

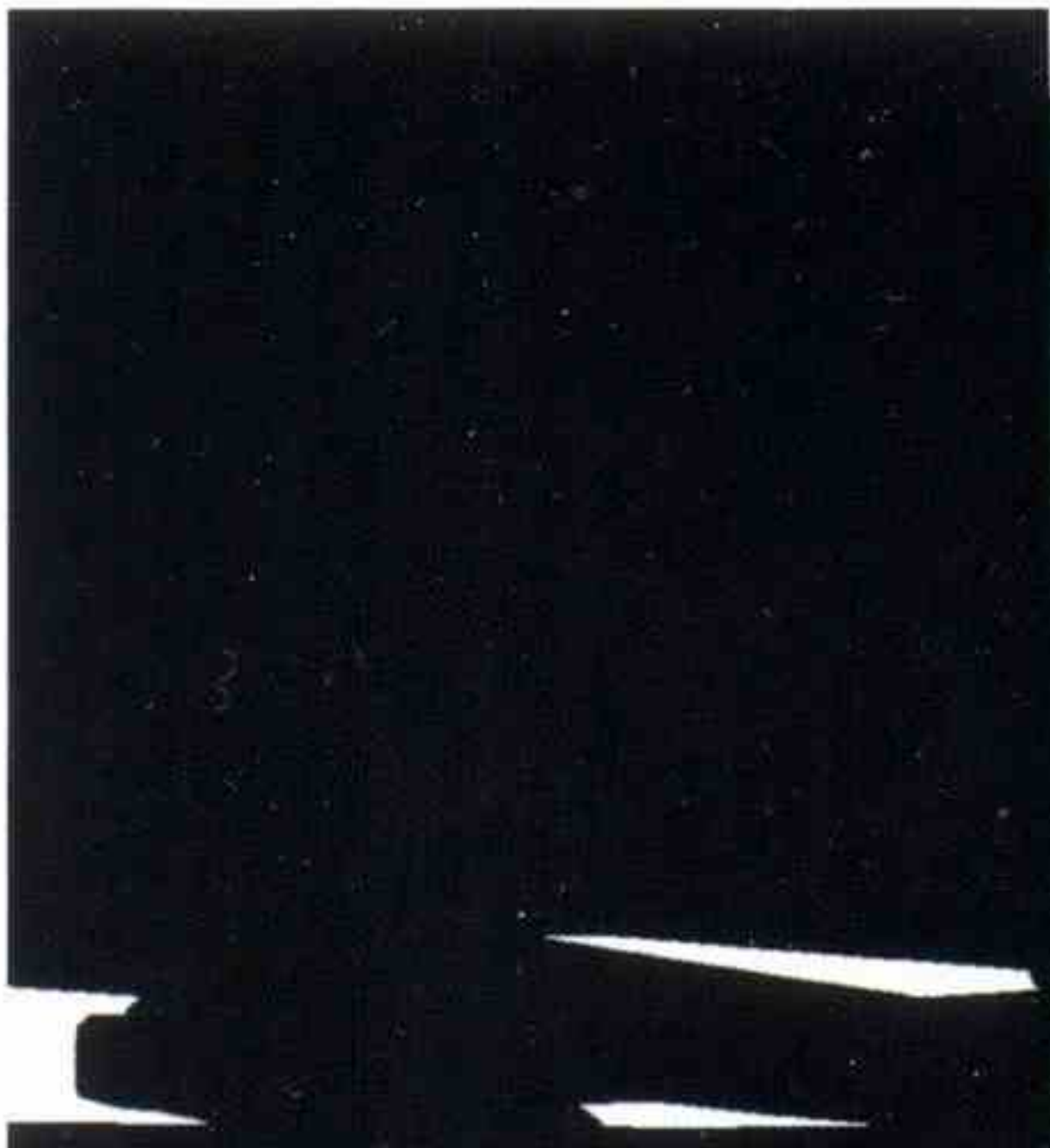
ДЕНИС КОЛИСНИЧЕНКО



PHP и MySQL

Разработка веб-приложений

5-е издание



СИНТАКСИС ЯЗЫКА PHP
ОСОБЕННОСТИ НОВЕЙШЕЙ
ВЕРСИИ PHP 5.5
ОБЪЕКТНО-ОРИЕНТИРОВАН-
НОЕ ПРОГРАММИРОВАНИЕ
НА PHP 5.5
ИСПОЛЬЗОВАНИЕ СЕРВЕРОВ
БАЗ ДАННЫХ MySQL и SQLite
САМЫЕ ПОЛЕЗНЫЕ
PHP-ФУНКЦИИ
ШАБЛОНИЗАТОР Smarty
ОСНОВЫ Ajax, ИНТЕГРАЦИЯ
PHP И Flash
РАБОТА С SMTP, POP, RSS,
MP3, reCAPTCHA, Disqus
ШИФРОВАНИЕ КОДА
СЦЕНАРИЯ С ПОМОЩЬЮ Zend
Guard
СБОРКА XAMPP

PRO

**ПРОФЕССИОНАЛЬНОЕ
ПРОГРАММИРОВАНИЕ**



Материалы
на www.bhv.ru

УДК 004.43+004.738.5
ББК 32.973.26-018.1
К60

Колисниченко Д. Н.

К60 PHP и MySQL. Разработка веб-приложений. — 5-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2015. — 592 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-3514-4

Даны начала программирования на PHP: установка и настройка Apache, PHP и MySQL, а также кроссплатформенной сборки XAMPP, выбор редактора PHP-кода, синтаксис языка и самые полезные функции PHP. Рассмотрено создание собственного движка сайта и ряда дополнительных модулей — фотогалереи, RSS-граббера, модуля для работы с MP3, сервиса reCAPTCHA, а также применение мощного шаблонизатора Smarty и создание простейшего собственного шаблонизатора. Показано, как с помощью технологии Ajax добиться обновления данных на странице без ее перезагрузки. В книге представлено два реальных проекта: универсальная система управления контентом, которая подойдет для построения типичного сайта, и одного из реальных проектов автора. В пятом издании рассмотрены нововведения версии 5.5 (генераторы, новые ключевые слова, Password Hashing API и др.), а также особенности создания индикатора загрузки файла и разыменовывания массивов.

На сайте издательства находятся дополнительные главы, все листинги из книги, а также необходимое программное обеспечение.

Для веб-программистов

УДК 004.43+004.738.5
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Екатерина Капалыгина</i>
Редактор	<i>Григорий Добин</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Дизайн серии	<i>Инны Тачиной</i>
Оформление обложки	<i>Марины Дамбиевой</i>

Подписано в печать 29.08.14.
Формат 70×100¹/₁₆. Печать офсетная. Усл. печ. л. 47,73.
Тираж 1500 экз. Заказ № 704.
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.
Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-3514-4

© Колисниченко Д. Н., 2015
© Оформление, издательство "БХВ-Петербург", 2015

Оглавление

Предисловие	15
Что нового в 5-м издании?	15
Немного истории	16
RНР 5.5, или что случилось с RНР 6?.....	17
Основные нововведения в RНР версии 5.4	19
Типажи.....	19
Разыменовывание массива.....	19
Встроенный веб-сервер	19
Индикатор загрузки файла.....	19
Основные нововведения в RНР версии 5.5	20
Генераторы.....	20
Ключевые слова <i>finally</i> и <i>class</i>	20
API хэширования паролей	20
Разыменовывание строк.....	20
Изменения в работе конструкции <i>foreach</i>	20
Поддержка графического формата WebP.....	20
Исключенные возможности.....	20
MySQL 21	21
Поддержка читателей	21
 <u>ЧАСТЬ I. ТЕОРИЯ</u>.....	23
 РАЗДЕЛ 1. БЫСТРЫЙ СТАРТ	25
Глава 1. Установка необходимого программного обеспечения.....	27
1.1. Нужно ли устанавливать программное обеспечение?	27
1.2. Выбор RНР-редактора и FTP-клиента	29
1.3. Установка связки Apache + RНР + MySQL в Windows.....	31
1.3.1. Для опытных пользователей: установка вручную	31
1.3.2. Установка для новичков (рекомендуется).....	38
1.4. Установка Apache + RНР + MySQL в Linux	42
1.5. Конфликты с другим программным обеспечением	43
1.6. Функция <i>phpinfo()</i> : получение информации о RНР.....	44
1.7. Несколько советов	46

Глава 2. Программа на PHP.....	47
2.1. Ваша первая программа	47
2.2. Запуск PHP-программы.....	48
2.3. Вывод текста без <i>echo</i>	49
Глава 3. Основы синтаксиса PHP.....	51
3.1. Переменные.....	51
3.1.1. Правила объявления переменных. Имена переменных.....	51
3.1.2. Типы данных переменных	52
3.1.3. Булевы переменные	53
3.1.4. Операции над переменными.....	54
3.1.5. Ссылки.....	55
3.2. Константы	56
3.3. Выражения и операции	57
3.3.1. Что такое выражение.....	57
3.3.2. Арифметические операции	57
3.3.3. Логические выражения	58
3.3.4. Приоритеты операций.....	58
3.3.5. Операторы эквивалентности <code>==</code> и <code>===</code>	59
3.3.6. Операции со строками.....	60
3.4. Условный оператор	61
3.5. Циклы	62
3.5.1. Цикл со счетчиком <i>for</i>	62
3.5.2. Цикл <i>while</i>	62
3.5.3. Цикл <i>do-while</i>	63
3.5.4. Принудительное завершение цикла и пропуск итерации.....	63
3.6. Оператор выбора <i>switch-case</i>	64
РАЗДЕЛ 2. ПЕРЕДАЧА ПАРАМЕТРОВ PHP-ПРОГРАММАМ.....	67
Глава 4. Методы <i>GET</i> и <i>POST</i>.....	69
4.1. Интерфейс CGI	69
4.2. Метод <i>GET</i>	71
4.3. Метод <i>POST</i>	71
Глава 5. Протокол HTTP и интерфейс CGI.....	73
5.1. Специальные переменные окружения CGI.....	73
5.2. Заголовки протокола HTTP	74
5.3. Коды ответов протокола HTTP	75
Глава 6. Передача параметров посредством HTML-формы	77
6.1. Создание простейшей формы и ее обработка в сценарии.....	77
6.2. Создание пользовательского интерфейса с помощью формы.....	80
6.2.1. Ввод текста. Теги <i>INPUT</i> и <i>TEXTAREA</i>	81
6.2.2. Зависимые и независимые переключатели.....	82
6.2.3. Списки выбора	83
6.2.4. Форма для передачи файлов.....	85
6.2.5. Кнопки.....	85

6.3. Проверка параметров формы.....	86
6.3.1. Проверка корректности e-mail.....	87
6.3.2. Проверка правильности номера кредитной карты.....	88
6.3.3. Удаление лишних пробелов.....	90
6.4. Защита от спама с помощью формы CAPTCHA.....	90
6.5. Форма поиска в Яндексе	92

Глава 7. Запоминание параметров с помощью Cookies и сессий 94

7.1. Что такое Cookies и как с ними работать?.....	94
7.2. Механизм сессий	96
7.2.1. Сессии и Cookies: преимущества и недостатки	96
7.2.2. Для чего нужны сессии?	96
7.2.3. Как работает механизм сессий?.....	98
7.2.4. Обход Cookies	99
7.3. Массивы и Cookies	100

Глава 8. Отдельное слово о директиве *register_globals* 101

8.1. Почему опасно использовать <i>register_globals</i> ?	101
8.2. Если <i>register_globals</i> отключена.....	105

РАЗДЕЛ 3. МАССИВЫ И СПИСКИ..... 107

Глава 9. Основные операции над массивами и списками 109

9.1. Массив и список. Цикл <i>foreach</i>	109
9.2. Функции <i>list()</i> и <i>array()</i>	111
9.3. Удаление массива	113
9.4. Слияние массивов	113
9.5. Функция <i>print_r()</i>	115
9.6. Разыменовывание массива.....	116

Глава 10. Функции сортировки массивов..... 118

10.1. Сортировка массивов	118
10.2. Функция <i>sort()</i> — сортировка списка.....	118
10.3. Функция <i>asort()</i> — сортировка массива по значениям.....	119
10.4. Функция <i>ksort()</i> — сортировка по ключам	120
10.5. Функции <i>array_reverse()</i> и <i>shuffle()</i>	121
10.6. Собственная функция сортировки.....	121
10.7. Натуральная сортировка	122

Глава 11. Особые операции над массивами..... 124

11.1. Добавление и удаление элементов массива.....	124
11.2. Упаковка переменных в массив и их извлечение	125
11.3. Получение части массива.....	127
11.4. Функции автоматического заполнения массива	127
11.5. Сравнение массивов	128
11.6. Полезные операции над массивом	129
11.6.1. Вычисление суммы и произведения всех элементов массива	129
11.6.2. Проверка существования элемента в массиве.....	129
11.6.3. Получение случайного элемента из массива.....	130

