизацию, более широкий диапазон стандартных конструкций для программирования и более развитые функции обеспечения безопасности.

Отвлечемся на короткое время от оболочек и вспомним, как возникла Web-среда. Первые World Wide Web-страницы представляли собой, главным образом, статические документы, которые http-демон доставлял пользователям в ответ на соответствующие запросы. Любые изменения приходилось вносить непосредственно в документы и копировать эти документы туда, где пользователи ожидали получить к ним доступ. Web-страницы были неспособны проделывать обычные в наши дни трюки, например, они не могли запоминать и идентифицировать пользователей и выдавать различное содержимое на основании данных о пользовательских предпочтениях. Читателям, для которых эта книга является первым экскурсом в серверные языки сценариев, может показаться, что PHP представляет собой решение этой проблемы. Однако на самом деле PHP и его функциональные аналоги, такие как Active Server Pages и Java Server Pages, работают на поверхности другой более ранней технологии, которая называется Common Gateway Interface (интерфейс общего шлюза), или CGI.

С помощью CGI Web-сервер открывает интерфейс для внешних программ, обычно написанных на одном из упомянутых выше интерпретируемых языков. Сервер передает такой программе инструкции для генерирования страницы, которая затем возвращается Web-серверу, а впоследствии в браузер пользователя. Здесь обе нити истории пересекаются.

Еще до возникновения идеи серверных модулей технология CGI в течение многих лет неплохо служила своей цели. PHP — один из таких модулей; Perl тоже может работать как модуль Web-сервера Apache. Подобное развитие технологий было продиктовано некоторыми проблемами CGI, например, трудностями, связанными с недостаточной производительностью и особенно с безопасностью.

CGI и сейчас довольно широко используется, даже во многих PHP-инсталляциях, несмотря на то, что это не самое оптимальное решение. Пользователи до сих пор иногда применяют инсталляции в CGI-стиле — они просто не передают конфигурационному сценарию специфических для Web-сервера флагов. Идея использования PHP для системного администрирования не нова. Некоторые системные администраторы в своей работе часто используют PHP-интерпретатор.

Но это применение даже в очень простых сценариях теряется из виду на фоне Web-ориентированного CGI-модуля PHP. В PHP версии 4.2.x в качестве экспериментальной функции был представлен CLI-интерфейс PHP; к версии 4.3.x, а затем и в серии 5.0 установка CLI стала использоваться по умолчанию. Если вы абсолютно уверены, что CLI не должен присутствовать в используемой системе, можете передать конфигурационному сценарию флаг `--disable-cli`.

Некоторые важные моменты

Для опытных пользователей shell и системных администраторов, которые работают с Perl или Python, будет полезно ознакомиться с некоторыми ключевыми отличиями в работе PHP CLI.

- ❑ PHP CLI не изменяет рабочий каталог на каталог, в котором находится исполняемый сценарий. Чтобы добиться этого, необходимо явно указать полные рабочие пути либо использовать константы или переменные.
- ❑ Вызов PHP CLI без параметров не приводит к запуску интерактивной оболочки, как это происходит в Python или Tcl.
- ❑ В отличие от Perl, код внутри CLI-сценария в PHP все равно должен находиться между тегами `<?php` и `?>`, даже если интерпретатор определен с помощью обычного синтаксиса `#!/usr/local/bin/php`. Строки, не заключенные в теги сценария, возвращаются в стандартный вывод (`stdout`) неизмененными.

PHP CLI принимает несколько ключей командной строки, которые описаны в приведенной ниже таблице.

Ключ	Описание
-s	Включает цветную подсветку исходного кода сценария. Пример использования: <code>php -s myscript</code> . Эта команда возвращает подсвеченную цветную версию сценария, используя HTML-тег <code>font</code> . В консоли это не особенно полезно, но если передать этот вывод в браузер, то он красиво отобразит код
-w	Отображает исходный код сценария без комментариев и пробельных символов
-f	Используется для указания файла, передаваемого интерпретатору, например, <code>php -f myscript</code> . Этот флаг не обязательный и команда <code>php myscript</code> приводит к идентичному результату. Данный флаг может оказаться полезным для повышения четкости в <code>crontab</code> или сценариях, вызывающих другие сценарии
-v	Распечатывает номер версии и завершает работу
-c	Указывает .ini-файл, отличный от используемого по умолчанию
-a	Запускает интерпретатор в интерактивном режиме
-d	Определяет значение переменной в <code>php.ini</code> , например, <code>php -d safe_mode=off</code> . Это значение сохраняется только в текущем экземпляре интерпретатора
-e	Генерирует расширенную информацию для отладчика и профайлера кода
-z	Загружает указанное Zend-расширение. Это расширение будет работать только в текущем экземпляре интерпретатора
-l	Выполняется только проверка синтаксиса. Это может оказаться очень полезным для не деструктивного тестирования сценариев, данный ключ иногда используется в Web-сценариях
-m	Возвращает перечень модулей, скомпилированных в PHP. При этом имена динамически загружаемых модулей не возвращаются
-i	Возвращает общую конфигурационную информацию PHP. Использование этого флага эквивалентно использованию функции <code>phpinfo()</code> без HTML-форматирования. Можно передать вывод программе страничного просмотра, например, <code>more</code> или <code>less</code>
-r	Выполняет код, введенный в командную строку без тегов сценария. Код необходимо заключить в одинарные кавычки
-h	Возвращает список ключей командной строки и параметров форматирования

Обработка параметров командной строки

PHP-сценарии в Web могут легко получить и обработать аргументы из различных источников, таких как cookie-файлы, Web-формы и значения полей баз данных. Значения полей баз данных также легко получить и в сценариях, использующих CLI, однако в этом случае отсутствует браузер, из которого можно было бы получить остальные аргументы. Тем не менее, во время выполнения сценария можно передавать ему значения или аргументы. В PHP CLI для передачи данных, введенных в командной строке, используются переменные `$argc` и `$argv`. Эти переменные известны программистам на C/C++. В PHP `$argc` — скалярная переменная, содержащая количество аргументов, а `$argv` — глобальный массив (с отсчетом элементов с нуля), содержащий сами аргументы в том порядке, в котором они были считаны.

Аргументы в PHP CLI

Следующий сценарий принимает несколько аргументов и выводит их на экран. Создайте файл с именем `myscript` и введите в него код, как показано ниже (Windows-пользователи могут пропустить первую строку):

```
#!/usr/local/bin/php
<?php
print "Получено $argc аргументов\n";
for ($i = 0; $i <= $argc - 1; $i++) {
    print "Это аргумент $i и его значение равно $argv[$i]\n";
}
?>
```

Сохраните файл, а затем запустите его из командной строки. Явно указывать интерпретатор нет необходимости, он уже задан в первой строке сценария. Этот метод используется во всех остальных примерах данного приложения. Передайте сценарию несколько аргументов и обратите внимание на вывод. Для этого в Linux введите такую команду:

```
> ./myscript Dan David Alan Steven Clark Wankyu
```

в Windows команда должна выглядеть так:

```
> php myscript Dan David Alan Steven Clark Wankyu
```

Результат будет выглядеть примерно так:

```
Получено 7 аргументов
Это аргумент 0 и его значение равно ./myscript
Это аргумент 1 и его значение равно Dan
Это аргумент 2 и его значение равно David
Это аргумент 3 и его значение равно Alan
Это аргумент 4 и его значение равно Steven
Это аргумент 5 и его значение равно Clark
Это аргумент 6 и его значение равно Wankyu
```

Первый аргумент — имя самого сценария, несмотря на то, что оно не было передано в командной строке в списке аргументов. Это удобно, когда нужно, чтобы сценарий обращался к самому себе, поскольку в CLI переменная `$_SERVER[SCRIPT_NAME]` недоступна. В результате первый индекс из всех реальных аргументов равен 1, хотя отсчет элементов начинается с 0.

Зная, как читать аргументы из командной строки, и понимая, как это происходит, можно создать более сложное и полезное приложение.

Запуск shell-команды

Команда `users` в Linux или в любой другой Unix-системе предоставляет системному администратору удобный способ узнать, кто в текущий момент зарегистрирован в системе. Однако недостаток этой команды заключается в том, что она возвращает неупорядоченный список разделенных пробелами имен, который трудно читать в бесцветной консоли. Для быстрого поиска определенного пользователя необходимо более гибкое приложение. Приведенный ниже короткий сценарий принимает в качестве аргумента имя пользователя, ищет его в списке активных в текущий момент пользователей, а затем возвращает сообщение о состоянии пользователя в системе. Создайте файл `check_user` и поместите в него следующий код:

```
#!/usr/local/bin/php
<?php
$command = "users";
$logged_in = explode(' ', shell_exec($command));
for ($i = 0; $i < count($logged_in); $i++) {
    if ($argv[1] == trim($logged_in[$i])) {
        $found_user = 1;
    }
}
if ($found_user) {
    print "Пользователь $argv[1] в настоящий момент находится в системе\n";
} else {
    print "Пользователь $argv[1] в настоящий момент в системе НЕ
        зарегистрирован\n";
}
?>
```

Запустите сценарий с помощью следующей команды:

```
> ./check_user someuser
```

в результате должен появиться подобный вывод:

```
Пользователь someuser в настоящий момент находится в системе
```

В этом примере следует отметить использование PHP-функции `trim()`. Общеизвестно, что утилиты Unix для форматирования своего вывода используют пробелы, особенно в консоли. Хотя разделяющие пробелы были удалены путем передачи пробела функции `explode()` в качестве разделителя, команда `users` в конце своего вывода вставляет жесткий перевод строки, который нужно убрать, если он встречается в последнем имени в списке. Функция `trim()` удаляет только пробельные символы (пробелы, символы табуляции и перевода строки), оставляя полезную часть вывода нетронутой.

Windows-пользователи также могут использовать DOS-команды. Создайте новый файл `directory` со следующим кодом:

```
<?php
$dir = shell_exec("dir");
echo "$dir";
?>
```

Запустите сценарий из командной строки с помощью следующей команды:

```
> php directory
```

На экран будет выведено содержимое текущего каталога.

Автоматизация PHP CLI

Выше было показано, как можно при необходимости запускать CLI-сценарии из командной строки, однако запуск таких сценариев можно автоматизировать. Например, можно отправлять e-mail-напоминания пользователям системы (такая функциональность является частью проекта PHP WebCalendar). Можно также в течение дня отслеживать пользователей в системе. В этом разделе описанный ранее сценарий будет расширен для отслеживания пользователей в гипотетической системе. Каждые полчаса сценарий будет просматривать список пользователей и отправлять системному администратору отчет по e-mail. Стандартным средством автоматизации задач в Windows является встроенный планировщик, весьма понятный и простой в использовании инструмент. В Linux таким средством является Cron, более сложная программа. В этом разделе рассматривается использование именно этой утилиты.

Windows-пример будет представлен далее. Пока начнем с первой версии сценария `check_user`. Чтобы включить в сценарий описанную выше функциональность, необходимо внести несколько изменений:

```
#!/usr/local/bin/php
<?php
$first_command = "users";
$second_command = "date";
$third_command = "hostname";
$logged_in = explode(' ', shell_exec($first_command));
for ($i = 0; $i <= count($logged_in); $i++) {
    $users .= trim($logged_in[$i]) . "\n";
}
$dt = shell_exec($second_command);
$subject = "Получасовой отчет о зарегистрированных пользователях.";
$body .= $subject . "\n\n";
$body .= trim($dt) . " "; В системе " . shell_exec($third_command) .
    "зарегистрированы следующие пользователи:\r\n";
$body .= $users;
mail("root@localhost", $subject, $body, "From:root@localhost\r\n");
?>
```

В этом сценарии, по сути, нет ничего нового. Все задействованные здесь функции уже рассматривались ранее в этой книге. Системные команды специфичны для Unix/Linux, но их имена ясно свидетельствуют об их назначении. Сначала данные, возвращаемые командой `users`, для более удобного чтения форматируются в виде списка в один столбец с одним именем в каждой строке. Команда `date` используется фактически для датирования сообщения. Затем вызывается команда `hostname`, которая может оказаться полезной, если запускать этот сценарий на разных машинах.