11.6.4. Удаление дубликатов из массива	130
11.6.5. Получение значений и ключей массива.....	131
11.6.6. Замена местами значений и ключей.....	131
11.6.7. Подсчет значений в массиве	131
11.6.8. Замена в массиве.....	132
11.6.9. Поиск в массиве	132
11.6.10. Прогулка по массиву	133
РАЗДЕЛ 4. ФУНКЦИИ В PHP	135
Глава 12. Полезные стандартные функции	137
12.1. Генератор случайных чисел.....	137
12.2. Дата и время.....	138
12.2.1. Кратко о timestamp.....	138
12.2.2. Функции <i>strtotime()</i> и <i>checkdate()</i>	138
12.2.3. Вывод даты.....	139
12.3. Математические функции	141
Глава 13. Функции для работы со строками	142
13.1. Основные строковые функции	142
13.2. Специальные функции замены	145
13.3. Преобразование строки.....	147
13.4. Функции преобразования кодировок	149
13.5. Функции для работы с отдельными символами строки. Разыменовывание строки	150
13.6. Функция <i>md5()</i> и другие функции шифрования/хэширования. API хэширования паролей.....	151
13.7. Функция <i>explode()</i> : выделение подстрок	152
13.8. Статистические функции	153
13.9. Функции вывода текста.....	154
13.10. Установка локали.....	156
13.11. Форматирование чисел и денежных величин.....	157
13.12. Преобразование систем счисления	157
Глава 14. Работаем с файлами и каталогами	158
14.1. Права доступа в UNIX.....	158
14.2. Чтение файла.....	160
14.2.1. Функции <i>fopen()</i> и <i>fread()</i>	161
14.2.2. Функция <i>file()</i> : построчное чтение файла	163
14.2.3. Чтение всего файла: функция <i>file_get_contents()</i>	164
14.3. Запись файла	164
14.4. Создание временных файлов	165
14.5. Работа с CSV-файлами	165
14.6. Специальные функции для работы с файлами	167
14.6.1. Функции для работы с именами файлов.....	168
14.6.2. Работа с правами доступа	168
14.6.3. Копирование, переименование и удаление файлов	169
14.6.4. Время доступа к файлу.....	170
14.6.5. Другие полезные функции	170
14.7. Совместный доступ к файлу	171
14.8. Функции для работы с каталогами.....	172

Глава 15. Вывод графических изображений средствами PHP	174
15.1. Библиотека GD	174
15.1.1. Получение информации об изображении	174
15.1.2. Конвертирование графических форматов	177
15.1.3. Вывод текста поверх картинки. Задание цвета	179
15.1.4. Прозрачность	182
15.1.5. Рисование графических примитивов	183
15.1.6. Поворот изображения	185
15.2. Изменение размера изображения	185
15.3. Создание водяных знаков	187
15.4. Поддержка графического формата WebP в PHP 5.5	189
Глава 16. Работа с сетевыми сокетами в PHP. Сетевые функции	191
16.1. Еще раз о том, что такое сокет	191
16.2. Использование сокетов	191
16.3. Пример использования сокетов	192
16.4. Блокирующий и неблокирующий режимы сокета	195
16.5. DNS-функции	196
Глава 17. Собственные функции	197
17.1. Зачем нужны собственные функции?	197
17.2. Особенности функций в PHP	197
17.3. Объявление функции	198
17.4. Области видимости функции	199
17.5. Вложенность функций	199
17.6. Переменное число аргументов	201
17.7. Передача массивов в качестве параметров	202
17.8. Генераторы	202
17.9. Особенности использования функции <i>empty()</i>	203
РАЗДЕЛ 5. СЕРВЕРЫ БАЗ ДАННЫХ MYSQL 6 И SQLITE	205
Глава 18. Основы SQL	207
18.1. Немного истории	207
18.2. Преимущества SQL	207
18.3. Как выглядят запросы?	208
18.4. Что такое база данных?	209
18.5. Создание таблиц	210
18.6. Добавление записей в таблицу	214
18.7. Обновление записей	214
18.8. Выборка записей	215
18.9. Удаление записей	216
18.10. Встроенные функции	217
18.11. Группировка записей. Сложные запросы	218
18.12. Копирование записей из одной таблицы в другую	221
18.13. Кэширование запросов	222
Глава 19. Функции для работы с MySQL	224
19.1. Подключение к серверу MySQL	224
19.2. Несколько MySQL-соединений	226

19.3. Передача запросов серверу	227
19.4. Работа с базой данных. Создание базы данных	230
19.5. Функция <i>mysql_real_escape_string(\$content)</i>	231
19.6. Получение информации о результате запроса	231
Глава 20. Альтернативная база данных SQLite	233
20.1. MySQL vs SQLite: что лучше?	233
20.2. Открытие базы данных	235
20.3. Передача запросов	235
20.4. Работа с результатом запроса	236
20.5. Список PHP-функций для работы с SQLite	237
Глава 21. Полезные приемы при работе с MySQL 6	239
21.1. Выбор кодировки	239
21.2. Сортировка: вывод новинок. Вывод случайных записей	240
21.3. Постраничный вывод таблицы	241
РАЗДЕЛ 6. ИНСТРУМЕНТЫ ДЛЯ СОЗДАНИЯ СЛОЖНЫХ ПРОЕКТОВ ...	245
Глава 22. Разработка собственного шаблонизатора	247
22.1. Организация файлов и каталогов проекта	247
22.2. Выносим параметры в отдельный файл	250
22.3. Подключение дополнительных файлов	251
22.3.1. Инструкции <i>include</i> и <i>require</i>	251
22.3.2. Альтернативный способ подключения сценариев	251
22.3.3. Инструкции <i>include_once</i> и <i>require_once</i>	253
22.4. Шаблоны	253
Глава 23. Шаблонизатор Smarty	258
23.1. Что такое Smarty?	258
23.2. Установка Smarty	258
23.3. Создание файла <i>setup.php</i>	261
23.4. Разработка шаблонов Smarty	262
23.4.1. Комментарии в шаблонах	262
23.4.2. Переменные в Smarty	263
23.4.3. Файлы конфигурации шаблонов	264
23.4.4. Служебная переменная <i>{\$_smarty}</i>	265
23.4.5. Модификаторы переменных	266
23.4.6. Стандартные (встроенные) функции Smarty	269
Функции <i>{include}</i> и <i>{insert}</i>	269
Функция <i>{foreach}</i>	269
Функции <i>{if}</i> , <i>{elseif}</i> , <i>{else}</i>	271
Функция <i>{capture}</i>	272
Функция <i>{php}</i>	272
Функция <i>{strip}</i>	272
23.4.7. Пользовательские функции Smarty	273
Функция <i>{assign}</i>	273
Функция <i>{cycle}</i>	273
Функция <i>{fetch}</i>	274

Функции <code>{html_checkboxes}</code> и <code>{html_radios}</code>	274
Функция <code>{html_image}</code>	275
Функция <code>{html_select_date}</code>	275
Функция <code>{html_select_time}</code>	276
Функция <code>{html_table}</code>	277
23.5. Smarty для программиста.....	278
23.5.1. Специальные переменные.....	278
23.5.2. Полезные методы класса <i>Smarty</i>	280
Глава 24. Объектно-ориентированное программирование.....	281
24.1. Основы ООП.....	281
24.2. Классы и объекты.....	282
24.3. Конструкторы и деструкторы класса.....	284
24.4. Наследование классов. Полиморфизм.....	285
24.5. Новые возможности PHP 5 и 5.4/5.5.....	286
24.5.1. Область видимости членов класса.....	286
24.5.2. Абстрактные классы и методы.....	287
24.5.3. Служебное слово <i>final</i>	288
24.5.4. Клонирование объектов.....	288
24.5.5. Обработка исключительных ситуаций.....	289
24.5.6. Константы-члены класса.....	292
24.5.7. Статические члены класса.....	293
24.5.8. Оператор <i>instanceof</i>	293
24.5.9. Итераторы.....	293
24.5.10. Пространства имен.....	294
24.5.11. Типажи (traits).....	294
24.5.12. Вызов метода или свойства класса выражением.....	295
Глава 25. Механизм сессий.....	297
25.1. Для чего нужны сессии?.....	297
25.2. Как работает механизм сессий?.....	298
25.3. Обход Cookies.....	300
25.4. Сценарий аутентификации.....	300
Глава 26. Введение в PEAR.....	305
26.1. Серьезные проекты и PEAR.....	305
26.2. Пример использования класса <i>DB</i>	306
Глава 27. Контроль версий.....	310
27.1. Выбор системы контроля версий.....	310
27.2. Практическое использование TortoiseHG (Mercurial).....	311
27.3. Просмотр внесенных изменений.....	313
Глава 28. Тестирование PHP-сценариев.....	315
28.1. Программа работает, но не так, как нам нужно.....	315
28.2. "Самодельные" точки останова.....	316
28.3. Система автоматического тестирования.....	317
28.4. Директива <i>error_reporting</i>	321

ЧАСТЬ II. ПРАКТИКА.....	323
РАЗДЕЛ 7. РАЗРАБОТКА ОСНОВНЫХ ЭЛЕМЕНТОВ САЙТА	325
Глава 29. Загрузка файлов на сервер	327
29.1. Что нужно знать о загрузке файлов на сервер?.....	327
29.2. Реализация загрузки файла	330
29.3. Загрузка нескольких файлов	332
29.4. Индикатор загрузки файла на PHP 5.4/5.5.....	334
29.4.1. Некоторые теоретические предпосылки.....	334
29.4.2. Пример практической реализации	337
29.5. Проблемы при загрузке файлов.....	344
Глава 30. Использование FTP-функций	345
30.1. Функции для работы с FTP	345
30.2. Примеры использования FTP-функций	348
Глава 31. Отправка и прием почты.....	351
31.1. Отправка почты средствами PHP — функция <i>mail()</i>	351
31.2. Отправка писем с вложениями — класс <i>HtmlMimeMail</i>	352
31.2.1. Отправка сообщения	352
31.2.2. Проблемы при отправке сообщения	356
31.2.3. MIME-типы	357
31.3. Класс <i>PHPMailer</i> . Разработка сценария автоматической рассылки прайс-листа	358
31.4. Получение писем по протоколу POP3	362
Глава 32. Работа с RSS: получаем новости на сайт.....	366
32.1. Краткие сведения о RSS	366
32.2. Формат RSS-файла	366
32.3. Написание сценария импорта новостей.....	369
32.4. Подключение файла <i>import.php</i> к сайту	371
32.5. Создание граббера новостей.....	372
Глава 33. Поиск с использованием регулярных выражений	376
33.1. Нужно что-то найти.....	376
33.2. Язык регулярных выражений RegExr.....	377
33.3. Управляющие конструкции	379
33.3.1. Квантификаторы	379
33.3.2. Альтернативный оператор 	379
33.3.3. Скобки	379
33.4. Псевдосимволы	380
33.5. Практическое использование RegExr-функций.....	380
33.6. Фильтры, или Конец эры регулярных выражений.....	384
Глава 34. Работаем с MP3	386
34.1. Формат MP3	386
34.2. Библиотека PEAR	387
34.3. Вывод ID3-тегов	388
34.4. Редактирование ID3-тегов	390
34.5. Удаление тега.....	390

Глава 35. Технология AJAX	392
35.1. Что такое AJAX?	392
35.2. Ваше первое AJAX-приложение	393
РАЗДЕЛ 8. РАЗРАБОТКА ТИПИЧНОГО САЙТА	399
Глава 36. Создание простейшего движка сайта	401
36.1. Планирование движка	401
36.1.1. Зачем нужно разрабатывать собственный движок?	401
36.1.2. Необходимые нам функции движка	402
36.1.3. Принципиальная схема движка	404
36.2. Основные функции движка	405
36.2.1. Разработка TPL-шаблона	405
36.2.2. Файл настроек	406
36.2.3. Основной файл CMS — index.php	407
36.2.4. Проектирование базы данных	409
Таблица <i>static</i>	409
Таблица <i>cats</i>	410
Таблица <i>pages</i>	411
36.2.5. Иерархическая структура сайта	412
Алгоритм работы меню	412
Сценарий menu.php	414
Вывод содержимого раздела и страницы	419
36.3. Дополнительные функции движка	423
36.3.1. Вывод информации из таблицы <i>static</i>	423
36.3.2. Функция вывода содержимого HTML-файла	424
36.3.3. Версия для печати	425
36.4. Где взять листинги этой главы?	426
Глава 37. Создание фотогалереи	427
37.1. Постановка задачи	427
37.2. Загрузка изображений на сервер	427
37.3. Вывод галереи	431
Глава 38. Гостевая книга	434
38.1. Пережиток прошлого?	434
38.2. Разработка базы данных и структура гостевой книги	435
38.3. Вывод гостевой книги	435
38.4. Добавление записей в гостевую книгу	438
38.5. Сервисный сценарий gb_service.php	441
Глава 39. Импорт новостей с чужого сайта в базу данных WordPress	443
39.1. Постановка задачи	443
39.2. Вспомогательные функции	444
39.3. Вкратце о cURL	446
39.4. Основной сценарий	446
Глава 40. Создание счетчика сайта	451
40.1. Постановка задачи	451
40.2. Файл конфигурации	451

40.3. Разработка таблиц <i>counter</i> и <i>ipaddr</i>	452
40.4. Сценарий <i>counter.php</i>	453
40.5. Сценарий <i>reset_counter.php</i>	457
Глава 41. Статистика сайта	459
41.1. Методы сбора статистики	459
41.2. Программы-анализаторы журналов веб-сервера	460
41.3. Системы статистики	462
Глава 42. Голосования и комментарии для сайта	465
42.1. Разработка собственной системы голосования	465
42.1.1. Общая структура системы	465
42.1.2. Разработка сценария <i>poll_form.php</i>	466
42.1.3. Сценарий <i>poll_process.php</i>	468
42.1.4. Сценарий <i>poll_results.php</i>	469
42.2. Комментарии DISQUS	472
42.2.1. Описание платформы	472
42.2.2. Установка комментариев на свой сайт	473
РАЗДЕЛ 9. СЛОЖНЫЙ ПРОЕКТ	479
Глава 43. Постановка задачи	481
43.1. Концепция проекта	481
43.2. Алгоритм работы	482
43.3. Структура базы данных	483
Глава 44. Разработка основной части сайта	487
44.1. Структура проекта	487
44.2. Основной сценарий <i>index.php</i>	488
44.2.1. Подключение механизмов WordPress	488
44.2.2. Аутентификация пользователя	489
44.2.3. Форма добавления цели	490
44.2.4. Добавление цели	491
44.2.5. Просмотр всех целей	495
44.3. Сценарий <i>v.php</i> — просмотр цели	499
44.4. Сценарии отправки уведомлений. Сценарии <i>yes.php</i> и <i>no.php</i>	500
Глава 45. Панель управления WordPress	508
45.1. О чем эта глава?	508
45.2. Вход в панель управления	508
45.3. Просмотр целей	508
45.4. Не забывайте о UTF-8	510
РАЗДЕЛ 10. ОБЕСПЕЧЕНИЕ БЕЗОПАСНОСТИ САЙТА	511
Глава 46. SSL-соединения	513
46.1. Защищаем передаваемые данные	513
46.2. Настройка SSL в DirectAdmin	514
46.3. SSL-переменные	516

Глава 47. Защита PHP с помощью конфигурационного файла	518
47.1. Конфигурационный файл <code>php.ini</code>	518
47.2. Отключение потенциально опасных функций	520
47.3. Рекомендованные значения некоторых конфигурационных директив	520
Глава 48. Защита сайта от атак	521
48.1. Сайт в опасности	521
48.2. Два самых распространенных метода взлома	522
48.3. Межсайтовый скриптинг	522
48.4. SQL-инъекции	524
48.5. Флуд	528
48.6. Защита форума PHPBB2 от спамеров	529
РАЗДЕЛ 11. ПОЛЕЗНЫЕ СВЕДЕНИЯ	531
Глава 49. Визуальный редактор SPAW2 для вашего сайта	533
49.1. Знакомство с визуальным редактором и его базовая настройка	533
49.2. Адаптация редактора к вашему сайту	537
49.2.1. Загрузка в редактор текста статьи	537
49.2.2. Вывод редактора в TPL-шаблоне	537
49.2.3. Запрет изменения размера редактора	538
49.2.4. Изменение размеров редактора "на лету" — в процессе выполнения сценария	539
49.2.5. Панель инструментов редактора	539
49.2.6. Ошибка Error 105 (net::ERR_NAME_NOT_RESOLVED)	542
Глава 50. WAP-сервер средствами PHP	543
50.1. Есть ли необходимость в WAP?	543
50.2. Настройка веб-сервера Apache	544
50.3. PHP-сценарий	545
Глава 51. Пишем коммерческий сценарий	546
51.1. Zend Guard: за и против	546
51.2. Установка Zend Guard	548
51.3. Быстрая защита сценария	548
51.4. Ошибка Unable to locate XML file	552
51.5. Лицензии	552
51.6. Zend Guard — панацея?	556
ПРИЛОЖЕНИЯ	557
Приложение 1. Программа phpMyAdmin	559
П1.1. Действия над таблицами	560
П1.2. Создание новой таблицы. Изменение структуры таблицы	562
П1.3. Вставка записей	563
П1.4. Обзор таблицы	563
П1.5. Выполнение произвольного SQL-кода	563
П1.6. Резервное копирование БД	565
Приложение 2. Flash и PHP	566

Приложение 3. Оптимизация PHP-кода	570
ПЗ.1. Правильное использование <i>echo</i>	570
ПЗ.2. Длина переменных.....	571
ПЗ.3. Правильное обращение к элементам массива	571
ПЗ.4. Циклы.....	572
ПЗ.5. Чтение файлов.....	573
ПЗ.6. Вывод текста: <i>echo</i> против <i>printf</i>	575
ПЗ.7. "Лишние" вызовы функций.....	577
ПЗ.8. Контрольные замеры	578
ПЗ.8.1. Чтение файла.....	578
ПЗ.8.2. Вывод содержимого файла	579
ПЗ.9. Выводы	581
Приложение 4. Описание электронного архива.....	582
Предметный указатель	584

Предисловие

PHP (Hypertext Preprocessor) — один из самых популярных языков программирования, используемый для разработки веб-приложений. В настоящее время PHP поддерживается подавляющим большинством хостинг-провайдеров, что делает его чуть ли не основным языком, с помощью которого можно разработать любой интернет-проект, — от простенького сайта до крупного портала.

На PHP написано огромное количество как отдельных скриптов, так и завершенных проектов: форумов, систем управления контентом и др.

Что нового в 5-м издании?

По традиции, обновление книги началось с *первой главы*. Однако в рассказе о PHP версии 5.5 банальной заменой иллюстраций уже не обойтись. Теперь у программиста есть выбор: или использовать уже готовые пакеты программного обеспечения типа XAMPP¹ или Денвер², или же настраивать систему самостоятельно. Если раньше инсталлятор PHP делал всю необходимую работу, а пользователю нужно было лишь выбрать каталог, в который он устанавливал Apache, то теперь редактировать конфигурационные файлы веб-сервера придется вручную, потому что инсталлятор PHP для Windows упразднен и, похоже, что навсегда. Отсутствие инсталлятора обнаружилось в PHP версии 5.4, и это сразу усложнило процесс установки программного обеспечения. И если во время подготовки 4-го издания книги, когда версия 5.4 только-только вышла, казалось, что разработчики PHP просто не успели создать инсталлятор для Windows, то после выхода версии 5.5 стало понятно, что простого инсталлятора больше не будет. Именно поэтому в *первой главе* описан процесс установки PHP версии 5.5 во всех подробностях.

¹ XAMPP — кроссплатформенная сборка веб-сервера, содержащая Apache, MySQL, интерпретатор скриптов PHP, язык программирования Perl и большое количество дополнительных библиотек, позволяющих запустить полноценный веб-сервер.

² Денвер (от сокр. Д.н.в.р или ДНВР — джентльменский набор Web(веб)-разработчика) — набор дистрибутивов и программная оболочка, предназначенные для создания и отладки сайтов (веб-приложений, прочего динамического содержимого интернет-страниц) на локальном ПК (без необходимости подключения к сети Интернет) под управлением ОС Windows.

В *главе 8* читатели найдут продолжение эпопеи с директивой `register_globals`. Если мне не изменяет память, то, начиная с РНР версии 4.3, она выключена, а в версии 6 (которая до сих пор не вышла) ее собирались упразднить. Однако на дворе 2014 год, вышла версия РНР 5.5, а директива `register_globals` все еще присутствует в конфигурационном файле `php.ini`. (Честно говоря, я думал, что в версии 5.5 от нее наконец-то избавятся, и в *главе 8* информацию о ней можно будет заменить чем-то другим.)

Из *главы 9* вы узнаете о том, что такое *разыменовывание массива*, как использовать `list()` внутри `foreach`, будет также приведено сравнение функций `var_dump()` и `print_r()` для анализа содержимого массива.

В *новой главе 13* рассмотрено *разыменовывание строки* (впрочем, эта процедура нам не очень полезна и вызывает разве что академический интерес), а также API хэширования паролей (Password Hashing API).

О поддержке нового графического формата, разработанного компанией Google, мы поговорим в *главе 15* — при рассмотрении библиотеки GD.

В *главе 17* будут рассмотрены функции-генераторы и приведены особенности использования функции `empty()`.

В *главе 24* вы познакомитесь с новыми служебными словами: `finally` и `class`. Первое служит для обработки исключений, а второе — для вывода имени класса.

Существенные изменения внесены в *главу 29*. Дело в том, что в РНР 5.4 появился механизм контроля процесса загрузки файлов, позволяющий создавать индикаторы загрузки практически средствами РНР, не прибегая к сторонним средствам. Конечно, "не прибегая к сторонним средствам" — это сказано сильно. Вам все равно придется использовать jQuery для обращения к этому механизму, но не будем забегать вперед — обо всем этом вы и узнаете из *главы 29*. Надеюсь, я вас заинтриговал...

Полностью заменена *глава 39*. Теперь вместо безнадежно устаревшего и неактуального примера интеграции в свой сайт галереи LiveJournal она посвящена импорту новостей с чужого сайта в базу данных WordPress.

Остальные главы книги также подверглись основательному пересмотру, как с точки зрения повышения их удобочитаемости, так и в плане приведения всей терминологии к современному звучанию.

Электронный архив с информацией, расширяющей и дополняющей материал "бумажной" книги, можно скачать с FTP-сервера издательства по ссылке: **ftp://ftp.bhv.ru/9785977535144.zip**, со страницы книги на сайте **www.bhv.ru** или с сайта автора: **http://dkws.org.ua/mybooks/php_adds.zip**. Архив этот также обновлен и теперь содержит дополнительные материалы к 4-му и 5-му изданиям книги, включая новое программное обеспечение (см. *приложение 4*).

Немного истории

В 1994 году датский программист Расмус Лердорф (Rasmus Lerdorf) создал надстройку (набор скриптов) над Perl/CGI для вывода и учета посетителей своего сайта и назвал ее Personal Home Page (отсюда и произошло первоначальное название — РНР).

Но Perl — довольно медленный интерпретатор, и вскоре его производительности перестало хватать, поэтому разработчик написал на языке C новый интерпретатор и назвал его PHP/FI (Personal Home Page / Form Interpreter). В новом интерпретаторе четко прослеживались черты Perl — например, символ доллара в начале имени переменной. Тогда же были заложены черты современного PHP, такие как автоматическая обработка форм, встраиваемость в HTML и др.

В 1997 году появилась вторая версия интерпретатора — PHP/FI 2.0. Она была установлена примерно на 50 тыс. серверов. С одной стороны, цифра внушительная, но это всего лишь 1 % от общего числа интернет-серверов на тот момент.

В 1998 году появился PHP 3.0. Именно с этой версии аббревиатура "PHP" стала рекурсивным акронимом PHP — Hypertext Preprocessor. Третья версия пользовалась существенно бóльшим успехом, чем вторая, — она была установлена на 10 % всех интернет-серверов, а это уже не мало!

PHP 3.0 частенько критиковали за медленное ядро, и действительно — сценарии выполнялись медленно. Поэтому разработку версии PHP 4 начали с переработки ядра. Начиная с PHP 4, интерпретатор сначала транслирует PHP-код во внутреннее представление, а потом выполняет (а не выполняет сценарий строка за строкой, как в PHP 3), благодаря чему существенно повысилась производительность сценариев. Четвертая версия PHP появилась в мае 2000 года. Обновления для этой версии PHP выпускались до конца 2007 года, но с августа 2008 года поддержка PHP 4 была прекращена.

Пятая версия PHP появилась в июле 2004 года. Она отличалась переработанным ядром Zend, что увеличило эффективность интерпретатора, добавлением поддержки XML, переработанной поддержкой объектно-ориентированного программирования (ООП). С этого момента PHP стал полноценным объектно-ориентированным языком программирования, а его объектная модель во многом схожа с моделью Java. Несмотря на наличие версии 5.4, все еще актуальной остается версия 5.3. Далеко не все хостинг-провайдеры успели перейти на новую версию PHP, поэтому как минимум еще год на хостингах версия 5.3 будет по-прежнему встречаться.

PHP 5.5, или что случилось с PHP 6?

Разработка шестой версии PHP началась в октябре 2006 года. В ней было сделано множество нововведений.

Прежде всего, в шестой версии значительно улучшена поддержка Unicode¹, что чрезвычайно важно. Раньше, когда в знаковый генератор можно было загрузить только 256 символов, поддержка разных языков оказывалась существенно ограниченной, вследствие чего и появилось множество разных кодировок — иногда даже по несколько для одного и того же языка. Русский язык не стал исключением. Для

¹ Получить список всех функций, поддерживающих Unicode, можно по адресу http://www.php.net/~scoates/unicode/render_func_data.php.

него было разработано три кодировки: CP866 (DOS), Windows-1251 (или CP-1251, Windows) и KOI8-R (UNIX/Linux). Использование разных кодировок всегда поднимало проблему их конвертирования — например, символ "Б" во всех этих кодировках имеет разные коды.

Первая версия Unicode, представленная в 1991 году, позволяла хранить 65 536 символов. Текущая версия Unicode позволяет хранить 2 147 483 648 символов, чего вполне достаточно для хранения символов всех языков и, в частности, кириллицы. Так что, когда все программное обеспечение будет переведено на Unicode (а это произойдет в ближайшем будущем), проблема несовместимости кодировок исчезнет¹.

Второе существенное усовершенствование PHP 6 — это поддержка пространств имен, что позволяет избежать коллизий между именами функций и классов. Подробно о пространствах имен мы поговорим в *главе 24*.

Кроме того, в PHP 6 обеспечивается "родная" (без использования дополнительных классов) поддержка SOAP и XML. Правда, рассмотрение SOAP и XML выходит за рамки этой книги, но все же вы должны знать, что такая поддержка есть уже на уровне PHP.

Вместе с тем некоторые функции из новой версии были удалены. В частности, из состава PHP 6 исключены следующие директивы (и выполняемые ими функции):

- ❑ `magic_quotes` — управляла экранированием кавычек;
- ❑ `register_globals` — регистрировала глобальные переменные;
- ❑ `register_long_arrays` — регистрировала длинные массивы;
- ❑ `safe_mode` — безопасный режим, но на самом деле он не делал безопаснее сценарии, поэтому его и устранили.

Исключение этих директив позволило сделать PHP-сценарии более защищенными. Не спорю, с включенными директивами `magic_quotes` и `register_globals` программировать было несколько удобнее, и разработчики очень часто жертвовали безопасностью ради комфорта. Сейчас же наоборот — безопасность на первом месте.