Сначала можно (и нужно) протестировать этот сценарий из командной строки. Сценарий должен отправлять e-mail-сообщение примерно следующего содержания:

```
Получасовой отчет о зарегистрированных пользователях.

Mon Jan 26 12:42:16 EST 2004; В системе ds3 зарегистрированы следующие пользователи:

akent
cmorgan
dmercer
dsquier
snowicki
wchoi
```

Это более или менее то, что и ожидалось от сценария. Можно воспользоваться утилитой Cron для автоматического запуска этого сценария через каждые полчаса; но прежде следует изучить работу самой утилиты Cron. Программа Cron состоит из двух основных частей. Реальную работу выполняет программа `cronpd`, которую чаще называют Cron-демон. В Unix-подобных системах демон представляет собой программу, которая работает постоянно в фоновом режиме, как правило, ожидая некоторых инструкций. Web-сервер, FTP-сервер и почтовый сервер — все это примеры демонов.

Cron-демон обычно запускается после всех остальных серверов в системе. Инструкции он получает от второй части утилиты, файла `crontab`, который считывается во время запуска программы. `crontab` представляет собой простой текстовый файл, в котором содержится перечень заданий и указывается время, когда их нужно выполнить. Файл `crontab` есть у каждого пользователя в системе, хотя он может быть пустым. Чтобы просмотреть все задания в `crontab`, можно ввести команду `crontab -l`, которая выводит на консоль список Cron-заданий. Этот список может выглядеть так:

```
# Run hourly cron jobs at 47 minutes after the hour:
47 * * * * /usr/bin/run-parts /etc/cron.hourly 1> /dev/null
#
# Run daily cron jobs at 4:40 every day:
40 4 * * * /usr/bin/run-parts /etc/cron.daily 1> /dev/null
#
# Run weekly cron jobs at 4:30 on the first day of the week:
30 4 * * 0 /usr/bin/run-parts /etc/cron.weekly 1> /dev/null
#
# Run monthly cron jobs at 4:20 on the first day of the month:
20 4 1 * * /usr/bin/run-parts /etc/cron.monthly 1> /dev/null
```

Первые пять полей `crontab`-записи описывают время запуска команды, которая задается в шестом поле, продолжающемся до конца строки. Символ `#` предшествует комментариям. Порядок полей времени следующий: минута, час, день, месяц и день недели. Следующий пример дает некоторое представление о том, как эти поля используются, хотя, возможно, сразу не понятно, что назначение команды заключается в запуске сценария каждые полчаса. Фактически строка для описания этого Cron-задания выглядит так:

```
*/30 * * * * user_report 1> /dev/null
```

По умолчанию демон Cron отправляет по e-mail отчет после успешного выполнения задания, что может оказаться излишним. Команда `1>/dev/null` служит для того, чтобы просто перенаправлять стандартный вывод Cron в Unix-корзину.

Администратору может понадобиться в течение дня отслеживать все изменения файлов в определенном каталоге. Например, Windows-пользователи с помощью следующего кода могут отправлять на определенный e-mail-адрес листинг текущего каталога:

```
<?php
$dir = shell_exec("dir");
$subject = "Ежедневный отчет об изменениях файлов.";
$body = "В текущем каталоге находятся следующие файлы: \n\n";
$body .= $dir;
mail("root@localhost", $subject, $body, "From:root@localhost\r\n");
?>
```

Для автоматического выполнения этого сценария в Windows зайдите в панель управления и выберите пункт Назначенные задания ⇒ Добавить задание. Мастер

поможет сделать необходимые настройки. В поле Выполнить необходимо ввести следующую команду:

```
C:\PHP5\php.exe directory
```

Можно также нажать кнопку Обзор, найти исполняемый файл PHP на второй странице мастера и добавить имя сценария, который необходимо запустить.

В зависимости от настроек по расписанию будут приходить письма с отчетами о содержимом каталога.

Интерактивность средствами PHP CLI

Интерактивность командной строки PHP развита не хуже, чем интерактивность Web-сценариев. Можно легко создавать CLI-сценарии, которые для выполнения больших задач с использованием потоков принимают последовательные биты пользовательского ввода. PHP предоставляет доступ к трем потокам, которые имитируют соответствующую функциональность Unix. Эти потоки описываются в приведенной ниже таблице.

<i>Поток</i>	<i>Чему соответствует</i>	<i>Доступ в PHP</i>
Стандартный ввод	Клавиатуре	php://stdin
Стандартный вывод	Консоли (экрану монитора)	php://stdout
Стандартный поток ошибок	Консоли или системному журналу	php://stderr

Рассмотрим простой сценарий, который собирает пользовательскую информацию и создает файл подписи для использования в e-mail-клиенте. Создайте файл sigfile и введите в него следующий код:

```
#!/usr/local/bin/php
<?php
$stdin = fopen('php://stdin', 'r');
echo "PHP-генератор файла подписей.\n";
echo "Введите свое полное имя: ";
$name = trim(fgets($stdin,100));
echo "Введите свой адрес (улицу): ";
$address = trim(fgets($stdin,100));
echo "Введите название города: ";
$city = trim(fgets($stdin,100));
echo "Введите название штата: ";
$state = trim(fgets($stdin,100));
echo "Введите свой почтовый индекс: ";
$zip = trim(fgets($stdin,100));
echo "Введите номер телефона: ";
$phone = trim(fgets($stdin,100));
echo "Введите e-mail-адрес: ";
$email = trim(fgets($stdin,100));
echo "Как Вы хотите обозначить эту подпись: ";
$sig = trim(fgets($stdin,100));
fclose($stdin);
$data = "$name\n$address\n$city, $state $zip\n$phone\n$email";
shell_exec("touch $sig");
if (!$sigfile = fopen($sig, 'w')) {
    print "Невозможно открыть файл для записи\n";
} else {
    if (!fwrite($sigfile,$data)) {
        print "\n\nНе удалось записать данные\n";
    }
}
```

```
    } else {  
        print "\n\nПодпись готова\n";  
    }  
    fclose($sigfile);  
}  
?>
```

Сначала открывается поток стандартного ввода, который остается открытым вплоть до получения последнего ответа пользователя. Затем из каждого последующего пользовательского ввода извлекается первых 100 символов. Эти символы передаются функции `trim()`, так как, к сожалению, нажатие пользователем клавиши `<Enter>` передает сценарию жесткий перевод строки. После получения от пользователя всех данных поток ввода закрывается и открывается файл с именем, которое выбрал пользователь. Составленная из ответов пользователя подпись записывается в этот файл, после чего он закрывается.

Windows-пользователи могут применить тот же сценарий, заменив строку

```
shell_exec("touch $sig");
```

на

```
'copy con $sig ^Z \n';
```

Сохраните файл. Сценарий генерирует примерно следующий вывод:

```
> php sigfile  
PHP-генератор файла подписей.  
Введите свое полное имя: David Mercer  
Введите свой адрес (улицу): 148 Mystreet  
Введите название города: Cape Town  
Введите название штата: WP  
Введите свой почтовый индекс: 8001  
Введите номер телефона: 021 555-1234  
Введите e-mail-адрес: davidm@doggiesrugby.co.za  
Как Вы хотите обозначить эту подпись: MySig
```

```
Подпись готова
```

Заключение

Очевидно, что PHP CLI обладает той же гибкостью и эффективностью, что и PHP в Web-среде. Можно с уверенностью утверждать, что потенциальные варианты применения ограничены только мощностью самого PHP, что, вообще говоря, нельзя назвать серьезным ограничением. Интерактивность, обеспечиваемая потоками, является особенно полезной. Системные администраторы могут использовать PHP CLI для решения своих повседневных задач, например, для резервного копирования, рассылки системных уведомлений и любых других административных функций. Выполняемые по расписанию CLI-сценарии совместно с Web-приложениями могут реализовывать такие возможности, о которых раньше разработчики программ на PHP могли только мечтать.



Конфигурация PHP5

Важную роль в работе PHP играет конфигурационный файл, который называется `php.ini`. При запуске PHP считывает конфигурационный файл и устанавливает заданные в нем параметры. Если PHP установлен в виде серверного модуля, то файл считывается один раз; если PHP работает как CGI, то файл считывается для каждого запускаемого экземпляра PHP.

В этом приложении описаны все конфигурационные разделы и параметры `php.ini`. Многие из этих параметров уже обсуждались на страницах книги, поэтому данное приложение не представляет собой полностью новый материал, а скорее является справочником.

С дистрибутивом PHP поставляется две версии — `php.ini`: `php.ini-recommended` и `php.ini-dist`. Согласно рекомендациям создателей PHP `dist`-файл подходит для разработки, тогда как для реальной среды лучше всего использовать файл `recommended`. В этом файле имеется несколько параметров, которые предназначены для повышения производительности, но могут нарушать работу старых сценариев, особенно это касается настроек `register_globals`, которые отключены (и не будут использоваться в будущих версиях PHP). В этом приложении описывается содержимое файла `php.ini-dist`.

Описание файла `php.ini-dist`

Как уже было сказано, файл `php.ini-dist` подходит для разработки сценариев, но не рекомендуется для работающих сайтов. В начале книги указывалось, где расположен этот файл и как PHP находит его во время запуска, а также описывалась организация `php.ini` (с примерами многих инструкций и допустимых для них значений).

```
;;;;;;;;;;  
; ВНИМАНИЕ ;  
;;;;;;;;;;  
; Это стандартный файл для новых инсталляций PHP.  
; По умолчанию PHP устанавливается с конфигурацией подходящей для
```


814 Приложение E

```
; целей разработки, а *НЕ* для использования на работающих сайтах.
; По некоторым соображениям, связанным с безопасностью
; перед размещением сайта в сети, рекомендуется изучить файл
; php.ini-recommended и страницу
; http://php.net/manual/en/security.php.
;
; Об этом файле ;
;
; Этот файл содержит большинство установок PHP. Чтобы PHP смог
; его прочитать, он должен называться 'php.ini'. PHP ищет файл в
; текущем каталоге, в случае неудачи — в каталоге, указанном в
; переменной окружения PHPRC, и, наконец, в каталоге, заданном при
; компиляции PHP (именно в таком порядке). В Windows путь,
; указанный при компиляции, соответствует каталогу Windows. Папка,
; в которой будет производиться поиск файла 'php.ini', может быть
; также переименована с использованием ключа -с командной строки.
;
; Синтаксис файла крайне прост. Пробельные символы (пробелы,
; переводы строк, символы табуляции и т. д.) и строки,
; начинающиеся с точки с запятой (;), игнорируются (как вы,
; наверное, уже догадались). Заголовки разделов (например, [Foo])
; также пропускаются, хотя, возможно, будут учитываться в будущем.

; Директивы задаются следующим образом:
; directive=value
; Имена директив *чувствительны к регистру символов*, т.е. foo=bar
; не то же самое, что FOO=bar.

; Значение (value) может быть строкой, числом, константой PHP
; (например, E_ALL или M_PI), одной из INI-констант (On, Off,
; True, False, Yes, No или None), выражением (например, E_ALL &
; ~E_NOTICE), а также строкой в кавычках ("foo").

; В выражениях могут использоваться только побитовые операторы, а
; также скобки:
; |      поразрядное ИЛИ (OR)
; &      поразрядное И (AND)
; ~      поразрядное НЕ (NOT)
; !      логическое отрицание (NOT)
; В качестве логических флагов со значением "истина" могут быть
; использованы значения 1, On, True или Yes. Значение "ложь" дают
; 0, Off, False и No.

; Пустая строка может быть задана, если после знака равенства не
; указать ничего или указать ключевое слово None:
; foo =                ; устанавливаем foo равным пустой строке
; foo=none             ; аналогично
; foo="none"           ; устанавливаем foo равным строке 'none'
;
; Если в качестве значения директивы используется константа,
; определяемая в динамически загружаемом расширении
;
; (PHP-расширении или Zend-расширении), то это значение должно
; быть указано *после* строки, которая загружает расширение.
;
; Все значения в файле php.ini-dist соответствуют встроенным
; значениям по умолчанию (т.е. если php.ini не задействуется, или
; же из него удалены некоторые строки, то будут установлены
; значения по умолчанию).
```

Настройки языка представляют собой конфигурационные параметры, которые непосредственно влияют на режим работы PHP. Например, значение Оп директивы `short_open_tag` позволяет вместо тегов `<?php` и `?>` использовать для ограничения PHP-кода теги `<? и ?>` (хотя рекомендуется все-таки использовать длинные теги).

Пример настройки безопасного режима: комбинация параметров `safe_mode = on` и `safe_mode_exec_dir = my_safe_mode_exec_dir` (имя каталога вымышленное). При включенном безопасном режиме, с помощью РНР-команды `exec` можно запускать только те исполняемые файлы, которые находятся в каталоге, указанном директивой `safe_mode_exec_dir` (если он существует).