Но шестой версии PHP так и не суждено было увидеть свет. Камнем преткновения стала как раз поддержка Unicode — от нее решили отказаться, а все наработки вынесли в отдельную ветку, которая вскоре превратилась в версию 5.4. И даже сейчас, когда у нас уже вовсю течет 2014 год, версия 6 так и не появилась на горизонте. Вместо нее 20 июля 2013 года вышла версия 5.5, которая и стала поводом для пятого издания этой книги.

Далее представлен список нововведений в версиях 5.4 (хотя с момента выхода версии 5.5 прошло уже более полугода, версия 5.4 остается актуальной, и в большинстве случаев на серверах хостинг-провайдеров установлена именно она) и 5.5.

¹ Подробно о Unicode вы можете прочитать по адресу <http://ru.wikipedia.org/wiki/Unicode>.

Основные нововведения в PHP версии 5.4

Типажи

Существенно упрощают мультинаследование *типажи* (traits). Сейчас мы не будем подробно их рассматривать, поскольку без знания объектно-ориентированного программирования читатель вряд ли что поймет. Скажу только, что это нововведение описано в *главе 24*. Там же описаны дополнительные новшества, появившиеся в PHP 5.4 и связанные с ООП.

Разыменовывание массива

Поддерживается *разыменовывание массива*. Представим, что у нас есть метод, возвращающий массив, например:

```
function foo() {  
    return array(1, 2, 3);  
}
```

Функция возвращает массив из трех элементов. Как обратиться, скажем, к нулевому элементу? Раньше это делалось так:

```
$res = foo();  
echo $res[0];
```

Теперь можно сделать так:

```
echo foo()[0];
```

Встроенный веб-сервер

Фактически теперь PHP-разработчикам не нужен Apache, поскольку в PHP 5.4 встроен собственный веб-сервер, запустить который можно из консоли:

```
php -S localhost:8080 -t /htdocs
```

В данном случае мы "повесили" сервер на порт 8080, а каталог /htdocs станет корневым каталогом документов, — именно в этом каталоге наш сервер будет искать документы. Конечно, этот сервер очень простой, но он лучше, чем ничего, когда нет возможности установить другой сервер.

Индикатор загрузки файла

Наверное, все вы видели индикаторы загрузки файла на сервер. Как такой сделать? В PHP 5.4 это очень просто. Теперь информация о процессе загрузки файла хранится в сессии пользователя — например, в ключе `upload_progress`. Далее дело техники — можно получить значение этой переменной и отобразить его пользователю (или преобразовать в графический индикатор). В этом вам поможет Ajax (чтобы изменять отображаемое значение "на лету" без перезагрузки страницы).

Конечно, это не все новшества версии 5.4, об остальных вы узнаете по ходу чтения книги.

Основные нововведения в PHP версии 5.5

Генераторы

Генератор — это функция, генерирующая последовательность значений. О том, как использовать генераторы, мы поговорим в *главе 17*.

Ключевые слова *finally* и *class*

Теперь в исключениях (блоках `try`) можно использовать слово `finally`, которого так не хватало многим PHP-программистам. О том, как использовать `finally`, мы поговорим, когда будем рассматривать исключения, а именно в *главе 24*. В этой главе мы рассмотрим и другое новое ключевое слово — `class`.

API хэширования паролей

В версии 5.5 появился механизм хэширования паролей — Password Hashing API. Его возможности будут рассмотрены в *главе 13*.

Разыменовывание строк

В версии 5.4, как уже отмечалось, появилась возможность разыменовывания массивов, а в версии 5.5 — возможность разыменовывания строк. Если вы читали предыдущее издание, то уже догадываетесь, как будет работать новый механизм. Если же нет, тогда загляните в *главу 13*.

Изменения в работе конструкции *foreach*

Теперь конструкция `foreach` может работать с ключами не скалярного типа. В `foreach` появилась также возможность использовать конструкцию `list()`. Эти нововведения будут рассмотрены в *главе 9*.

Поддержка графического формата WebP

Формат WebP — это формат сжатия изображений с потерями и без потерь качества, предложенный компанией Google в 2010 году. Теперь поддержка этого формата появилась и в PHP, а рассмотрена она будет в *главе 11*.

Исключенные возможности

Из PHP 5.5 исключены следующие возможности:

- ☐ удалены функции, касающиеся PHP Logo: `php_logo_guid()`, `php_egg_logo_guid()`, `php_real_logo_guid()`, `zend_logo_guid()`;
- ☐ в функции `preg_replace()` больше не поддерживается модификатор `/e`;

- ❑ больше не поддерживаются операционные системы Windows XP и Windows 2003. Чтобы установить PHP 5.5, вам придется обновить свою операционную систему до Windows 7 или 8;
- ❑ расширение `mysql` использовать не рекомендуется, вместо него лучше обратиться к `mysqli`.

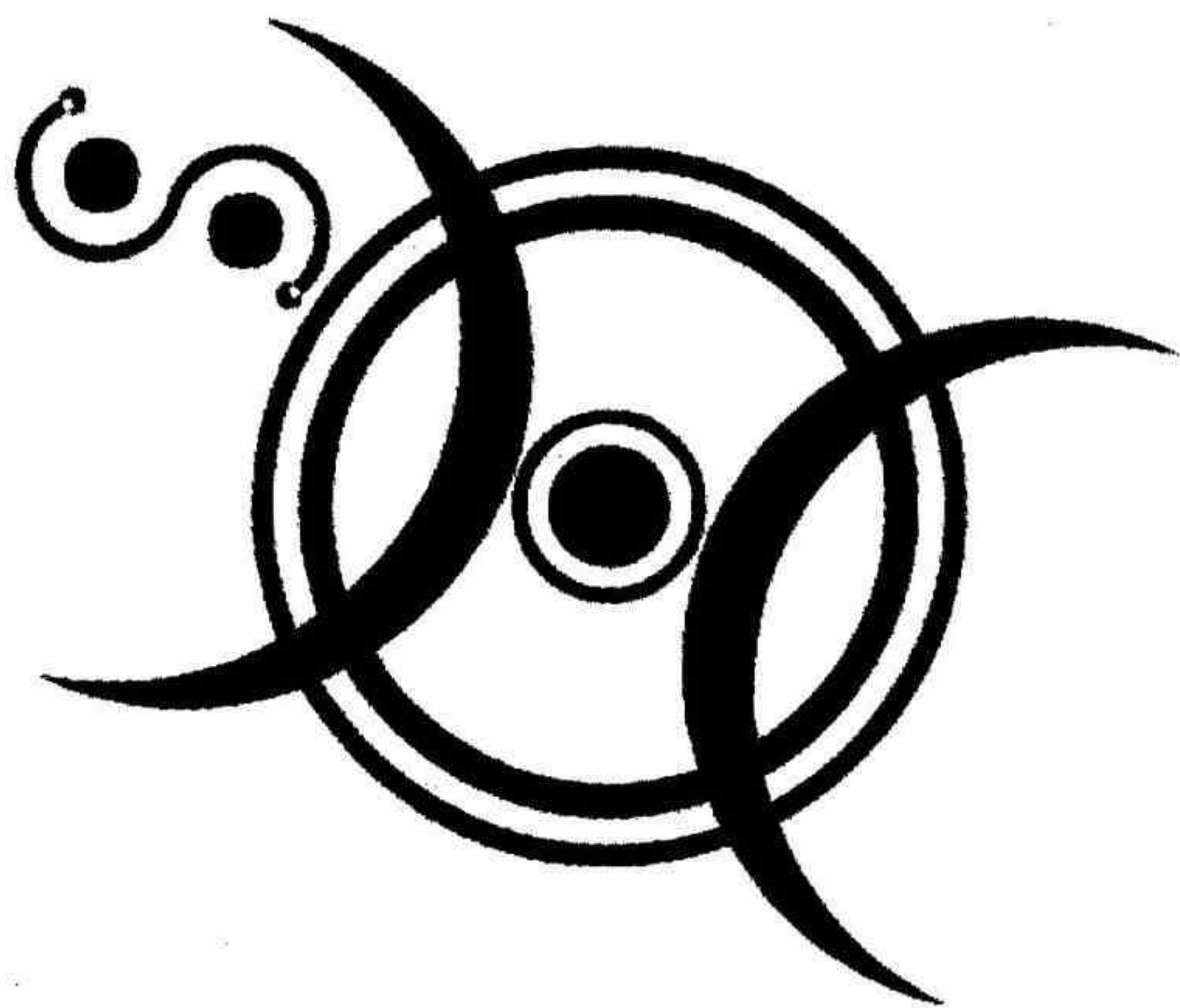
На мой взгляд, некоторые изменения в версии 5.5 вызывают вопросы. Как бы и изменения есть, но пользы от них нет. Конечно, есть и полезные нововведения — например, генераторы, ключевое слово `finally`, но что касается разыменовывания строк и массивов, а также использования функции `empty()` над другой функцией (см. главу 17), то пользы от таких новшеств мало. Но обо всем по порядку. В соответствующих главах будут рассмотрены те или иные нововведения и рассказано, почему одни из них представляются полезными, а другие — нет.

MySQL

Ни один серьезный веб-проект не обходится без базы данных — ведь хранение данных в файлах давно считается пережитком прошлого, хотя до сих пор можно встретить проекты, работающие по старинке. PHP поддерживает много разных баз данных, но стандартом де-факто является сервер баз данных MySQL. В настоящее время доступна уже шестая версия этого популярного сервера. Кроме MySQL мы здесь рассмотрим еще одну альтернативную систему управления базами данных — SQLite. Почему именно SQLite? На заре 5-й версии PHP ходили слухи, что новая (т. е. 5-я на тот момент) версия PHP из-за лицензионных разногласий не будет вообще поддерживать MySQL. Поэтому возникла необходимость в легком и простом сервере баз данных. Им и стал SQLite. К счастью, поддержка MySQL из PHP 5 удалена не была, и последние версии PHP полностью поддерживают MySQL.

Поддержка читателей

Если при чтении книги у вас возникнут какие-либо вопросы, вы всегда можете задать их на форуме сайта www.dkws.org.ua.



ЧАСТЬ I

ТЕОРИЯ



НАСТР. I

ТЕОРИЯ

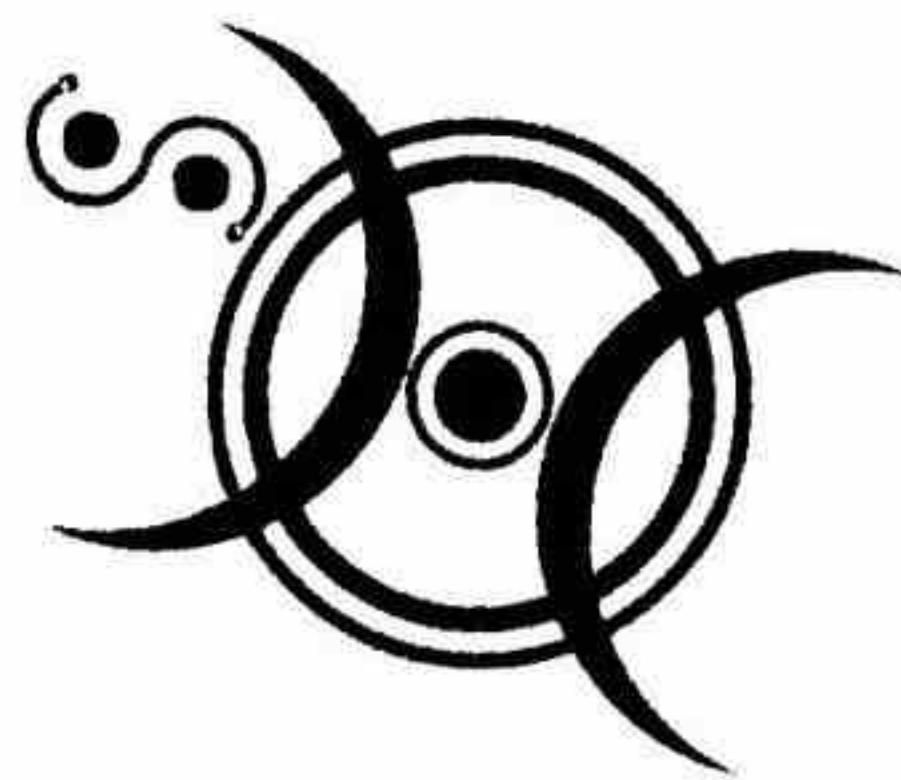


I. РАЗДЕЛ 1

Быстрый старт

Глава 1.	Установка необходимого программного обеспечения
Глава 2.	Программа на PHP
Глава 3.	Основы синтаксиса PHP

ГЛАВА 1



Установка необходимого программного обеспечения

1.1. Нужно ли устанавливать программное обеспечение?

Подготавливая книгу к печати, я вообще сомневался в целесообразности включения в нее такой главы. Почему? Если вы купили эту книгу, значит, со временем собираетесь профессионально заниматься PHP (разрабатывать сайты, сценарии) или же планируете создать собственный сайт на PHP. Следовательно, вам понадобится хостинг с поддержкой PHP. Учитывая, что PHP-хостинг сейчас стоит очень дешево, как и безлимитный Интернет, то все, что требуется для PHP-разработки, — это какой-нибудь текстовый редактор с поддержкой PHP-синтаксиса.

Ведь вам всего-то необходимо написать сценарий. В свою очередь, сценарий — это обычный текстовый файл. Как только сценарий будет готов, его нужно загрузить по FTP на ваш хостинг и запустить через браузер. При этом совсем не имеет значения, какую операционную систему вы используете: Windows, Linux или даже Mac OS.

Ранее имело смысл устанавливать на свой компьютер связку Apache + PHP + сервер баз данных MySQL. Сейчас же можно производить разработку и отладку сценариев непосредственно на сервере. Так даже правильнее, поскольку конфигурации Apache и PHP на локальном и удаленном сервере в 99 % случаев окажутся различны, поэтому при переносе сценариев могут возникнуть проблемы. Какие именно? Начиная с самых безобидных, вроде отличающихся параметров базы данных MySQL, до разницы в версии PHP. С параметрами базы данных все просто — при переносе сценария с одного сервера на другой не забывайте проверять параметры доступа, иначе ваши сценарии откажутся работать. Чуть позже мы продемонстрируем пример, позволяющий легко избавиться от этой проблемы. А вот с версией PHP и установленными PHP-расширениями проблемы могут возникнуть глобальные. Помню, у меня был отлично работающий проект, отправляющий прайс-листы клиентам одной из поддерживаемых мною организаций. Сценарий был предельно прост — пользователь загружал новый прайс-лист, а сценарий отправлял его всем клиентам, адреса которых были в базе данных. Для отправки сообщений использовался класс `PHPMailer`. Но после переноса на другой сервер сценарий перестал за-

пускаться, причем даже не выдавал сообщение об ошибке. Для выявления причины такого нештатного поведения сервера первым делом я запустил функцию `phpinfo()`:

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

Результат меня просто поразил — оказалось, что на сервере установлена четвертая (!) версия PHP. Учитывая существование версии 5.5, было странно, что кто-то еще пользуется таким антиквариатом, как PHP 4. Решить проблему удалось путем загрузки класса `PHPMailer` для PHP 4. Хорошо, что таковой до сих пор еще сохранился на серверах разработчиков!

Иная распространенная проблема — отсутствие на другом сервере необходимых вам расширений. Например, вы создали галерею картинок, использующую функции из библиотеки GD. Но эта галерея не будет работать, если на сервере не установлено расширение GD (PHP-библиотека GD). Поэтому, прежде чем размещать сценарий на сервере, вызовите функцию `phpinfo()` и убедитесь, что на сервере установлена нужная версия PHP и нужные расширения.

Как узнать, под какую версию PHP готовить сценарии? Раньше на серверах могла быть установлена или одна из ранних версий PHP 5, или проверенная временем четвертая версия. Но сейчас PHP 4 на сервере за редким исключением уже не встретишь — все перешли на PHP 5.

Однако не забывайте, что в каждой версии PHP: 5.1, 5.2, 5.3, 5.4 и 5.5 — имеются свои изменения. Например, новые параметры у функций и новые функции. Для изучения PHP, по большому счету, все равно, какую взять версию. Главное, чтобы вы знали, какая она именно, дабы не возникало потом вопросов, почему та или иная функция не работает, как следует.

Впрочем, на самом деле все не так страшно. Большинство функций будут работать во всех версиях PHP 5, так что вы не испытаете никаких проблем. Исключение могут составить разве что версии 5.4/5.5. По сути, это версия 6.0 (которая так и не вышла), но без поддержки Юникода.

Пока хостинг-провайдеры не спешат устанавливать версию 5.4 (не говоря уже о 5.5). Так, на одном из моих хостингов стоит версия 5.3, а на другом — вообще 5.2, хотя версии 5.4/5.5 уже давно стабильны.

Так как же быть с самыми последними версиями: 5.4 и 5.5? Поскольку есть релиз версии 5.5, то изучать ее стоит. Однако будьте пока готовы, что сейчас далеко не все хостеры используют 5.5, но со временем переход обязательно состоится. В настоящее время вы можете загрузить PHP 5.5 на свой компьютер и изучить новые возможности этой версии.

Итак, если вы не собираетесь устанавливать веб-сервер Apache, интерпретатор PHP и сервер баз данных на свой компьютер, вам понадобятся две программы: текстовый редактор и FTP-клиент.

Мы уже отмечали, что, начиная с версии 5.4, у PHP имеется свой встроенный веб-сервер, но я рекомендую все же использовать Apache, поскольку в большинстве

случаев на сервере хостера будет установлен именно этот сервер, так что условия окажутся ближе к "боевым".

1.2. Выбор PHP-редактора и FTP-клиента

Из всех PHP-редакторов, которыми я пользовался, мне более всех понравились два редактора: PHP Expert Editor и Zend Studio (вы без проблем найдете сайты разработчиков этих редакторов). Оба редактора коммерческие, но они стоят своих денег. PHP Expert Editor (рис. 1.1) обеспечивает не только подсветку синтаксиса, но и умеет проверять синтаксис сценария. Правда, для проверки синтаксиса вам нужно установить PHP (об этом позже) и указать его в настройках программы.

ПРИМЕЧАНИЕ

Условно-бесплатную версию программы PHP Expert Editor вы найдете в каталоге *software* сопровождающего книгу электронного архива (см. приложение 4). Последнюю версию программы также всегда можно скачать с сайта разработчиков: <http://phpexperteditor.com/>.

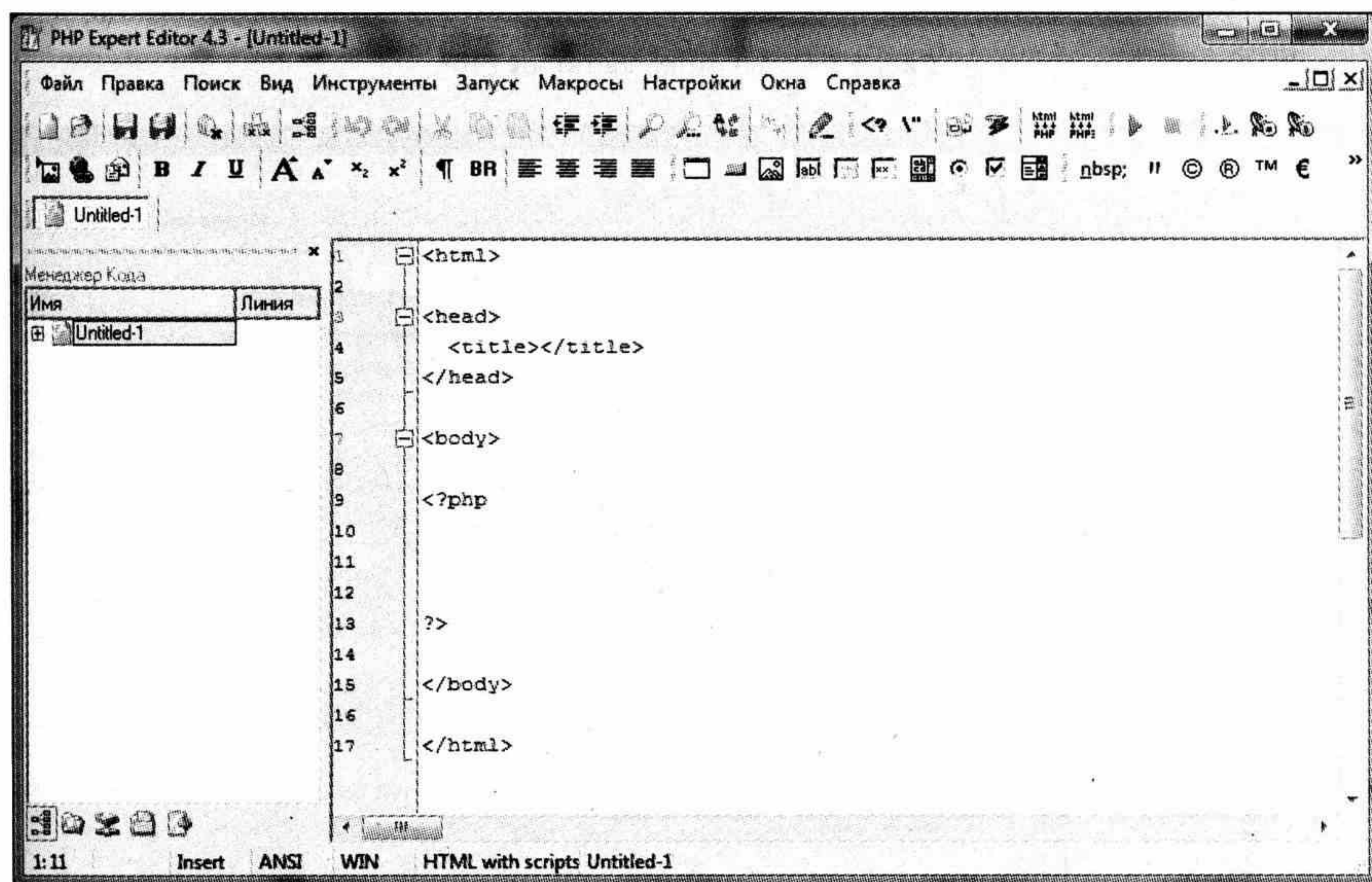


Рис. 1.1. Программа PHP Expert Editor

Чтобы указать PHP в настройках программы PHP Expert Editor, выполните команду **Запуск | Настройки**, в открывшемся окне перейдите на вкладку **Интерпретаторы скриптов** (рис. 1.2) и укажите путь к интерпретатору PHP. Для проверки синтаксиса нажмите клавишу <F7>.

Возможности Zend Studio (рис. 1.3) куда шире. Чего стоит только автодополнение кода, когда программа "читает" ваши мысли и дописывает за вас PHP-код. Если вы