Таким образом, параметр `safe_mode` и другие связанные с ним настройки позволяют устанавливать параметры безопасности по-разному в зависимости от того, какие функции должно выполнять приложение. Чтобы защитить систему и пользовательские данные, необходимо тщательно изучить руководства по обеспечению безопасности.

```

;::::::::::::::::::
; Настройки языка ;
;::::::::::::::::::

; Разрешает работу PHP для сервера Apache.
engine=On

; Разрешает использовать короткий тэг <?. Иначе будут
; распознаваться только тэги <?php и <script>.
; ПРИМЕЧАНИЕ: использования коротких тегов следует избегать при
; разработке приложений или библиотек, которые задуманы для
; распространения или будут использоваться на чужих PHP-серверах,
; поскольку короткие теги на этих серверах могут не поддерживаться.
; Для создания переносимого распространяемого кода короткие теги
; использовать не следует
short_open_tag=On

; Позволяет использовать тэги в стиле ASP <% %>.
asp_tags=Off

```

816 Приложение Е

```
; Число значащих цифр после запятой, которые отображаются для
; чисел с плавающей точкой.
precision=12
; Признак коррекции дат (проблема 2000 года, которая может
; вызвать непонимание со стороны не рассчитывающих на это
; браузеров)
y2k_compliance=Off

; Использование буферизации вывода. Позволяет посылать заголовки
; (включая cookie) после вывода текста. Это происходит ценой
; незначительного замедления вывода.
; Можно разрешить буферизацию во время выполнения сценария путем
; вызова функций буферизации, или же включить ее для всех файлов
; с помощью следующей директивы. Чтобы ограничить размер буфера,
; в качестве значения этой директивы можно использовать
; количество байтов вместо слова 'On'
; (например, output_buffering=4096).
output_buffering=Off

; Весь вывод сценария можно передать какой-либо функции.
; Например, если установить параметр output_handler равным
; "mb_output_handler", то кодировка символов будет прозрачно
; преобразована в заданную кодировку.
; Установка любого обработчика вывода автоматически включает
; буферизацию вывода.
; Примечание: при написании переносимых сценариев не следует
; полагаться на эту директиву. Вместо этого необходимо
; явно задать обработчик буфера с помощью функции
; ob_start().
; Использование этой директивы может вызвать проблемы,
; если не известно, что делает сценарий.
; Примечание: использовать одновременно оба значения
; "mb_output_handler" и "ob_iconv_handler" нельзя.
; Также нельзя одновременно использовать значения
; "ob_gzhandler" и "zlib.output_compression".
; output_handler =

; Прозрачное сжатие вывода с помощью библиотеки zlib
; Значениями этого параметра могут быть 'off', 'on' или
; определенный размер буфера, используемого для сжатия
; (по умолчанию 4 Кб)
; Примечание: размер результирующего блока может отличаться из-за
; сжатия. PHP в результате сжатия выводит блоки по
; несколько сотен байт. Чтобы использовать больший
; размер для повышения производительности, необходимо
; в дополнение к этому включить буферизацию вывода.
; Примечание: вместо стандартной директивы output_handler
; необходимо использовать zlib.output_handler
; иначе вывод будет поврежден.
zlib.output_compression = Off

; Если активизировано zlib-сжатие, то задавать дополнительные
; обработчики буфера нельзя. Этот параметр делает то же, что и
; output_handler, но в другом порядке.
;zlib.output_handler =

; Директива неявной отсылки говорит PHP о том, что выводимые
; данные нужно автоматически передавать браузеру после вывода
; каждого блока данных. Ее действие эквивалентно вызовам функции
; flush() после каждого использования print() или echo() и после
; каждого HTML-блока. Включение этой директивы серьезно замедляет
; работу, поэтому ее рекомендуется применять лишь в отладочных
; целях.
implicit_flush=Off
```

```
; Функция обратного вызова будет вызываться (с именем
; неопределенного класса в качестве параметра), если
; десериализатор обнаружит неопределенный класс, объект которого
; должен быть создан. Появляется предупреждение, если указанная
; функция не определена, или если функция не включает/реализует
; недостающий класс. Поэтому этот параметр следует устанавливать,
; только если действительно необходимо реализовать функцию
; обратного вызова.
unserialize_callback_func=

; Параметр serialize_precision определяет количество сохраняемых
; при сериализации значимых цифр в дробной части
; float- и double-чисел. Значение по умолчанию гарантирует, что
; десериализуемые с помощью функции unserialize float-данные
; останутся такими же, как до сериализации.
serialize_precision = 100

; Следующий параметр определяет, должен ли PHP при выполнении
; сценария всегда передавать функциям аргументы по ссылке. Этот
; метод устарел, и, скорее всего, он не будет поддерживаться в
; будущих версиях PHP/Zend. Описание того, каким способом должен
; быть передан аргумент — по ссылке или по значению —
; рекомендуется указывать при объявлении функции. Лучше всего,
; если вы попытаетесь установить параметр в Off и проверите, все
; ли сценарии по-прежнему работают. Если это так, то все в
; порядке, и сценарии будут совместимы и с будущими версиями
; PHP. В противном случае вы будете получать предупреждения
; каждый раз, когда аргументы передаются по значению там, где
; должны передаваться по ссылке.
allow_call_time_pass_reference = On

; Безопасный режим
;
safe_mode = Off

; По умолчанию в безопасном режиме при открытии файлов
; выполняется проверка UID. Чтобы смягчить строгость до проверки
; GID, необходимо включить директиву safe_mode_gid.
safe_mode_gid = Off

; Когда безопасный режим включен, не проверять UID/GID при подключении файлов,
; находящихся в указанном каталоге и его подкаталогах (кроме этого, каталог
; также должен быть записан в директиве include_path или при подключении файлов
; должен быть задан полный путь).
safe_mode_include_dir =

; При включенном безопасном режиме разрешается выполнять с помощью exec-функций
; только файлы, находящиеся в каталоге, указанном в директиве safe_mode_exec_dir.
safe_mode_exec_dir =

; Установка некоторых переменных окружения может потенциально породить
; "дыры" в защите сценариев.
; Следующая директива содержит разделенный
; запятыми список префиксов. При включенном безопасном режиме
; пользователь сможет изменять только те переменные окружения, имена
; которых начинаются с перечисленных префиксов.
; По умолчанию пользователь имеет возможность устанавливать только
; те переменные окружения, имена которых начинаются с префикса
; PHP_ (например, PHP_FOO=BAR).
; Замечание: если значение этой директивы не указано, PHP позволяет
; пользователям модифицировать любые переменные окружения!
safe_mode_allowed_env_vars=PHP_
```

```
; Следующая директива содержит разделенный запятыми список имен
; переменных окружения, которые конечный пользователь не
; сможет изменять путем вызова putenv(). Эти переменные будут
; защищены даже в том случае, если директива
; safe_mode_allowed_env_vars разрешает их изменять.
safe_mode_protected_env_vars = LD_LIBRARY_PATH

; Директива open_basedir (если она установлена) разрешает
; использовать файловые операции только в пределах заданного каталога
; и его подкаталогов. Лучше всего, если эта директива будет
; использоваться для отдельных каталогов или для конфигурационного
; файла виртуального сервера. Директива действует независимо от того,
; включен безопасный режим или нет!
;open_basedir =

; Следующая директива дает возможность запрещать вызовы некоторых
; функций из соображений безопасности. Список задается в виде имен
; функций, отделенных друг от друга запятыми. Директива действует
; независимо от того, включен безопасный режим или нет!
disable_functions =

; Следующая директива дает возможность запрещать некоторые классы
; из соображений безопасности. Список задается в виде имен классов,
; отделенных друг от друга запятыми. Директива действует независимо
; от того, включен безопасный режим или нет!
disable_classes =

; Цвета для режима подсветки синтаксиса. Можно использовать любой
; цвет, допустимый в тэге <font color=?????>.
;highlight.string = #DD0000
;highlight.comment = #FF8000
;highlight.keyword = #007700
;highlight.bg = #FFFFFF
;highlight.default = #0000BB
;highlight.html = #000000
;
; Другие директивы
;
; Следующая директива указывает, должен ли PHP добавлять свою
; сигнатуру в заголовки, посылаемые браузеру, и, таким образом,
; обнаруживать себя. Это никак не может повлиять на безопасность
; сценария, однако позволяет пользователю определить, использовался
; PHP для генерации страницы или нет.
expose_php = On
```

Ограничения ресурсов

Для решения задач все приложения используют серверные ресурсы. Ограничения ресурсов включают в себя ограничения циклов процессора, дискового пространства и т.д. Конфигурационный файл `php.ini` содержит параметры, ограничивающие использование PHP-интерпретатором этих ресурсов.

Эти параметры позволяют задать PHP ограничение на использование ресурсов, необходимых для обработки одного запроса. Значения, установленные по умолчанию, подходят для большинства приложений. Если сценарий достигает одного из этих пределов, то это должно побудить разработчика проверить код на предмет дефектов или рассмотреть другие способы решения задачи.

```
;;;;;;;;;;;;;
; Ограничения ресурсов ;
;;;;;;;;;;;;;
```

```

max_execution_time = 30 ; Максимально возможное время выполнения
; сценария в секундах.
max_input_time = 60 ; Максимально возможное время
; синтаксического анализа данных запроса
memory_limit = 8M ; Максимальный объем памяти, выделяемый
; сценарию (8MB)

```

Обработка и протоколирование ошибок

В зависимости от параметров этого раздела PHP может по-разному обрабатывать ошибки. В ходе разработки Web-приложения целесообразно оставить стандартные параметры вывода сообщений об ошибках, поскольку это позволит быстрее находить ошибки в программе. С другой стороны на реальном Web-сайте PHP-ошибки могут сообщать конечным пользователям информацию, которую предоставлять нежелательно, и, таким образом, создавать потенциальные проблемы безопасности Web-сайтов. Кроме всего прочего, ошибки в Web-приложении ставят разработчика в неудобное положение. В таком случае рекомендуется использовать протоколирование ошибок, при котором ошибки незаметно записываются в журнал, который не доступен конечным пользователям.

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Обработка и протоколирование ошибок ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Директива error_reporting должна задаваться в виде битового
; поля. Его значение можно устанавливать с помощью следующих
; констант:
; E_ALL          - Все предупреждения и ошибки.
; E_ERROR        - Критические ошибки времени выполнения.
; E_WARNING      - Предупреждения времени выполнения (не
;                  критические ошибки).
; E_PARSE        - Ошибки трансляции.
; E_NOTICE       - Замечания времени выполнения (
;                  предупреждения, которые, скорее всего,
;                  свидетельствуют о логических ошибках в
;                  сценарии, — например, использовании
;                  неинициализированной переменной в расчете на
;                  автоматическую инициализацию с пустой строкой).
; E_STRICT       - Замечания времени выполнения. Эта константа
;                  позволяет PHP предлагать изменения кода,
;                  которые обеспечат наилучшие возможности
;                  взаимодействия и совместимость с будущими
;                  версиями PHP.
; E_CORE_ERROR   - Критические ошибки в момент запуска PHP.
; E_CORE_WARNING - Предупреждения (не критические ошибки) во время
;                  запуска PHP.
; E_COMPILE_ERROR - Критические ошибки времени трансляции.
; E_COMPILE_WARNING - Предупреждения времени трансляции.
; E_USER_ERROR   - Сгенерированные пользователем ошибки.
; E_USER_WARNING - Сгенерированные пользователем предупреждения.
; E_USER_NOTICE  - Сгенерированные пользователем замечания.