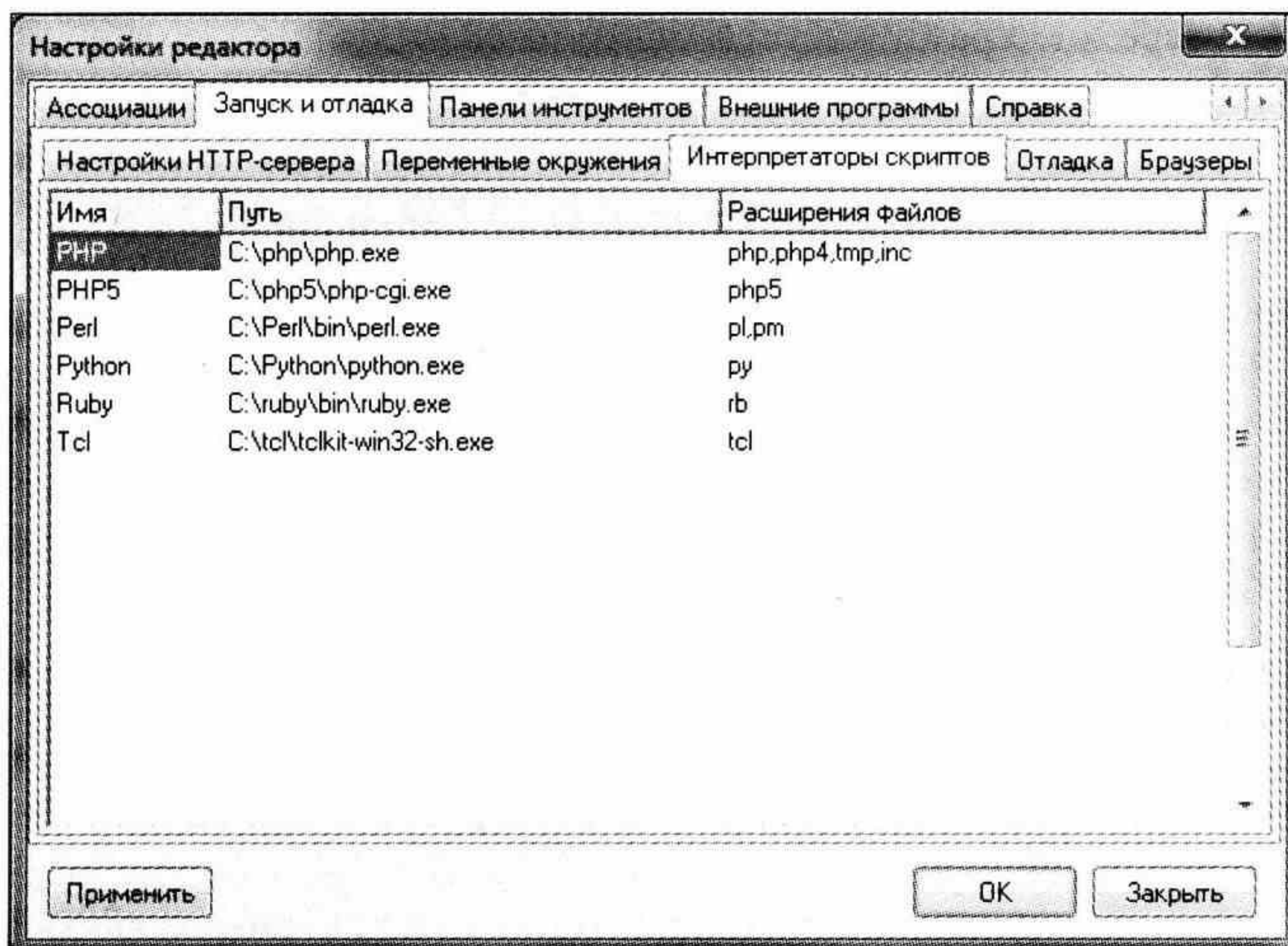


Рис. 1.2. Путь к интерпретатору PHP

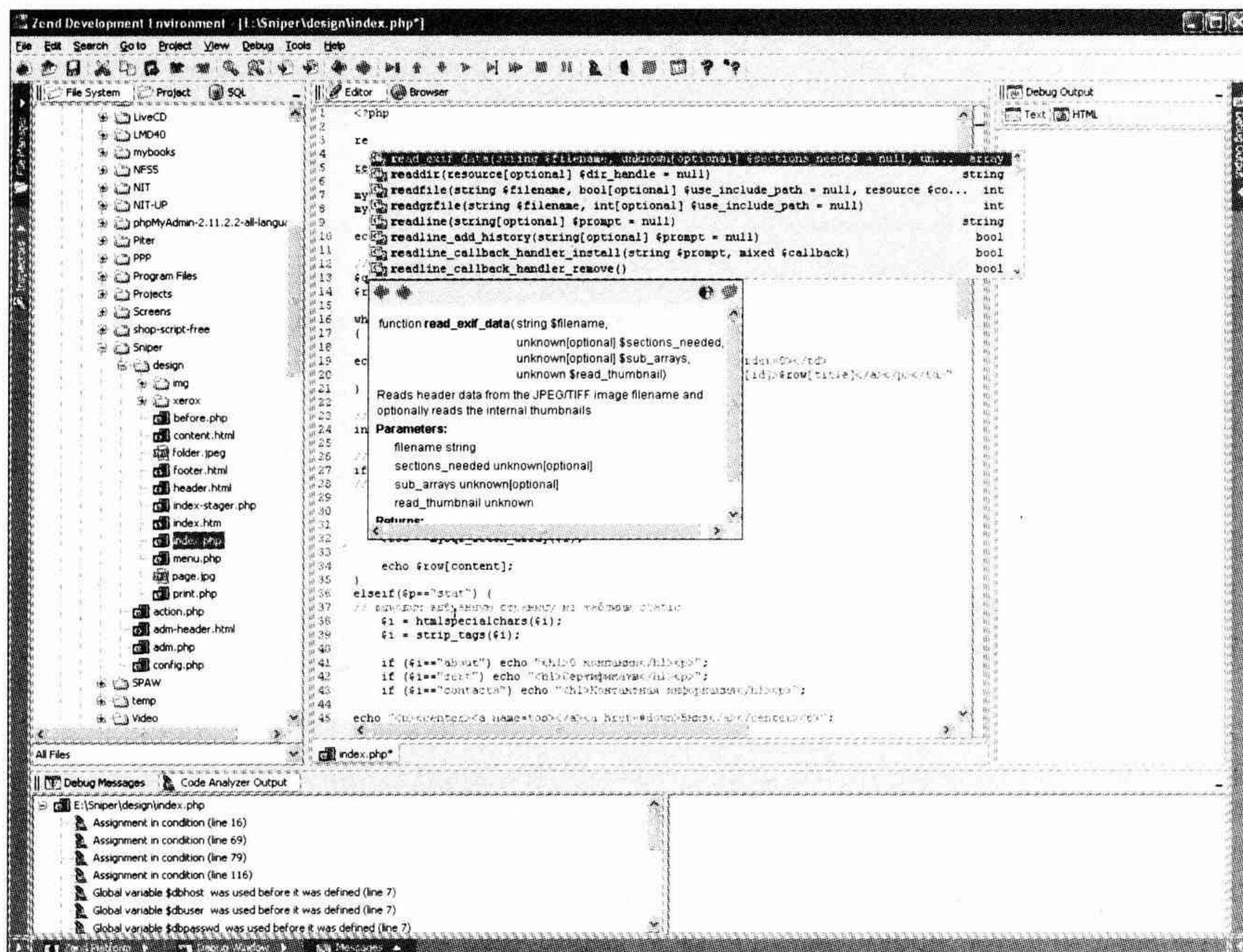


Рис. 1.3. Zend выводит не только имена функций, но и краткую справку по каждой функции

работали с Delphi и другими продуктами Borland, то поймете, о чем я говорю. Поверьте, очень удобно.

В качестве FTP-клиента могу порекомендовать бесплатную программу FileZilla (рис. 1.4), которая доступна как для Windows, так и для Linux.

ПРИМЕЧАНИЕ

Версия 3.5.3 FTP-клиента FileZilla находится в каталоге *software* сопровождающего книгу электронного архива (см. приложение 4).

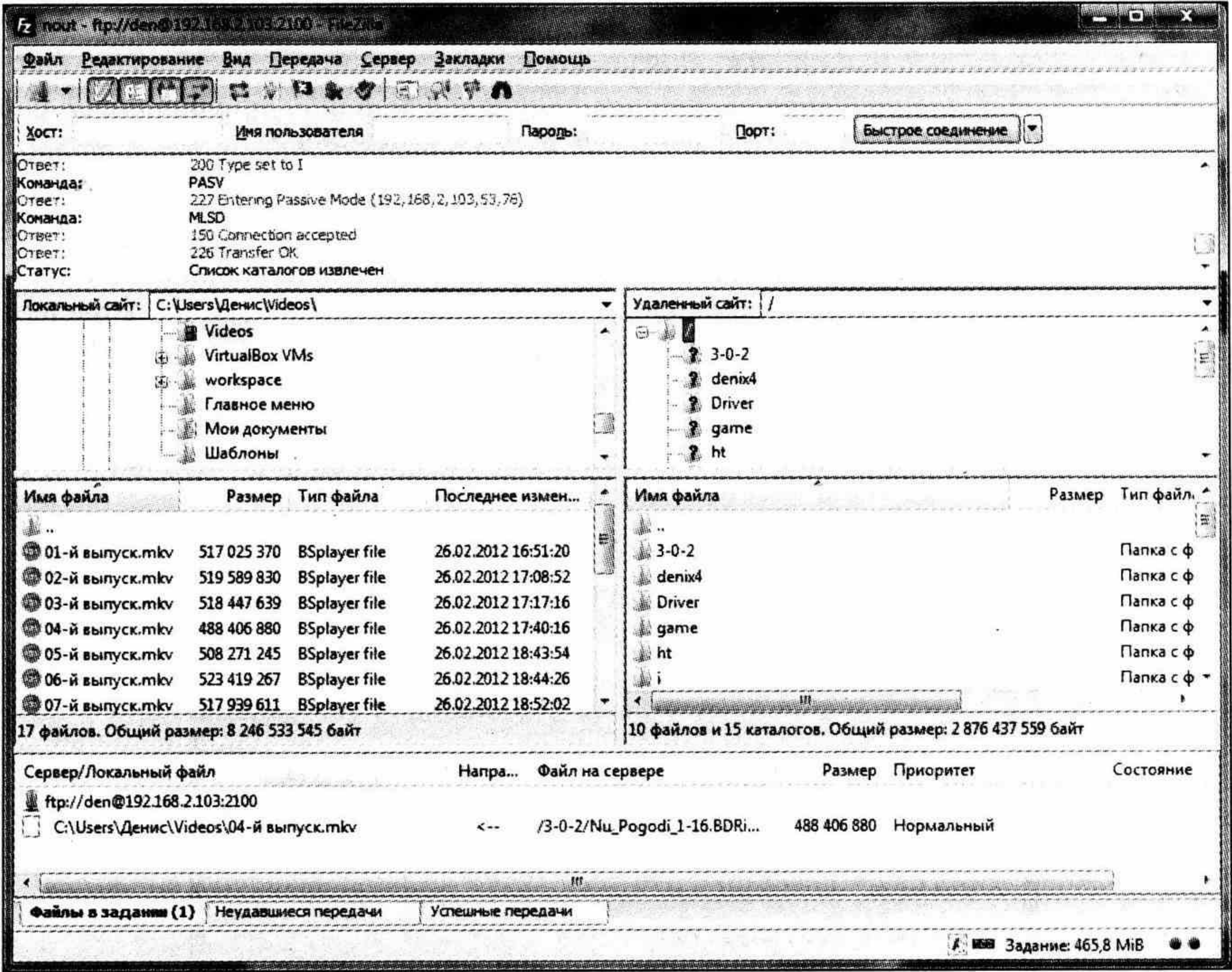


Рис. 1.4. Программа FileZilla

1.3. Установка связки Apache + PHP + MySQL в Windows

1.3.1. Для опытных пользователей: установка вручную

Если вы все-таки решили настроить собственный сервер, приступим к его установке. Первым делом вам нужно скачать с сайта www.apache.org веб-сервер Apache —

это файл `httpd_2.2.X-win32-x86-no_ssl.msi`. Запустите скачанный файл. Процесс установки обычно проходит без осложнений, но мы все же его рассмотрим.

ПРИМЕЧАНИЯ

Версия 2.4 доступна пока только для пользователей UNIX, поэтому Windows-пользователям придется довольствоваться версией 2.2.

В каталоге *software* сопровождающего книгу электронного архива (см. приложение 4) вы найдете последнюю версию Apache для Windows. Linux-версии в архиве нет, поскольку Apache входит в состав любого дистрибутива Linux, и вам нужно будет только установить его. В архив включены версия Apache 2.2.25 (файл `httpd_2.2.X-win32-x86-no_ssl.msi`), а также модуль *mod_ftp* для Windows (файл `mod_ftp-0.9.6-beta-win32-x86.zip`). Несмотря на наличие в архиве установочного файла Apache, не забудьте посетить страницу <http://archive.apache.org/dist/httpd/binaries/win32/> — вполне возможно, что к моменту покупки книги вышла новая версия Apache.

ОСОБЕННОСТИ УСТАНОВКИ В WINDOWS VISTA И WINDOWS 7

Установку Apache, PHP и MySQL в Windows Vista и в Windows 7 желательно производить от имени администратора. Для этого щелкните на установочном файле правой кнопкой мыши и выберите команду **Запуск от имени администратора**.

Важно правильно указать информацию о сервере: доменное имя сети, доменное имя компьютера (если вы не планируете использовать сервер во всей локальной сети, лучше установить `localhost`) и адрес администратора (рис. 1.5). Также вам нужно выбрать способ запуска сервера:

- ☐ **for All Users** — сервер будет установлен как Windows-сервис и станет доступен всем пользователям;
- ☐ **only for the Current User** — сервер нужно будет запускать вручную, и он станет использовать порт 8080, т. е. при обращении к серверу придется указывать этот порт в URL — например: **`http://localhost:8080/`**.

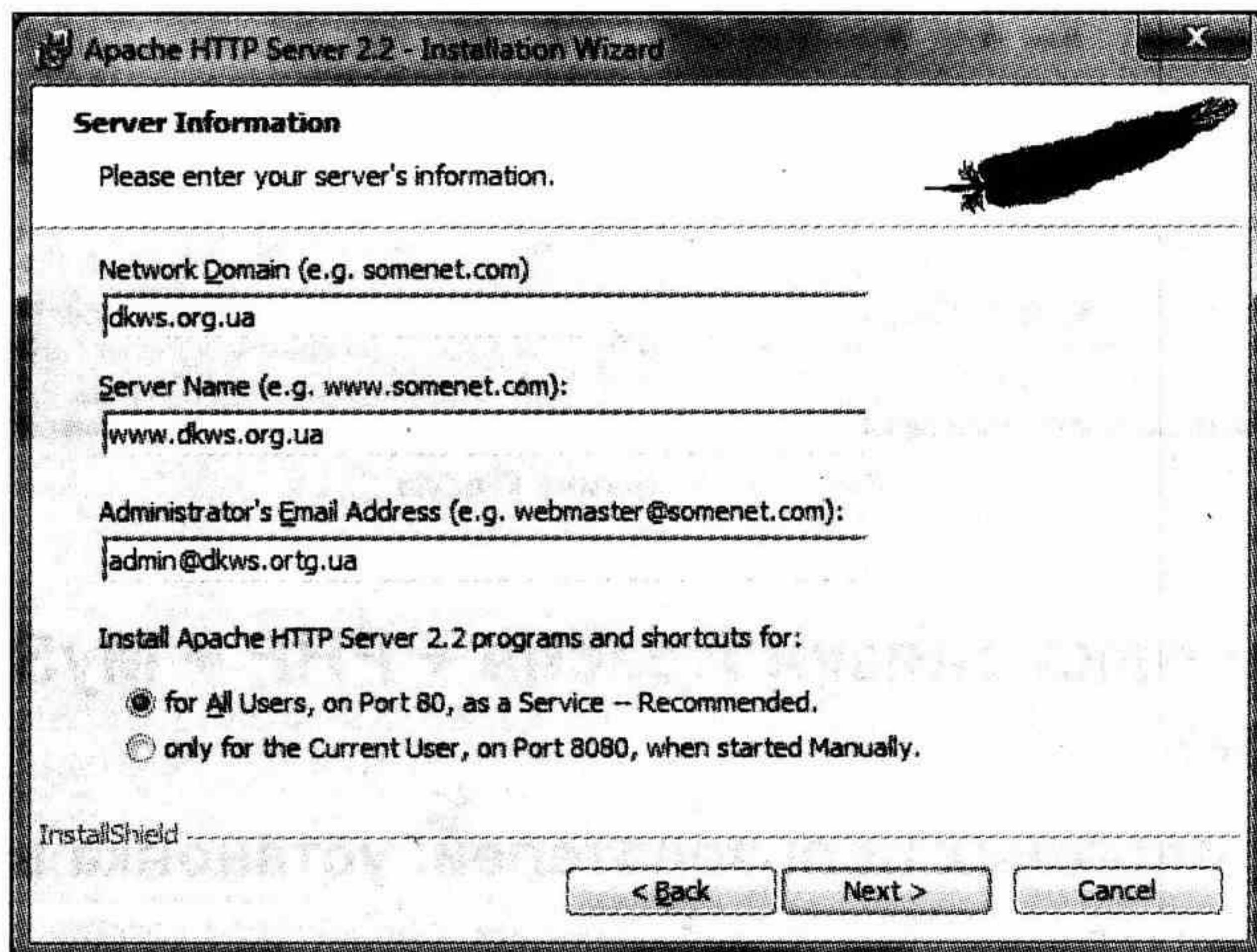


Рис. 1.5. Параметры сервера

Затем выберите тип установки **Custom** и измените каталог установки Apache (рис. 1.6). По умолчанию используется каталог C:\Program Files\Apache Software Foundation\Apache2.2\, но намного проще установить сервер в каталог C:\WWW или D:\WWW. Нажав кнопку **Space**, вы можете посмотреть, сколько свободного места осталось на ваших дисках.

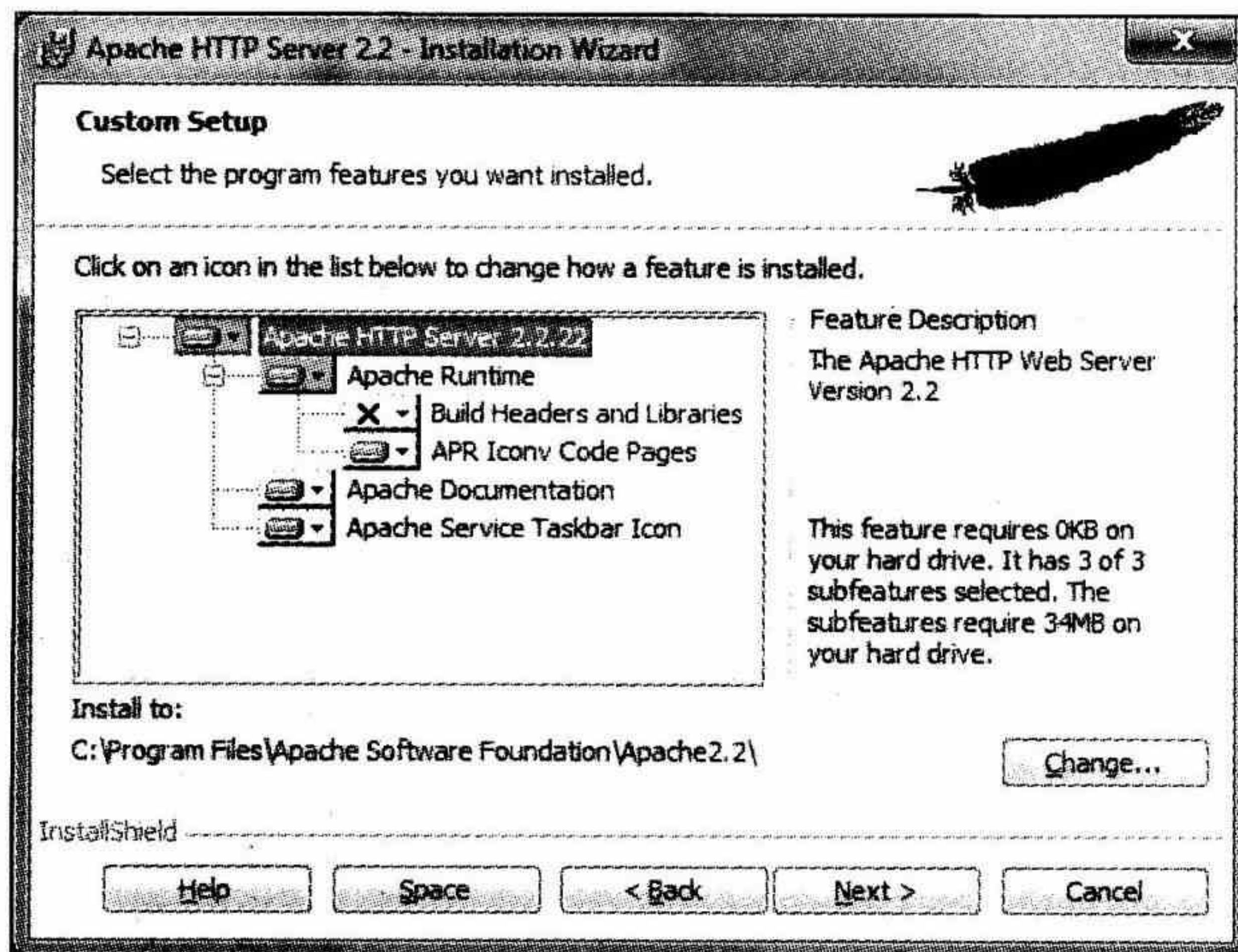


Рис. 1.6. Каталог для установки Apache

После установки Apache запустите браузер и введите следующий URL:

`http://localhost`

Если в ответ вы увидите сообщение **It works!** (рис. 1.7), значит, установка Apache прошла успешно.

По умолчанию сервер Apache не поддерживает PHP, поэтому PHP придется установить отдельно. Ранее (до версии 5.4) в PHP был доступен инсталлятор, который существенно упрощал процесс установки PHP, помогал внести все необходимые изменения в конфигурационные файлы Apache и изменить параметры Windows. Сейчас все это придется делать вручную. Если вам этим заниматься не с руки, удалите Apache и установите пакет XAMPP, где в одном флаконе содержится все необходимое: Apache, PHP и MySQL, и вам вручную ничего настраивать не придется.

PHP 5.5

В каталоге *software* сопровождающего книгу электронного архива (см. приложение 4) вы найдете файл `php-5.5.9-Win32-VC11-x86.zip` с последней (5.5.9) версией PHP на момент написания этих строк.

Итак, распакуйте архив с PHP (упомянутый ранее файл `php-5.5.9-Win32-VC11-x86.zip`) в каталог C:\php (рис. 1.8).

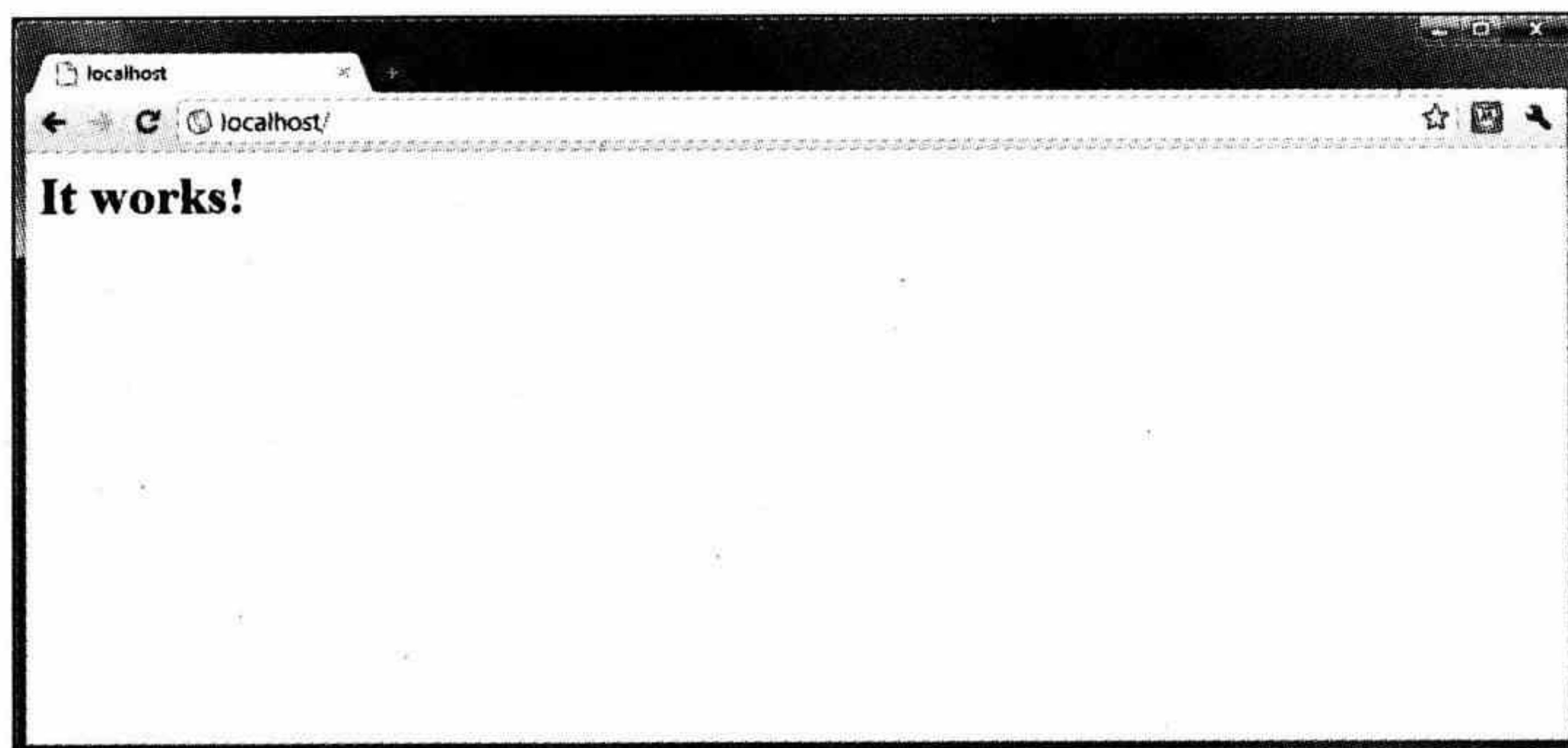


Рис. 1.7. Веб-сервер работает!

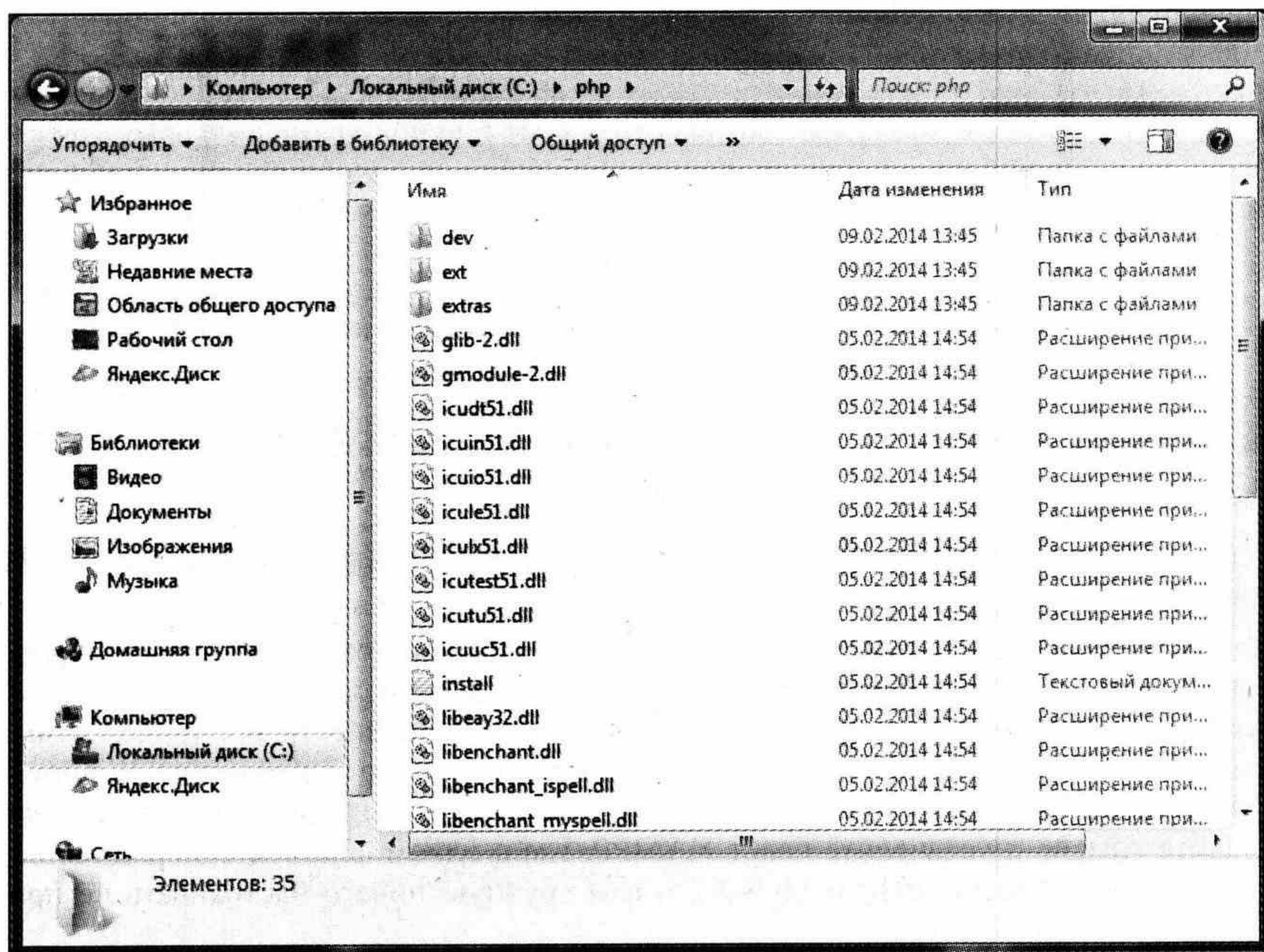


Рис. 1.8. PHP 5.5.9 распакован в каталог C:\php

После этого добавьте каталог C:\php в путь поиска приложений. Для этого выполните следующие действия:

1. Откройте меню **Пуск** и щелкните правой кнопкой мыши на элементе **Компьютер**. Выберите команду **Свойства**.

2. В окне **Система** на панели слева выберите команду **Дополнительные параметры системы**.
3. Нажмите кнопку **Переменные среды**.
4. В окне **Переменные среды** выберите переменную **Path** и нажмите кнопку **Изменить** (рис. 1.9).
5. Добавьте каталог `C:\php` в путь поиска программы (рис. 1.10).
6. Закройте последние три окна, последовательно нажимая кнопки **ОК**.
7. Закройте окно **Система**.