; Примеры:
; показывать все ошибки, за исключением замечаний
; error_reporting = E_ALL & ~E_NOTICE
; показывать только сообщения об ошибках
; error_reporting=E_COMPILE_ERROR|E_ERROR|E_CORE_ERROR
; отображать все ошибки, кроме замечаний и предупреждений о нарушении
; стандартов написания кода
error_reporting = E_ALL & ~E_NOTICE & ~E_STRICT

```

820 Приложение Е

```
; Печать ошибок и предупреждений прямо в браузер. Для готовых сайтов
; настоятельно рекомендуется отключать следующую директиву и
; использовать вместо нее протоколирование (см. ниже). Включенная
; директива display_errors в "рабочих" сайтах может открыть
; пользователю доступ к секретной информации, например, полный путь к
; документу на Web-сервере, имя используемой базы данных и т.д.
display_errors = On

; Даже если display_errors включена, ошибки, возникающие во время
; запуска PHP, не отображаются. Рекомендуется оставлять этот параметр
; выключенным за исключением случая, когда он необходим для отладки.
display_startup_errors = Off

; Сохранять ли сообщения об ошибках в файле журнала. Журнал может
; определяться настройками сервера, быть связанным с потоком stderr
; или же задаваться директивой error_log, описанной ниже. Как уже
; было сказано, в коммерческих проектах желательно записывать ошибки
; в журнал, а не отображать ошибки в браузер.
log_errors = Off

; Следующая директива устанавливает максимальный размер сообщения об
; ошибках, записываемого в журнал. В журнал добавляется информация о
; файле с исходным кодом. Стандартное значение этого параметра равно
; 1024 значение 0 снимает всякие ограничения на размер записываемых
; сообщений.
log_errors_max_len = 1024

; Не записывать повторяющиеся сообщения. Если директива
; ignore_repeated_source включена, то будут игнорироваться ошибки,
; возникающие в одной и той же строке одного файла.
ignore_repeated_errors = Off

; Не записывать имя файла с ошибкой. Когда значение этого параметра
; равно On, повторяющиеся сообщения из различных файлов или разных
; строк не будут записываться в журнал.
ignore_repeated_source = Off

; Если значение следующего параметра равно Off, то утечки памяти не
; будут показываться (на stdout или в журнале). Это имеет значение
; только в отладочной компиляции, и если разрешено выводить
; предупреждения (E_WARNING)
report_memleaks = On

; Сохранять ли последнее сообщение об ошибке или предупреждение в
; переменной $php_errormsg (boolean).
track_errors = On

; Не включать в сообщения об ошибках HTML-теги
; Примечание: этот параметр не следует использовать для "реальных"
; сайтов.
html_errors = Off

; Если HTML-теги включаются в сообщения об ошибках, то PHP
; генерирует ссылки, ведущие на страницы с подробным описанием
; ошибок или функций, вызвавших ошибки. Со страницы
; http://www.php.net/docs.php можно скачать копию руководства по PHP
; и указать в директиве docref_root базовый URL локальной копии
; документации, включая символ косой черты в начале пути. Кроме того,
; необходимо указать расширение, которое будет использоваться,
; включая точку.
; Примечание: не следует использовать этот параметр для "рабочих
; сайтов".
docref_root = "/phpmanual/"
docref_ext = .html
```

```
; Строка, которая выводится перед сообщением об ошибке.
;error_prepend_string = "<font color=ff0000>"
; Строка, которая отображается после сообщения об ошибке.
;error_append_string="</font>"
; Записывать журнал в указанный файл.
;error_log = filename;
; Использовать системный журнал (в Windows NT - Журнал событий,
; использовать этот параметр в Windows 95 нет смысла)
;error_log = syslog
```

Обработка данных

Самый важный параметр в этом разделе, а возможно и во всей конфигурации — `register_globals`. Эта директива позволяет автоматически преобразовывать входные данные в переменные, например, поле ввода с именем `last_name` в HTML-форме автоматически преобразовывается в переменную `$last_name`, которую затем можно использовать в сценарии. Такая мощная функция дает возможность очень быстро разрабатывать приложения. Вместе с тем, если `register_globals` используется не правильно, то может возникнуть путаница и серьезный риск для безопасности (это, возможно, единственная самая распространенная причина проблем с безопасностью PHP-приложений), поэтому, чтобы обезопасить сайт, следует аккуратно использовать эту функцию и всегда тщательно проверять входные данные. Те, кто долгое время использовал PHP, привыкли включать `register_globals`, потому что некогда эту функцию невозможно было отключить. В настоящее время по умолчанию функция выключена.

Также стоит упомянуть директиву `magic_quotes_gpc`. Она способствует повышению безопасности приложений, автоматически экранируя “опасные” символы. Обычно это делается для того, чтобы предотвратить атаки с использованием SQL-кода (SQL injection). Если приложение уже обрабатывает входные данные таким способом, то директиву следует отключить, иначе некоторые символы будут экранированы дважды, а во входных строках будут появляться лишние символы обратной косой черты.

```
;;;;;;;;;;;;;;
; Обработка данных ;
;;;;;;;;;;;;;;
;
;Замечание: директива track_vars всегда включена, начиная с PHP
; 4.0.3.

; Здесь указывается символ-разделитель для отделения друг от друга
; аргументов в генерируемых PHP URL. По умолчанию используется
; амперсанд "&".
;arg_separator.output = "&";

; В следующей директиве задается разделитель (или список
; разделителей), которые используются в PHP для преобразования URL в
; переменные. По умолчанию используется амперсанд "&".
; Замечание: разделителем считается каждый символ в этом списке.
;arg_separator.input = ";&"

; Следующая директива определяет, в каком порядке PHP будет
; регистрировать GET- и POST-данные, данные, полученные из
; cookie-файлов, а также переменные окружения и встроенные переменные
; (соответственно, значение задается буквами G, P, C, E и S,
; например, EGPCS или GPC). Регистрация производится по этой строке
; слева направо, новые значения переопределяют старые.
variables_order="EGPCS"
```



```
; Должен ли PHP регистрировать EGPCS-переменные как глобальные
; переменные. Можно отключить эту функцию, чтобы не "засорять"
; глобальную область видимости сценария. Это особенно полезно,
; если используется директива track_vars — в этом случае
; получить доступ к GPC-данным можно через переменные $HTTP_*_VARS[].
; Желательно так писать сценарии, чтобы они по возможности
; обходились без директивы register_globals. Использование
; данных, поступивших из формы, как глобальных переменных,
; потенциально может породить проблемы в защите сценария, если
; программист специально не позаботится об их устранении.
register_globals = Off

; Следующая директива заставляет PHP регистрировать массивы входных
; данных в старом стиле, т.е. массивы HTTP_GET_VARS и др. Если эти
; массивы в приложении не используются, то рекомендуется отключить
; директиву в целях повышения производительности интерпретатора.
register_long_arrays = On

; Следующая директива указывает PHP, обязан ли он создавать
; переменные $argv и $argc на основе информации, поступившей
; посредством метода GET. Если эти переменные не используются в
; приложениях, то директиву register_argc_argv следует отключить —
; это повысит производительность PHP.
register_argc_argv = On

; Максимально допустимый размер принимаемых POST-данных.
post_max_size = 8M

; Следующая директива устарела — вместо нее следует использовать
; variables_order.
gpc_order="GPC"

; Автоматическая обработка кавычек и апострофов
;
; Использовать ли автокавычки для входящих GET/POST/Cookie-данных
magic_quotes_gpc = Off

; Заключать ли в автокавычки данные, генерируемые во время
; выполнения, например, данные из SQL, exec() и т.д.
magic_quotes_runtime = On

; Нужно ли PHP оформлять автокавычки в стиле Sybase-style (заменять '
; на '', а не на \')
magic_quotes_sybase = Off

; Следующие директивы указывают PHP, содержимое каких файлов он
; должен обрабатывать до и после вывода сценария соответственно.
auto_prepend_file =
auto_append_file =

; Начиная с версии 4.0b4, PHP всегда сообщает браузеру об
; используемой кодировке в заголовке Content-type. Чтобы отключить
; эту функцию необходимо просто оставить следующую директиву пустой.
;
; По умолчанию используется text/html.
default_mimetype = "text/html"
;default_charset = "iso-8859-1"

; Следующая директива (если она используется) заставляет PHP
; наполнять данными массив $HTTP_RAW_POST_DATA.
;always_populate_raw_post_data = On
```

Пути и каталоги

PHP для решения многих задач использует файловую систему. В этом разделе представлены настройки, которые влияют на обработку этих задач. Например, чтобы подключить какой-либо файл к сценарию с помощью функций `require()` или `include()`, можно просто разместить этот файл в каталоге сценария, а можно поместить его в специальный каталог подключаемых файлов и указать путь к этому каталогу в директиве `include_path`. Это позволит не указывать впоследствии весь путь к файлу.

```

;;;;;;;;;;;;;;;;;;;;;;;;
; Пути и каталоги ;
;;;;;;;;;;;;;;;;;;;;;;;;

; UNIX: "/path1:/path2"
;include_path = "./php/includes"
;
; Windows: "\ path1;\ path2"
;include_path = ".;c:\ php\ includes"

; Корневой каталог для PHP-сценариев.
; Игнорируется, если значение равно "". Если PHP не был скомпилирован
; с ключом FORCE_REDIRECT, то директива doc_root ДОЛЖНА быть
; установлена, если PHP работает как CGI на любом сервере (кроме
; IIS). Обратитесь к документации по вопросам безопасности сайтов.
; Можно также использовать параметр cgi.force_redirect; см. ниже.
doc_root =

; Каталог, который PHP использует при открытии сценария вида
; /~username. Игнорируется, если значение равно "".
user_dir =
; Каталог, в котором хранятся динамически загружаемые расширения
; (модули).
extension_dir = ./

; Следующая директива разрешает или запрещает использование функции
; dl(). Функция dl() работает неправильно в многопоточных
; Web-серверах, например, в IIS или Zeus, и автоматически отключается
; для них.
enable_dl = On

; Директива cgi.force_redirect необходима для обеспечения
; безопасности при работе PHP в виде CGI в большинстве Web-серверов.
; Если она не определена, то PHP по умолчанию включает ее
; автоматически. Ее можно отключить здесь на собственный страх и
; риск.
; **Эту директиву МОЖНО, а по сути, НУЖНО отключить для IIS.**
; cgi.force_redirect = 1

; Если предыдущая директива включена и используется Web-сервер
; отличный от Apache или Netscape(iPlanet), то может возникнуть
; необходимость создать переменную окружения, которую PHP будет
; проверять и в зависимости от ее значения продолжать или
; останавливать выполнение сценария. Установка этой переменной может
; вызвать проблемы в защите сценариев. Используйте ее, только если
; вы знаете что делаете.
; cgi.redirect_status_env = ;

; FastCGI на Web-сервере IIS (в операционных системах на основе
; WINNT) поддерживает возможность имитировать маркеры безопасности
; вызывающего клиента. Это позволяет IIS определить контекст
; безопасности, в котором выполняется запрос. В Web-сервере Apache
; mod_fastcgi по состоянию на 17.03.2002 не поддерживает эту функцию.

```

```
; Установите значение 1 для IIS. По умолчанию используется 0.
; fastcgi.impersonate = 1;

; Параметр cgi.rfc2616_headers сообщает PHP, заголовки какого типа
; использовать при отправке кода HTTP-ответа. Значение 0
; соответствует заголовку Status:, который поддерживается
; Web-сервером Apache, а значение 1 соответствует
; RFC2616-совместимому заголовку. По умолчанию используется 0.
;cgi.rfc2616_headers = 0
```

Загрузка файлов на сервер

Если приложение зависит от загрузки файлов на сервер, то для временного хранения загружаемых файлов потребуется немало свободного места на диске. Чтобы избежать переполнения диска, можно даже использовать файловую систему в памяти (такая файловая система создается из операционной системы, а в директиве `upload_tmp_dir` необходимо указать соответствующий путь). Стандартная настройка максимального размера загружаемых файлов `upload_max_filesize` весьма консервативна, поэтому ее значение можно увеличить, если есть уверенность, что пользователи не превысят указанный предел.

```
;;;;;;;;;;
; Загрузка файлов на сервер ;
;;;;;;;;;;

; Следующая директива разрешает PHP загружать файлы на сервер.
file_uploads = On

; Каталог для временных файлов, в который PHP помещает закачанные по
; HTTP файлы (если в директиве указана пустая строка, то используется
; системный временный каталог)
;upload_tmp_dir =

; Максимальный размер закачанного файла
upload_max_filesize = 2M
```

Обработчики функции fopen

Самым важным параметром в этом разделе является директива `allow_url_fopen`, предоставляющая удобный способ рассматривать Web-сайты как обычные файлы. По умолчанию значение этой директивы равно `On`. Чтобы запретить другим программам, работающим с этим сервером, использовать эту функцию, необходимо установить значение `Off`.

Описанные ниже параметры имеют важное значение. Иногда требуется, чтобы PHP открывал файлы на удаленных машинах, используя URL. В таких случаях возможность устанавливать таймаут (`default_socket_timeout`) или имя пользователя для анонимного доступа (`from`) позволяет найти и открыть необходимый файл на удаленном сервере.

```
;;;;;;;;;;
; Обработчики функции fopen ;
;;;;;;;;;;

; Следующая директива определяет, можно ли трактовать URL (http://
; или ftp://) как файлы.
```

```

allow_url_fopen = On

; Здесь задается пароль для анонимного FTP-доступа (email-адрес).
; from="john@doe.com"

; Здесь определяется строка User-Agent.
; user_agent="PHP"

; Таймаут по умолчанию для потоков данных, основанных на сокетах
; (в секундах).
default_socket_timeout = 60

; Если сценарии должны работать с файлами, созданными в системах
; Macintosh, или PHP работает на Macintosh-машине и необходимо
; работать с файлами из Unix- или Win32-систем, установка этого флага
; заставляет PHP автоматически определять символы конца строки (EOL)
; так, чтобы функции fgets() и file() работали правильно.
; auto_detect_line_endings = Off

```

Динамически загружаемые расширения

Расширения позволяют добавлять функциональность в PHP без перекомпиляции исходного кода. Обычно для работы с базами данных пользователю приходится раскомментировать имя расширения. Например, если PHP работает на Windows-платформе и при этом нужно подключаться к PostgreSQL-серверу, следует раскомментировать строку `extension=php_pgsql.dll`. В Unix-платформах используется файловое расширение `.so` вместо `.dll`.

```

;::::::::::::::::::::::::::::::::::::::::::
; Динамически загружаемые расширения ;
;::::::::::::::::::::::::::::::::::::::::::
;
; Чтобы модули загружались автоматически, необходимо задавать
; настройки в формате:
;
;   extension=modulename.extension
;
; Например, для Windows:
;
;   extension=mysqli.dll
;
; для UNIX:
;
;   extension=mysqli.so
;
; Примечание: указывается только имя файла расширения без пути.
; Чтобы задать каталог, в котором расположены расширения, следует
; использовать директиву extension_dir, описанную выше.
;
; Расширения для Windows
; Примечание: поддержка MySQL и ODBC теперь включена в ядро PHP,
; поэтому для нее уже не нужны никакие библиотеки DLL.
;
;extension=php_bz2.dll
;extension=php_cpdf.dll
;extension=php_curl.dll
;extension=php_dba.dll
;extension=php_dbase.dll
;extension=php_dbx.dll
;extension=php_exif.dll
;extension=php_fdf.dll

```

```
;extension=php_filepro.dll
;extension=php_gd2.dll
;extension=php_gettext.dll
;extension=php_iconv.dll
;extension=php_ifx.dll
;extension=php_iisfunc.dll
;extension=php_imap.dll
;extension=php_interbase.dll
;extension=php_ldap.dll
;extension=php_mbstring.dll
;extension=php_mcrypt.dll
;extension=php_mhash.dll
;extension=php_mime_magic.dll
;extension=php_ming.dll
;extension=php_mssql.dll
;extension=php_mysql.dll
;extension=php_mysqli.dll
;extension=php_oci8.dll
;extension=php_openssl.dll
;extension=php_oracle.dll
;extension=php_pdf.dll
;extension=php_pgsql.dll
;extension=php_shmop.dll
;extension=php_snmp.dll
;extension=php_sockets.dll
;extension=php_sybase_ct.dll
;extension=php_w32api.dll
;extension=php_xmlrpc.dll
;extension=php_xsl.dll
;extension=php_yaz.dll
;extension=php_zip.dll
```

Настройки расширений

Настройки расширений — параметры, которые передаются загружаемым расширениям (чтобы узнать, какие расширения загружаются, обратитесь к разделу “Динамически загружаемые расширения”). В этом разделе задаются параметры многих расширений, поэтому следует изучить настройки для всех необходимых расширений, остальные можно проигнорировать. Например, если вместо баз данных Oracle нужно использовать базы данных PostgreSQL, то лучше сразу просмотреть параметры, имена которых начинаются с `pgsql`.

```
;;;;;;;;;;;;;
; Настройки расширений ;
;;;;;;;;;;;;;

[Syslog]
; Следует определять различные переменные Syslog, такие как
; $LOG_PID, $LOG_CRON и т.д. Для ускорения работы рекомендуется
; выключать следующую директиву. Во время выполнения сценария
; эти переменные можно определить с помощью функции
; define_syslog_variables().
define_syslog_variables = Off

[mail function]
; Только для Win32.
SMTP = localhost

; Только для Win32.
;sendmail_from = me@example.com
```

```
; Только для UNIX. Можно задать аргументы (по умолчанию:
; "sendmail -t -i").
;sendmail_path =

; Передавать программе sendmail заданные параметры как.
; дополнительные Эти параметры всегда заменяют значение
; пятого параметра mail(), даже в безопасном режиме.
;mail.force_extra_paramaters =

[SQL]
sql.safe_mode = Off

[ODBC]
;odbc.default_db = Пока не реализовано
;odbc.default_user = Пока не реализовано
;odbc.default_pw = Пока не реализовано

; Разрешает или запрещает постоянные соединения
odbc.allow_persistent = On

; Проверка доступности соединения перед его использованием.
odbc.check_persistent = On

; Макс. число постоянных соединений. -1 означает, что ограничений нет.
odbc.max_persistent = -1

; Макс. число соединений (в том числе постоянных).
; -1 означает, что ограничений нет.
odbc.max_links = -1

; Обработка LONG-полей. Возвращает в переменную количество байтов. 0
; означает режим passthru
odbc.defaultlrl = 4096

; Установки для бинарных данных. 0 означает режим passthru, 1 - режим
; "как есть", 2 - преобразование в тип char.
; См. документацию по odbc_binmode и odbc_longreadlen для более
; детального разъяснения смысла директив odbc.defaultlrl и
; odbc.defaultbinmode.
odbc.defaultbinmode = 1

[MySQL]
; Разрешать или запрещать постоянные соединения.
mysql.allow_persistent = On

; Максимальное количество постоянных соединений. -1 означает, что
; ограничений нет.
mysql.max_persistent = -1

; Максимальное количество соединений (включая постоянные). -1
; означает, что ограничений нет.
mysql.max_links = -1

; Порт по умолчанию для функции mysql_connect(). Если порт не задан,
; функция попытается использовать переменную $MYSQL_TCP_PORT
; или запись mysql-tcp в файле /etc/services, либо заданную во время
; компиляции PHP константу MYSQL_PORT (именно в таком порядке).
; В PHP для Win32 используется только константа.
mysql.default_port =

; Здесь определяется имя сокета по умолчанию для локальных соединений
; MySQL. Если сокет не задан, то используется встроенное значение по
```

```
; умолчанию в MySQL.
mysql.default_socket =

; Хост по умолчанию для mysql_connect() (не работает в безопасном
; режиме).
mysql.default_host =

; Пользователь по умолчанию для mysql_connect()
; (не работает в безопасном режиме).
mysql.default_user=

; Пароль по умолчанию для mysql_connect() (не работает в безопасном
; режиме).
; Замечание: вообще идею хранить пароль в этом файле нельзя назвать
; хорошей. *Любой* пользователь, который может запускать PHP, сможет
; узнать пароль путем выполнения следующего оператора:
; echo cfg_get_var("mysql.default_password")
; Конечно, узнать пароль сможет также и пользователь, который имеет
; права на чтение этого файла.
mysql.default_password =

; Максимальное время (в секундах) ожидания соединения. -1 означает,
; что ограничений нет.
mysql.connect_timeout = 60

; Режим трассировки. Когда режим трассировки активен (trace_mode =
; On), отображаются предупреждения о сканировании таблиц/индексов, а
; также SQL-ошибки.
mysql.trace_mode = Off

[MySQL]
; Разрешать или запрещать постоянные соединения.
mysql.allow_persistent = On

; Максимальное количество постоянных соединений. -1 означает, что
; ограничений нет.
mysql.max_persistent = -1

; Максимальное количество соединений (включая постоянные). -1
; означает, что ограничений нет.
mysql.max_links = -1

[PostgreSQL]
; Разрешать или запрещать постоянные соединения.
pgsql.allow_persistent = On

; Всегда обнаруживать разорванные постоянные соединения с помощью
; pg_pconnect(). Вносит незначительные издержки.
pgsql.auto_reset_persistent = Off

; Максимальное количество постоянных соединений. -1 означает, что
; ограничений нет.
pgsql.max_persistent = -1

; Максимальное количество соединений (включая постоянные). -1
; означает, что ограничений нет.
pgsql.max_links = -1

; Следующая директива определяет, следует ли игнорировать замечания
; PostgreSQL-серверов.
pgsql.ignore_notice = 0

; Записывать замечания PostgreSQL-серверов в журнал или нет.
```

```
; Если pgsql.ignore_notice не равно 0, то расширение не может
; записывать сообщения в журнал.
pgsql.log_notice = 0

[Sybase]
; Разрешать или запрещать постоянные соединения.
sybase.allow_persistent = On

; Максимальное количество постоянных соединений. -1 означает, что
; ограничений нет.
sybase.max_persistent = -1

; Максимальное количество соединений (включая постоянные). -1
; означает, что ограничений нет.
sybase.max_links = -1

sybase.interface_file = "/usr/sybase/interfaces"

; Минимальный уровень серьезности отображаемых ошибок.
sybase.min_error_severity = 10

; Минимальный уровень серьезности отображаемых сообщений.
sybase.min_message_severity = 10

; Режим совместимости со старыми версиями PHP 3.0.
; Если этот режим включен, то PHP не трактует все результаты как
; строки, а автоматически назначает им типы согласно типам в Sybase.
; Этот режим, скорее всего, в будущих версиях будет упразднен,
; поэтому лучше всего его выключить и внести соответствующие
; изменения в код.
sybase.compatability_mode = Off

[Sybase-CT]
; Разрешать или запрещать постоянные соединения.
sybct.allow_persistent = On

; Максимальное количество постоянных соединений. -1 означает, что
; ограничений нет.
sybct.max_persistent = -1

; Максимальное количество соединений (включая постоянные). -1
; означает, что ограничений нет.
sybct.max_links = -1

; Минимальный уровень серьезности отображаемых сообщений сервера.
sybct.min_server_severity = 10

; Минимальный уровень серьезности отображаемых сообщений клиента.
sybct.min_client_severity = 10

[dbx]
; Возвращаемые имена столбцов можно конвертировать в целых
; совместимости. Возможные значения директивы dbx.colnames_case:
; "unchanged" ("не изменять"; используется по умолчанию, если
; директива не установлена)
; "lowercase" ("в нижний регистр")
; "uppercase" ("в верхний регистр")
; Рекомендуется использовать либо значение "uppercase", либо
; "lowercase" в настоящее время в целях обратной совместимости
; используется значение "unchanged"
dbx.colnames_case = "unchanged"

[bcmath]
; Количество десятичных цифр для всех bcmath-функций.
```


830 Приложение Е

```
bcmath.scale = 0

[browscap]
;browscap = extra/browscap.ini

[Informix]
; Хост по умолчанию для ifx_connect() (не работает в безопасном
; режиме).
ifx.default_host =

; Пользователь по умолчанию для ifx_connect() (не работает в
; безопасном режиме).
ifx.default_user =

; Пароль по умолчанию для ifx_connect() (не работает в безопасном
; режиме).
ifx.default_password =

; Разрешать или запрещать постоянные соединения.
ifx.allow_persistent = On

; Максимальное количество постоянных соединений. -1 означает, что
; ограничений нет.
ifx.max_persistent = -1

; Максимальное количество соединений (включая постоянные). -1
; означает, что ограничений нет.
ifx.max_links = -1

; Если следующая директива установлена в On, select-оператор
; возвращает содержимое поля типа text blob вместо его
; идентификатора.
ifx.textasvarchar = 0

; Эта директива заставляет оператор select возвращать значение поля
; типа byte blob вместо его идентификатора.
ifx.byteasvarchar = 0

; Заставляет PHP удалять завершающие пробелы из столбцов типа char
; фиксированного размера. Может помочь пользователям Informix SE.
ifx.charasvarchar = 0

; Если следующая директива установлена, то содержимое полей text и
; byte сохраняется в файле, а не в памяти.
ifx.blobinfile = 0

; Если значение следующей директивы не равно 1, то NULL-значения
; возвращаются как пустые строки, иначе они возвращаются
; как строки 'NULL'.
ifx.nullformat = 0

[Session]
; Определяет режим хранения данных сеансов.
session.save_handler = files

; Следующая директива задает аргумент, передаваемый save_handler.
; В случае режима сохранения в файлах здесь должен указываться
; каталог, в который будут сохраняться файлы сеансов.
; Примечание: чтобы использовать функции сеансов в Windows-системах,
; необходимо изменить это значение.
;
; Начиная с PHP 4.0.1, можно определить путь как:
;
;     session.save_path = "N;/path"
```

```
;
; где N - целое число. В результате все файлы сеанса будут
; сохраняться не в каталоге /path, а в подкаталогах с N-уровнем
; вложений. Это полезно, если в операционной системе имеются
; сложности с сохранением большого количества файлов в одном
; каталоге, а, кроме того, это более эффективно для серверов,
; обрабатывающих большое количество сеансов.
; Примечание 1: PHP автоматически не создает такую структуру
; каталогов. С этой целью можно использовать
; сценарий, находящийся в каталоге ext/session
; Примечание 2: см. ниже раздел настроек, управляющих
; сборкой мусора, если для хранения сеансов
; используются подкаталоги.
; Модуль сохранения сеансов в файлах создает файлы,
; используя по умолчанию права доступа 600.
; Изменить настройку можно с помощью директивы:
; session.save_path = "N;MODE;/path"
; где MODE - восьмеричное представление прав доступа.
; При этом umask процесса не меняется.
session.save_path = "/tmp"

; Должен ли PHP использовать cookie-файлы.
session.use_cookies = 1

; Следующая директива позволяет администраторам защитить своих
; пользователей от атак, в которых используется передача
; идентификатор сеанса в URL; по умолчанию значение равно 0.
; session.use_only_cookies = 1

; Имя сеанса (используется как имя cookie-файла).
session.name = PHPSESSID

; Инициализация сеансов в начале запроса.
session.auto_start = 0

; Время жизни cookie-файла сеанса. 0 - до закрытия браузера.
session.cookie_lifetime = 0

; Путь, для которого cookie-файл является достоверным.
session.cookie_path=/

; Домен, для которого cookie-файл является достоверным.
session.cookie_domain=

; Функция, используемая для сериализации данных. Значение php задает
; стандартную функцию PHP.
session.serialize_handler= php

; Вероятность того, что при очередном запуске сценария, работающего с
; сеансами, будет вызвана функция "сборки мусора". Вероятность
; вычисляется по формуле gc_probability/gc_divisor, 1/100 означает,
; что сборка мусора запускается 1 раз в 100 запросов.
session.gc_probability = 1
session.gc_divisor = 100

; После указанного здесь промежутка времени сохраненные данные будут
; удалены автоматически сборщиком мусора.
session.gc_maxlifetime = 1440

; Примечание: если для хранения файлов сеансов используются
; подкаталоги (см. раздел session.save_path выше),
; то сборка мусора *не* включается автоматически.
; Необходимо использовать собственную сборку мусора
; посредством shell-сценария, cron-задания или другого
; метода. Например, следующий сценарий эквивалентен
```

832 Приложение Е

```
;          настройке session.gc_maxlifetime = 1440 (24 минуты):
;          cd /path/to/sessions; find -cmin +24 | xargs rm;

; В версиях PHP до 4.2 включительно есть недокументированная функция
; (или дефект), которая позволяет инициализировать переменные сеанса
; в глобальной области видимости, даже тогда, когда директива
; register_globals отключена. PHP 4.3 и выше предупреждает, если эта
; функция используется. Эту функцию и предупреждение можно отключать
; по отдельности. В настоящее время предупреждение отображается,
; только если включена директива bug_compat_42.
session.bug_compat_42 = 1
session.bug_compat_warn = 1

; Проверять заголовок HTTP Referer для аннулирования внешне
; сохраненных URL, содержащих идентификаторы. Если HTTP_REFERER
; не содержит этой подстроки, то сеанс считается недостоверным.
session.referer_check =

; Указывает, сколько байтов читать из файла.
session.entropy_length = 0

; Файл, используемый для генерации идентификаторов сеанса.
session.entropy_file =

;session.entropy_length = 16

;session.entropy_file=/dev/urandom

; Установите одно из значений nocache, private, public для
; определения параметров HTTP-кэширования.
session.cache_limiter = nocache

; Документ будет считаться устаревшим по истечении заданного
; здесь количества минут
session.cache_expire = 180

; Поддержка "переходящих" SID отключена по умолчанию.
; Использование переходящих SID рискованно для безопасности
; пользователей. Эту директиву следует использовать осторожно.
; - Пользователь может отправить по email/irc и т.д.
;   URL, содержащий идентификатор активного сеанса.
; - URL, содержащий идентификатор активного сеанса, может
;   быть сохранен на компьютере в общественном месте
; - Пользователь может зайти на сайт, используя тот же
;   идентификатор сеанса (например, сохранив закладку в браузере)
session.use_trans_sid = 0

; Выбор хеш-функции
; 0: MD5 (128 бит)
; 1: SHA-1 (160 бит)
session.hash_function = 0
; Здесь определяется количество битов, сохраняемых в каждом символе
; при преобразовании бинарных хеш-данных в читабельную форму.
; 4 бита: 0-9, a-f
; 5 бит: 0-9, a-v
; 6 бит: 0-9, a-z, A-Z, "-", ",", " "
session.hash_bits_per_character = 4

; Для перезаписи URL-адресов, выбирается определенный набор
; HTML-тегов. form/fieldset - особый случай;
; если они будут здесь включены, то при перезаписи URL будет
; добавлено скрытое поле ввода с информацией, которая в противном
; случае добавляется в конец URL. Для соответствия XHTML-стандарту
; запись form необходимо удалить.
```

```
; Следует заметить, что в корректной записи всегда должен
; присутствовать знак "=", даже если значение отсутствует.
url_rewriter.tags = "a:href,area:href,frame:src,input:src,form:,fieldset="

[MSSQL]
; Разрешать или запрещать постоянные соединения.
mssql.allow_persistent = On

; Максимальное количество постоянных соединений. -1 означает, что
; ограничений нет.
mssql.max_persistent = -1

; Максимальное количество соединений (включая постоянные). -1
; означает, что ограничений нет.
mssql.max_links = -1

; Минимальный уровень серьезности отображаемых ошибок.
mssql.min_error_severity = 10

; Минимальный уровень серьезности отображаемых сообщений.
mssql.min_message_severity = 10

; Режим совместимости со старыми версиями PHP 3.0.
mssql.compatability_mode = Off

; Таймаут соединения
;mssql.connect_timeout = 5

; Таймаут запроса
;mssql.timeout = 60

; Допустимый диапазон 0 - 2147483647. По умолчанию = 4096.
;mssql.textlimit = 4096

; Допустимый диапазон 0 - 2147483647. По умолчанию = 4096.
;mssql.textsize = 4096

; Ограничение количества записей в каждом пакете.
; 0 - все записи в одном пакете.
;mssql.batchsize = 0

; Режим возвращения значений столбцов datetime и datetim4:
; On - возвращаются данные, преобразованные в типы SQL Server
; Off - значения возвращаются в формате YYYY-MM-DD hh:mm:ss;
;mssql.datetimeconvert = On
; Использование NT-аутентификации при подключении к серверу
mssql.secure_connection = Off

; Здесь указывается максимальное количество процессов.
; По умолчанию 25
;mssql.max_procs = 25

[Assertion]
; Assert(expr); функция активна по умолчанию.
;assert.active = On

; Генерирует предупреждения PHP для каждой неудавшейся проверки
; выражения.
;assert.warning = On

; По умолчанию не завершать программу в случае неудачи.
;assert.bail = Off

; Пользовательская функция, которая будет вызвана при неудачной
```

```
; проверке.
;assert.callback = 0

; Вычислять выражения в eval с использованием текущих установок
; error_reporting().TRUE означает, что для функции eval() действует
; установка error_reporting(0).
;assert.quiet_eval = 0

[Ingres II]
; Разрешать или запрещать постоянные соединения.
ingres.allow_persistent = On

; Максимальное количество постоянных соединений. -1 означает, что
; ограничений нет.
ingres.max_persistent = -1

; Максимальное количество соединений (включая постоянные). -1
; означает, что ограничений нет.
ingres.max_links = -1

; База данных по умолчанию (формат: [node_id::]dbname[/srv_class]).
ingres.default_database =

; Пользователь по умолчанию.
ingres.default_user =

; Пароль по умолчанию.
ingres.default_password =

[Verisign Payflow Pro]
; Сервер Payflow Pro по умолчанию.
pfpro.defaulthost = "test-payflow.verisign.com"

; Порт по умолчанию.
pfpro.defaultport = 443

; Таймаут по умолчанию (в секундах).
pfpro.defaulttimeout = 30

; IP-адрес прокси-сервера по умолчанию (если требуется).
pfpro.proxyaddress =
; Прокси-порт по умолчанию.
pfpro.proxyport =

; Имя пользователя для прокси-сервера по умолчанию.
pfpro.proxylogon =

; Пароль для прокси-сервера по умолчанию.
pfpro.proxypassword =

[SocketS]
; Использование системной функции read() вместо оболочки php_read().
sockets.use_system_read = On

[com]
; Путь к файлу, содержащему GUID или IID или имена файлов TypeLibs;
com.typelib_file =
; Разрешить DCOM-вызовы
com.allow_dcom = true
; Автоматическая регистрация констант typelib-компонентов в com_load()
com.autoregister_typelib = true
; Имена констант нечувствительны к регистру символов
com.autoregister_casesensitive = false
; Показывать предупреждение при дублирующей регистрации констант
```

```
;com.autoregister_verbose = true

[mbstring]
; Язык внутреннего представления символов.
;mbstring.language = Japanese

; Внутренняя кодировка и кодировка сценариев.
; Некоторые кодировки не могут работать как внутренние
; (например, SJIS, BIG5, ISO-2022-*).
;mbstring.internal_encoding = EUCLJP

; Кодировка входных HTTP-данных.
;mbstring.http_input = auto

; Кодировка исходящих HTTP-данных. Для работы этой функции должен
; быть зарегистрирован обработчик mb_output_handler
;mbstring.http_output = SJIS

; Разрешать автоматическое преобразование кодировки согласно
; параметру mbstring.internal_encoding Если эта директива включена,
; то входные символы преобразовываются во внутреннюю кодировку.
; Примечание: не используйте автоматическое преобразование
; кодировки для переносимых библиотек и приложений.
;mbstring.encoding_translation = Off

; Порядок автоматического определения кодировки.
; По умолчанию auto
;mbstring.detect_order = auto

; Использование функции замены символов, когда преобразование
; невозможно
;mbstring.substitute_character = none;

; Переопределять (заменять) однобайтовые функции mbstring-функциями.
; Функции mail(), ereg() и т.д. заменяются mb_send_mail(), mb_ereg()
; и т.д. Возможные значения: 0,1,2,4 или их комбинации.
; Например, 7 означает "переопределять все функции".
; 0: не переопределять
; 1: переопределять mail()
; 2: переопределять функции str*()
; 4: переопределять функции ereg*()
;mbstring.func_overload = 0

[FrontBase]
;fbsql.allow_persistent = On
;fbsql.autocommit = On
;fbsql.default_database =
;fbsql.default_database_password =
;fbsql.default_host =
;fbsql.default_password =
;fbsql.default_user = "SYSTEM"
;fbsql.generate_warnings = Off
;fbsql.max_connections = 128
;fbsql.max_links = 128
;fbsql.max_persistent = -1
;fbsql.max_results = 128
;fbsql.batchSize = 1000

[exif]
; Комментарии Exif UNICODE обрабатываются как UCS-2BE/UCS-2LE, а JIS
; как JIS. Поддержка mbstring позволяет автоматически конвертировать
; их в кодировку, заданную соответствующей директивой. Если значение
; не задано, то используется mbstring.internal_encoding.
; В настройках декодирования различается порядок следования байтов
```

```
; motorola и intel. Настройки декодирования не могут быть пустыми.
;exif.encode_unicode = ISO-8859-15
;exif.decode_unicode_motorola = UCS-2BE
;exif.decode_unicode_intel    = UCS-2LE
;exif.encode_jis =
;exif.decode_jis_motorola = JIS
;exif.decode_jis_intel    = JIS

; Локальные переменные:
; tab-width: 4
; End:
```

Предметный указатель

A

API-интерфейс, 363; 561; 790
ArgoUML, 520
ASCII, 85

B

base64, 595
BEdit, 36

C

Caldera, 41
CDATA, 346
CERT, 49
CGI, 38; 805; 813
CGI-программа, 48; 63
CLI-интерфейс, 804; 805
cookie-файл, 106; 145; 146; 153
CPAN, 553
Cron, 809
CSV, 304

D

dBase, 374
DBM, 374; 774
Debian, 39; 41
Definite, 41
DHTML, 560
Dia, 520
DOM, 344; 363
DSN, 791; 793
DSO-объекты, 46
DTD, 100; 345

E

EBNF, 348
Emacs, 36
Eudora, 590

F

Fatal Error, 224
FreeBSD, 39
FreeType, 630
FTP, 295
FTP-сервер, 71

G

GD, 606
GPL, 376; 774

H

HTML, 35; 65; 100; 343
атрибуты, 101
дескрипторы, 101
<form>, 101; 111
<input>, 111
<select>, 111; 130; 233; 801
<textarea>, 111; 119; 332
теги, 84; 100
форма, 74; 110; 111; 338
формат письма, 596
HTML DTD, 101
HTTP, 34; 36; 50; 295
заголовок, 73; 74; 148; 151
запрос, 71; 72; 73; 153; 221
методы, 73; 99
GET, 73; 113
HEAD, 73
POST, 73; 106; 111; 113; 115
ответ, 72; 74; 106; 111
сообщение, 73
тело, 73; 74

I

IBM Rational Rose, 520
inode, 297; 322
IP-адрес, 72; 106
ISO, 265

J

JavaScript, 332; 566; 597

K

KDE, 50
Konquerer, 44

L

LAMP, 40
libxml2, 364
LZW-сжатие, 606

M

make, 379
Mandrake, 41
MIME, 589; 595
 заголовки
 Content-Description, 595
 Content-Disposition, 595
 Content-Id, 595
 Content-Transfer-Encoding, 595
 Content-Type, 595
 MIME Version, 595
MS Access, 794
MTA, 589
MVC, 566
MySQL, 40; 48
 avg(), 427
 count(), 427
 curdate(), 422; 448
 curtime(), 422; 448
 date_format(), 448
 dayname(), 449
 max(), 427
 min(), 427
 now(), 422; 447
 password(), 394; 416; 463
 substring(), 422
 sum(), 427
 to_days(), 449
mysqladmin, 379
mysqld, 379

N

Notepad, 36; 57
Notice, 224
NTFS, 293

O

ODBC, 790
Outlook, 590

P

PCRE, 242
PCS, 555
PEAR, 49; 69; 268; 553; 597
 стандарты написания кода, 555
PECL, 554
Perl, 553
PFC, 554
PHPUnit, 635; 640
PHP-интерпретатор, 291; 323
PHP-расширения
 DOM, 363
 GD, 612
 mailparse, 589
 simpleXML, 343; 364
 SQLite, 775
Poseidon For UML, 520
PostgreSQL, 46

R

RAD, 267
Red Hat, 39; 41
RFC, 590
 2045, 595
 2821, 590
 2822, 590
 821, 590
root, 62
RPM, 37; 40; 378

S

SAPI, 38; 50; 57
Sendmail, 589
SGML, 343
SimpleXML, 565
Smarty, 635; 639
 section, 638
 шаблон, 669
SMTP-сервер, 590; 592
SOAP, 353
SQL, 240; 385; 423
 выражения, 232

запросы, 388; 421
 DELETE, 394; 791
 INSERT, 394; 791
 REPLACE, 395
 SELECT, 791
 UPDATE, 394; 791

ключевые слова

AFTER, 413
 CHANGE, 413
 FIRST, 413
 MODIFY, 413

команды, 790

ALTER TABLE, 775
 CREATE, 405
 CREATE TABLE, 408
 GRANT, 395; 776
 INSERT, 415
 REPLACE, 415
 REVOKE, 396; 776

операторы, 388

IGNORE, 476
 INSERT, 782
 LIKE, 426
 NOT LIKE, 426
 UPDATE, 445

предложения

GROUP BY, 428
 HAVING, 429
 LIMIT, 423
 ORDER BY, 425
 WHERE, 393; 423

SQL injection, 821

SQLite, 40; 374; 774

SSL, 143; 149

SuSE, 41

SVG, 608

Syslog, 826

T

True Type, 630

try/catch, 232; 255

TurboLinux, 41

U

Umbrello, 520

UML, 518; 519

UML-диаграмма, 503; 519; 646

активности, 526

классов, 520; 644

последовательностей, 644

ситуаций, 527

Unix, 53

URI, 350

URL, 113; 146; 250; 295; 346

URL-кодирование, 114

V

vi, 36

Visio, 520

W

Warning, 224

Web-браузер

IE, 36

Mozilla, 36

Web-сервер, 38; 50; 65; 71; 76; 805

Apache, 34; 36; 38; 46; 293

IIS, 34; 36; 187

Personal Web Server (PWS), 39; 297

Web-службы, 352

WSDL, 353

X

XHTML, 65; 68; 114; 345; 349

XML, 100; 343

XML Namespace, 351

xml_parser, 344

XML-анализатор, 345

XML-язык, 353

Y

Yellow Dog, 41

Z

Zend Engine, 76; 104

Zend-расширение, 806

A

Абстрактный метод, 513

Агрегирование, 525

Аддитивная цветовая модель, 607

Администратор, 62

Актор, 527
Алгоритм шифрования, 469
Аргументы командной строки, 807
Аргументы функций, 231
Атрибуты, 344
 action, 113; 338
 checked, 122
 method, 113
 multiple, 130
 name, 133
 value, 122
Аутентификация пользователей, 466

Б

База данных, 374
Базовые классы РНР-кода, 554
Базы данных, 146
 встроенные, 374
 запись (строка), 381
 индекс, 387
 клиент/серверные, 375
 нормализация, 382
 поле (столбец), 381
 реляционные, 381
Безопасный режим, 815; 817
Бесконечные циклы, 187; 230
Бизнес-логика, 635
Булевы
 значения, 161
 логика, 161
 операторы, 166
 термы, 161

В

Веб-публикации, 53
Веб-узел по умолчанию, 58
Векторные изображения, 608
Видимость членов класса, 515
Владелец файла, 314
Вложенность функций, 285
Внешний ключ, 384; 775
Водяные знаки, 620
Временная метка, 148; 311; 321
Вызов функции, 272
Выражения, 82

Г

Гиперссылки, 109
Группы пользователей, 314

Д

Демон, 50; 805; 810
Дескриптор, 343; 344
 каталога, 322
 результата, 777
 файла, 294
Деструктор, 499
Динамический модуль, 38; 47
Диспетчера пакетов PEAR, 555
Доменное имя, 250; 590
Дочерний элемент, 348
Драйвер базы данных, 790

Ж

Жесткие ссылки, 322
Журнал ошибок, 255

З

Зависимость пакетов, 555
Зив Сураски, 35
Значение, 94

И

Идентификатор
 группы (GID), 315
 изображения, 609
 сеанса, 145; 147; 153
 соединения, 398
Имя
 источника данных, 791
 файла, 294
Индексированное изображение, 609
Индексный узел, 322
Инкапсуляция, 488; 515; 669
Инсталлятор, 37
Интерактивная оболочка, 804
Интерактивность CLI, 811
Интерпретаторы, 805
Интерпретируемый язык
 программирования, 804
Интерфейс, 488; 512; 535; 645; 664

разработки серверных приложений, 39; 50
Итерация, 181; 199

К

Каталог, 292; 293
Класс, 487; 488
Классы символов, 244
Клиент, 71
Ключ, 94
Ключевые слова
 \$this, 492
 abstract, 513
 break, 177
 case, 177
 class, 489
 extends, 503
 function, 270
 global, 80; 106; 281; 319
 interface, 513
 parent, 510
 private, 493; 515
 public, 492
 return, 271
 static, 80; 282
Код HTTP-состояния, 74
Кодировка, 595
 ISO-8859-1, 354
 UTF-16, 347
 UTF-8, 347
Комментарии, 70; 555; 556
Компилятор, 43
Конкатенация, 213
Консорциум W3C, 100; 344
Константы, 81
Конструктор, 497; 647
Контроль типов, 517
Конфигурационные параметры
 auto_prepend_file, 469
 display_errors, 254
 error_reporting, 254
 log_errors, 254
 magic_quotes_gpc, 821
 odbc.default_db, 793
 odbc.default_pw, 793
 odbc.default_user, 793
 register_globals, 821
 safe_mode, 776
 sendmail_from, 592
 sendmail_path, 592
 short_open_tag, 815

Конфигурационный файл, 37; 813
Корневой элемент, 344; 346; 348
Курсор, 791
Кэширование, 63

Л

Логика представления, 635
Локализация неисправностей, 223
Локальный узел, 67

М

Маркеры границ, 595
Маршрут, 72
Маршрутизация, 72
Массивы, 94; 196
 ассоциативные, 104; 202; 792
 индекс, 94; 133
 символьный, 196
 числовой, 196
 инициализация, 95; 197
 многомерные, 209
 предопределенные
 \$_COOKIE, 106
 \$_ENV, 107
 \$_FILES, 106
 \$_GET, 106
 \$_POST, 106
 \$_REQUEST, 107
 \$_SERVER, 106
 \$_SESSION, 107
 \$GLOBALS, 106
 сортировка, 97
 суперглобальные, 106
 элемент, 94; 133
Международная организация по
 стандартизации, 265
Международный почтовый консорциум, 604
Методы, 269; 488
 классов
 __construct(), 496
 __destruct(), 499
 __get(), 494
 __set(), 494
Многоэлементные MIME-сообщения, 595
Множественность, 525
Модель RGB, 607
Модульная организация классов, 486

Н

Наследование, 488; 503
Непрозрачность, 624
Нормализация, 382
Нормальные формы, 382
Нотация
 @, 254
 венгерская, 557
 инфиксная, 82
 постфиксная, 82
 префиксная, 82

О

Область видимости, 281
Обобщение, 524
Оболочки
 Bash, 804
 Csh, 804
 shell, 315
 Zsh, 804
Обработка ошибок, 254
Образец поиска, 241
Общественная библиотека PHP-
 расширений, 554
Объединение, 430
Объект, 487; 489
Объектно-ориентированное
 программирование, 485; 487
Операторы
 != и <>, 170
 @, 321
 == и ===, 169
 ->, 490
 > и <, 166
 AND, OR, !, 172
 break, 234
 exit, 234
 if, 165
 include, 288
 include_once, 291
 new, 490
 require, 288
 require_once, 291
 return, 272
 switch, 176; 277
 арифметические, 83
 бинарные, 82
 битовые, 83

 ветвления, 164
 для работы с массивами, 83
 инкрементные/декрементные, 83; 89
 контроля ошибок, 83
 логические, 83; 172
 присваивания, 81; 83; 88
 сравнения, 83
 строковые, 83
 тернарные, 82
 унарные, 82
 условные, 164
Операционная система
 Linux, 34; 36; 293; 314
 MS-DOS, 804
 Red Hat Fedora Linux, 38
 Red Hat Linux, 34
 Windows, 34; 293
 Windows 2000, 34; 36; 38
 Windows 98, 39
Определение класса, 489
Определение типа документа, 345; 561
Оптимизация кода, 268
Отладка, 232
Отображение ошибок, 224
Отступы, 555
Ошибки
 времени выполнения, 222; 227
 критические, 224; 227
 логические, 223; 227
 предупреждение, 224
 синтаксические, 222; 224
 уведомление, 224

П

Пакеты, 553
 RPM, 41; 42
 данных, 72
Палитра, 610
Первичный ключ, 383; 388; 407; 778
Переменные, 76
 \$argc, 807
 \$argv, 807
 внутренние переменные класса, 492
 время жизни, 281
 глобальные, 282
 именование, 77
 локальные, 282
 область видимости, 77; 80; 281
 предопределенные, 104

\$_POST, 103
 \$_SERVER, 103
 \$GLOBALS, 105
 сеанса, 154
 статические, 80; 282
 суперглобальные, 104
 \$_ENV, 319
 \$_GLOBALS, 281
 тип данных, 77; 78
 экземпляра, 513
 Переопределение, 507
 Повторное использование кода, 486
 Подключаемые файлы, 289
 Поисковые машины, 116
 Поле формы, 73; 116
 скрытое, 134; 145; 153
 Полиморфизм, 508
 Полноцветное изображение, 609
 Пользовательская оболочка, 315
 Постановка задачи, 160
 Почтовый сервер, 71
 Права доступа, 62; 293
 Приведение типов, 79
 Приоритет операторов, 82
 Пространство имен, 351
 Протокол
 FTP, 71
 HTTP, 63; 71; 72; 144
 IMAP, 590
 POP3, 590
 SMTP, 590
 SSL, 137
 TCP/IP, 72
 передачи гипертекста, 34
 Псевдокод, 70; 160
 Псевдонимы, 428

Р

Разделители, 69; 344
 Расмус Лердорф, 35
 Растровые изображения, 608
 Расширение файлов, 67
 Регулярные выражения, 241; 426; 572
 POSIX-совместимые, 242
 альтернативы, 245
 границы слова, 245
 квалификаторы, 245
 квантификаторы, 246
 специальные символы, 243
 языка Perl, 242

Результирующее множество, 389; 452; 777; 791
 Рекурсия, 285; 325
 Ресурс, 227
 Ричарда Хипп, 774
 Родительский элемент, 346

С

Свойства, 488
 Сглаживание цветов, 623
 Сглаживание шрифта, 624
 Сеансы, 71; 145; 153
 Сервер, 71
 Серверный модуль, 813
 Сжатие
 без потерь, 608
 с потерями, 608
 Синтаксический анализ, 67
 Система управления реляционными базами
 данных, 36
 Системные ресурсы, 36
 Системный DSN, 795
 Соглашения по именованию, 539; 557
 Состояние приложения, 144
 Специальные символы, 238; 444
 Спецификатор доступа, 494
 Спецификация, 265; 266
 Стандарт написания кода, 264; 268
 Стандартный ввод, 811
 Стандартный вывод, 306; 811
 Стандартный поток ошибок, 811
 Статические методы, 542
 Статический модуль, 38; 48
 Строка запроса, 73; 110; 146; 153
 Структуры ветвления, 159
 Структуры управляющей логики, 159
 СУРБД, 373; 374; 376; 381; 774
 DB/2, 375
 MS SQL Server, 375; 793
 MySQL, 36; 375
 Oracle, 375
 PostgreSQL, 232; 375
 SQLite, 36; 642

Т

Тело HTTP-запроса, 115
 Тело HTTP-ответа, 75
 Тип
 данных, 486; 775

- скалярный, 79
- сложный, 79
- специальный, 79
- курсора, 799
- Типизация
 - слабая, 76; 78
 - строгая, 78
- Типоператора, 83
- Точка отсчета, 307
- Транзакции, 792
- Трансляция, 76
- Триггеры, 775

У

- Указатель файла, 294
- Унифицированный указатель ресурсов, 250
- Управляющие структуры, 555
- Уровень отображения ошибок, 223
- Учетная запись, 460

Ф

- Файл, 293
 - crontab, 810
 - httpd.conf, 51
 - INSTALL, 38; 46; 49
 - php.ini, 37; 57; 62; 104; 154; 224; 255; 469; 592
 - README, 38
 - режим доступа, 296
- Файловая система, 292; 322
- Фильтры ISAPI, 58
- Форматы изображений
 - GIF, 609
 - JPEG, 608
 - PNG, 608
 - SVG, 608
 - WBMP, 608
- Функции, 67; 271
 - addslashes(), 240; 444
 - append_child(), 364
 - array(), 94; 198
 - array_count_values(), 97
 - array_flip(), 97
 - array_multisort(), 214
 - array_pop(), 334
 - arsort(), 207
 - asort(), 97; 207
 - asXML(), 365

- basename(), 319
- chdir(), 324
- chr(), 85
- close(), 325
- closedir(), 322
- copy(), 320
- count(), 97
- create_element(), 364
- current(), 203
- date(), 65; 68
- date_str(), 318
- define(), 81
- dir(), 325
- dirname(), 324
- disk_free_space(), 706
- disk_total_space(), 706
- display(), 329
- domxml_new_doc(), 363
- domxml_open_file(), 363
- domxml_open_mem(), 363
- each(), 205
- echo(), 176
- ereg(), 242; 330
- ereg_replace(), 252; 330
- eregi(), 458
- eregi_replace(), 458
- error_reporting(), 402
- exec(), 322
- explode(), 241; 459; 808
- fclose(), 292
- feof(), 303
- feof(), 302
- fgetc(), 302
- fgetcsv(), 302
- fgets(), 302
- file(), 294
- file_exists(), 310
- file_get_contents(), 359
- file_info(), 319
- fileatime(), 311; 572
- filectime(), 311
- filegroup(), 315
- filemtime(), 312
- fileowner(), 315
- filesize(), 311
- filetype(), 315
- floor(), 89
- fopen(), 292; 294
- fpasssthru(), 305
- fputs(), 302

- fread(), 298
fseek(), 307
ftell(), 307
func_get_args(), 509
fwrite(), 294; 298
getdate(), 312
gettype(), 79
header(), 148; 470; 612
HTMLSpecialChars(), 238; 444
imagealphablending(), 624
imagearc(), 613; 708
imagecolorallocate(), 609
imagecolorallocatealpha(), 624
imagecolorat(), 622
imagecolorexact(), 623
imagecolorresolve(), 610
imagecolortransparent(), 623
imagecopy(), 621
imagecopymerge(), 624
imagecopyresampled(), 626
imagecopyresized(), 626
imagecreate(), 610
imagecreatefromgif(), 618
imagecreatefromjpeg(), 618
imagecreatefrompng(), 618
imagecreatetruecolor(), 610
imagecreatetruecolor(), 609
imagedestroy(), 612
imagedestroy(), 618
imageellipse(), 613
imagefilledarc(), 708
imagefontheight(), 629
imagefontwidth(), 629
imagefttext(), 630
imageline(), 611
imageloadfont(), 628
imagepng(), 612
imagepolygon(), 614
imagerectangle(), 612
imagesetpixel(), 610
imagestring(), 627
imagesx(), 621
imagesy(), 621
import_request_variables(), 105
include, 71
include(), 268
include_once(), 270
is_array(), 94
is_dir(), 317
is_file(), 317
isset(), 157; 168; 259; 358; 690
key(), 203
ksort(), 207
list(), 205
mail(), 575; 589; 592
make_dir(), 328
md5(), 596
mkdir(), 324
mkdir_form(), 328
move_uploaded_file(), 338
mysql_field_name(), 450
mysql_affected_rows(), 399; 447
mysql_close(), 398
mysql_connect(), 398
mysql_create_db(), 410
mysql_data_seek(), 420
mysql_drop_db(), 410
mysql_errno(), 402
mysql_error(), 402
mysql_fetch_array(), 419; 705
mysql_fetch_field(), 453
mysql_fetch_object(), 420
mysql_fetch_row(), 399; 418
mysql_field_flags(), 451
mysql_field_len(), 450
mysql_field_name(), 452
mysql_field_type(), 450
mysql_insert_id(), 463
mysql_list_dbs(), 398
mysql_list_fields(), 450
mysql_list_tables(), 399
mysql_num_fields(), 450
mysql_num_rows(), 399
mysql_numrows(), 704
mysql_query(), 409; 704
mysql_result(), 420
mysql_select_db(), 398
next(), 204
odbc_autocommit(), 792
odbc_close(), 791
odbc_close_all(), 791
odbc_commit(), 792
odbc_connect(), 791
odbc_do(), 791
odbc_error(), 792
odbc_errormsg(), 792
odbc_exec(), 791
odbc_execute(), 791
odbc_fetch_array(), 792
odbc_fetch_row(), 792

- odbc_free_result(), 791
- odbc_num_rows(), 791
- odbc_pconnect(), 791
- odbc_prepare(), 791
- odbc_procedures(), 792
- odbc_result(), 792
- odbc_rollback(), 792
- odbc_statistics(), 792
- odbc_tableprivileges(), 792
- odbc_tables(), 792
- opendir(), 322
- pg_exec(), 704
- pg_fetch_array(), 705
- pg_numrows(), 704
- phpinfo(), 104; 806
- pi(), 89
- posix_getgid(), 315
- posix_getpuid(), 315
- prev(), 204
- print_r(), 96; 106; 362
- rand(), 89; 167
- read(), 325
- readdir(), 323
- readfile(), 305
- rename(), 320
- require, 71
- require(), 268; 334
- require_once(), 268; 270
- rewind(), 307
- rewinddir(), 324
- rmdir(), 324
- roundrect(), 616
- rsort(), 207
- session_id(), 157
- session_register(), 153; 154
- session_start(), 153; 467
- setcookie(), 106; 148
- settype(), 79
- simplexml_load_file(), 366
- simplexml_load_string(), 365
- sort(), 97; 207; 324
- sqlite_array_query(), 778
- sqlite_busy_timeout(), 779
- sqlite_changes(), 778
- sqlite_close(), 777
- sqlite_column(), 778
- sqlite_create_aggregate(), 779
- sqlite_create_function(), 779
- sqlite_current(), 778
- sqlite_error_string(), 779
- sqlite_escape_string(), 779
- sqlite_fetch_array(), 778
- sqlite_fetch_single(), 778
- sqlite_field_name(), 778
- sqlite_has_more(), 778
- sqlite_last_error(), 779
- sqlite_last_insert_rowid(), 778
- sqlite_libencoding(), 777
- sqlite_libversion(), 777
- sqlite_next(), 777
- sqlite_num_fields(), 778
- sqlite_num_rows(), 778
- sqlite_open(), 776
- sqlite_popen(), 777
- sqlite_query(), 777
- sqlite_rewind(), 777
- sqlite_seek(), 777
- sqlite_udf_decode_binary(), 779
- sqlite_udf_encode_binary(), 779
- sqlite_unbuffered_query(), 777
- stat(), 297
- str_replace(), 322
- stripslashes(), 240; 444
- strlen(), 84; 87; 239
- strpos(), 84
- strstr(), 84; 239
- substr(), 240; 422
- system(), 322
- time(), 68; 148; 321
- touch(), 574
- traverse_dir(), 326
- trim(), 808
- unlink(), 320
- urlencode(), 576
- var_dump(), 583
- vimagecreate(), 609
- xml_parse_into_struct(), 359
- xml_parser_create(), 359
- аргументы, 271
- группировки, 427
- параметры, 271
- передача по значению, 279
- передача по ссылке, 279
- пользовательские, 270
- регулярных выражений, 239

X

Хранимые процедуры, 792

Ц

Циклы, 181
do while, 188
for, 191
foreach, 215
while, 181; 801

Э

Экстремальное программирование, 267
Элементы управления, 116
переключатели, 124; 127
списки, 130
текстовые поля, 116
флажки, 122
Энди Гутманс, 35

Я

Языки
ASP, 69; 70
C, 40
JavaScript, 35
PERL, 35; 242; 804
Python, 242; 804
Tcl, 804
Visual Basic, 70
XML Schema, 345
написания сценариев, 67
разметки SGML, 100
Языковая конструкция, 67
Языковое ядро, 76
Якоря, 245

Научно-популярное издание

**Дэйв У. Мерсер, Аллан Кент, Стивен Новицки, Дэвид Мерсер,
Дэн Скуайер, Ван Кью Чой**

РНР 5 для начинающих

Литературный редактор	<i>П.Н. Мачуга</i>
Верстка	<i>В.И. Бордюк</i>
Художественный редактор	<i>В.Г. Павлютин</i>
Корректоры	<i>А.В. Луценко, В.В. Смоляр, Л.В. Чернокозинская</i>

Издательский дом "Вильямс"
101509, г. Москва, ул. Лесная, д. 43, стр. 1

Подписано в печать 04.07.2006. Формат 70х100/16.
Гарнитура Times. Печать офсетная.
Усл. печ. л. 68,37. Уч.-изд. л. 50,18.
Тираж 3 000 экз. Заказ № .

Отпечатано по технологии СtP
в ОАО "Печатный двор" им. А. М. Горького
197110, Санкт-Петербург, Чкаловский пр., 15

По договору между издательством **"Вильямс"** и Интернет-Магазином "Books.Ru - Книги России" единственный легальный способ получения данного файла с книгой **“PHP 5 для начинающих ” (ISBN 5-8459-1039-0)** – покупка в Интернет-магазине "Books.Ru - Книги России".

Если вы получили данный файл каким-либо другим образом, вы нарушили законодательство об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству **"Вильямс"** где именно вы получили данный файл.