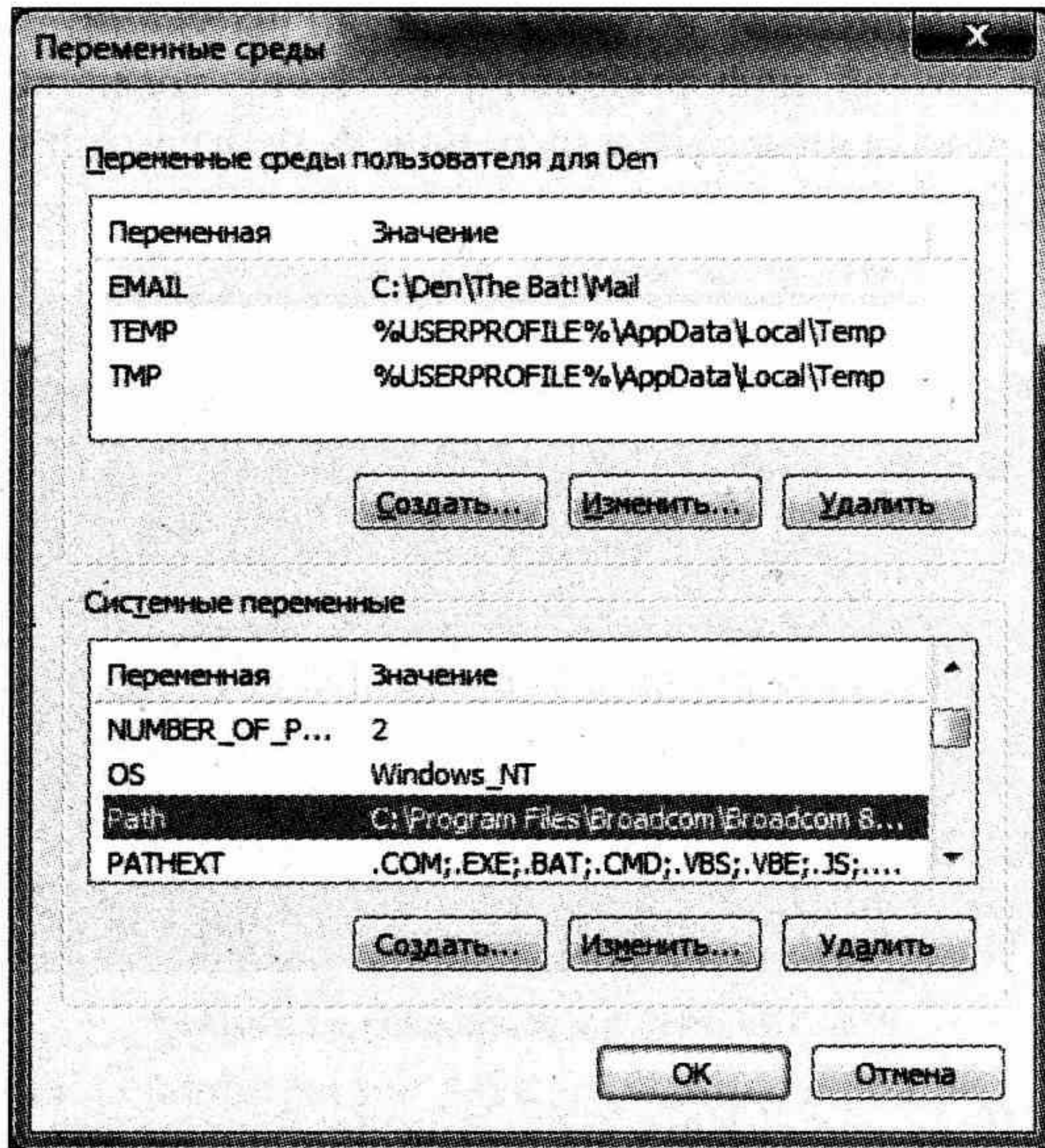


Рис. 1.9. Переменные среды

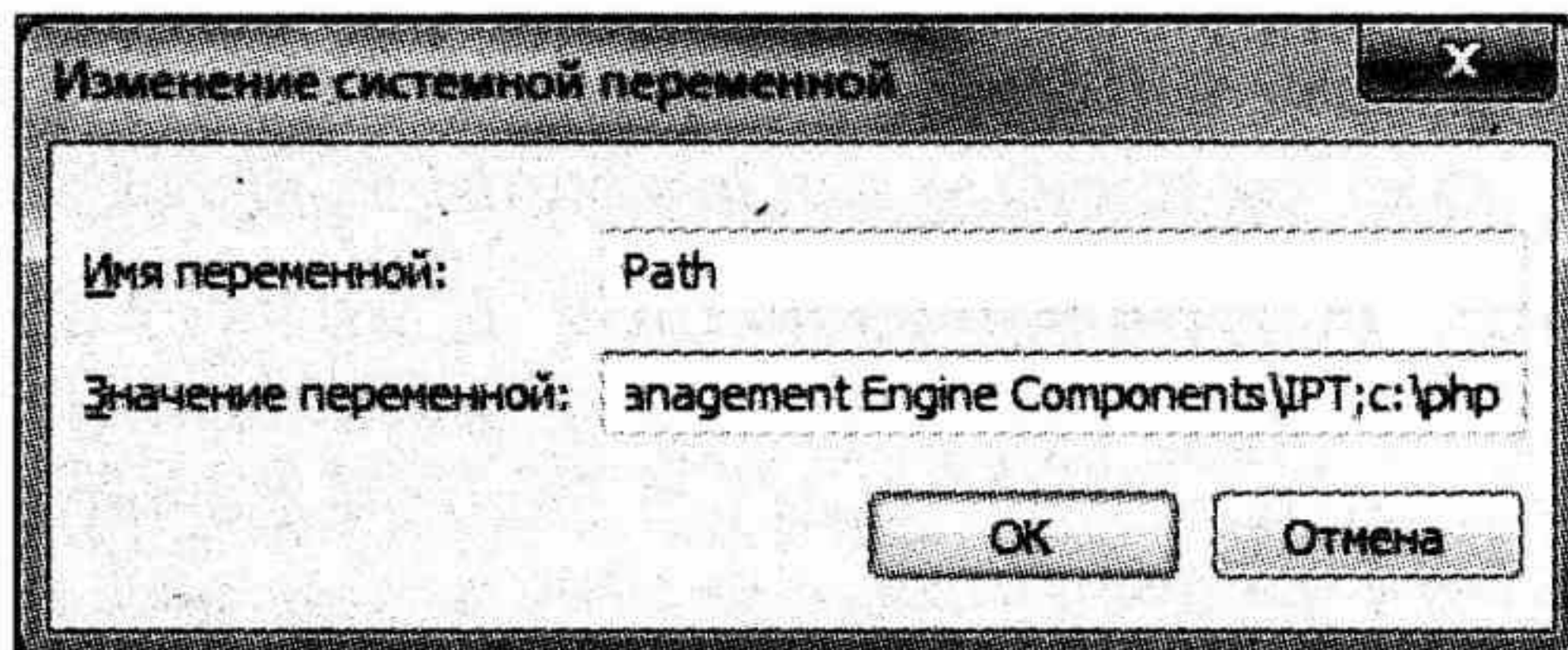


Рис. 1.10. Редактирование переменной Path

Откройте любой файловый менеджер (подойдет и Проводник, но удобнее использовать приложение вроде Total Commander). Перейдите в каталог `C:\php` и скопируйте файл `php.ini-development` в `php.ini`. Откройте файл `php.ini` и найдите в нем строку:

```
; extension_dir = "ext"
```


Эту строку нужно раскомментировать, чтобы получилось так:

```
extension_dir = "ext"
```

ПРИМЕЧАНИЕ

В дистрибутиве вы найдете два файла: *php.ini*: *php.ini-development* и *php.ini-production*. Для разработки и отладки кода лучше использовать настройки для разработчика, т. е. файл *php.ini-development*. Файл *php.ini-production* содержит так называемые *производственные параметры*. Разработчику от них мало пользы, т. к. сообщения об ошибках выводиться не будут.

Сохраните файл *php.ini*. Теперь проверим, все ли вы сделали правильно. Откройте командную строку и введите команду `php -v`. Если вы увидите сообщение с номером версии PHP и другой информацией (рис. 1.11), тогда вы все сделали как надо — т. е. добавили каталог с PHP в путь поиска программ и создали файл *php.ini* с основными настройками.

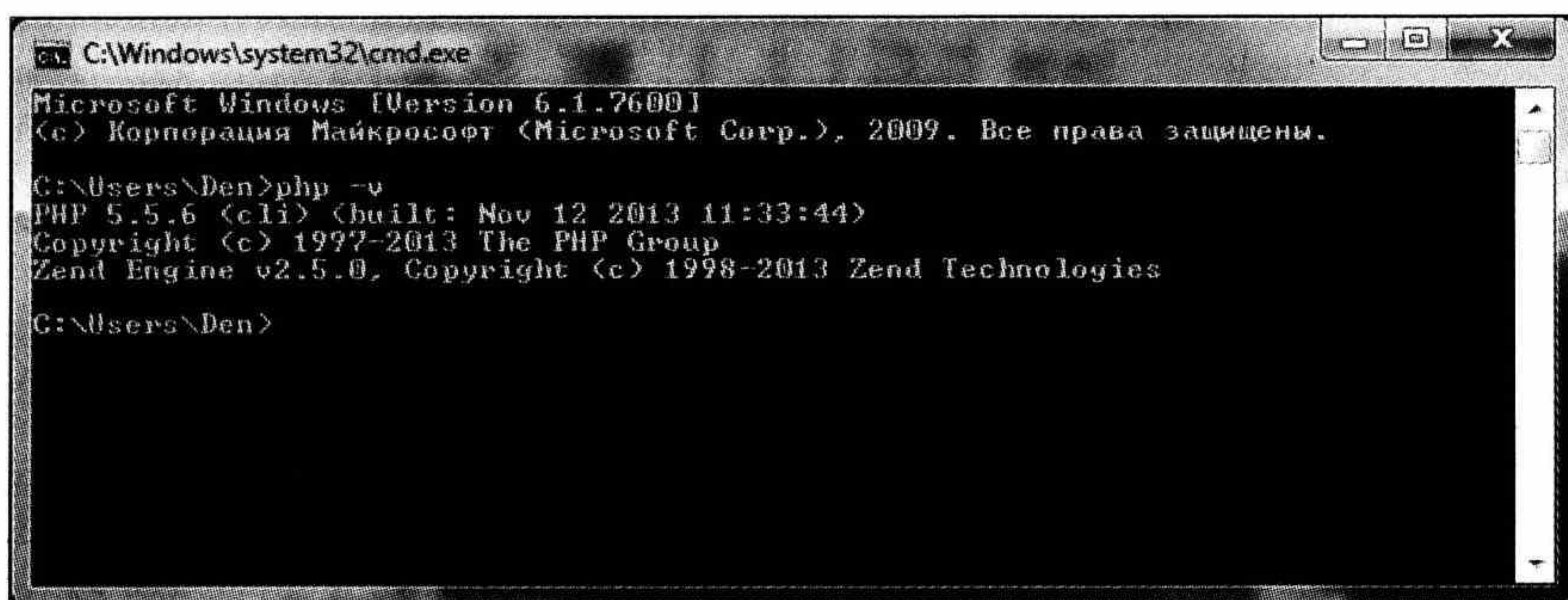


Рис. 1.11. PHP 5.5 установлен и работает

После установки PHP нужно связать его с Apache. Раньше все эти операции делал инсталлятор, сейчас, повторяю, приходится выполнять их вручную.

В каталоге `C:\php` имеется файл `install.txt`. Откройте его и найдите в нем следующие строки:

```
# For PHP 5 do something like this:
LoadModule php5_module "c:/php/php5apache2.dll"
AddType application/x-httpd-php .php

# configure the path to php.ini
PHPIniDir "C:/php"
```

Эти строки нужно выделить и скопировать в буфер обмена.

Перейдите в каталог, куда вы установили Apache (в нашем случае — это `C:\WWW`), затем — в подкаталог `conf`. Откройте файл конфигурации Apache — `httpd.conf`. Найдите в этом файле секцию `LoadModule`. Каждая строка `LoadModule` загружает какой-то модуль Apache. Пролистайте файл, пока не закончатся строки `LoadModule`. После

последней строки `LoadModule` добавьте недавно скопированные из файла `install.txt` в буфер строки. У вас должно получиться примерно так:

```
#LoadModule watchdog_module modules/mod_watchdog.so
#LoadModule xml2enc_module modules/mod_xml2enc.so
```

```
# For PHP 5 do something like this:
LoadModule php5_module "c:/php/php5apache2.dll"
AddType application/x-httpd-php .php
```

```
# configure the path to php.ini
PHPIniDir "C:/php"
```

```
<IfModule unixd_module>
```

Казалось бы, все идет нормально, но если вы заглянете в каталог `C:\php`, то не найдете в нем файла `php5apache2.dll`. На самом деле эта библиотека называется `php5apache2_4.dll`. Поэтому вернитесь к файлу `httpd.conf` и отредактируйте соответствующую строку:

```
LoadModule php5_module "c:/php/php5apache2_4.dll"
```

После этого найдите в файле `httpd.conf` следующие строки:

```
<IfModule dir_module>
DirectoryIndex index.html
</IfModule>
```

Добавьте сюда поддержку файла `index.php`:

```
<IfModule dir_module>
DirectoryIndex index.php index.html
</IfModule>
```

На этом все... Файл конфигурации Apache можно сохранить. Далее с помощью приложения `Apache Service Monitor`, которое устанавливается вместе с Apache, перезапустите Apache, нажав кнопку **Restart**. Также можно перезапустить Apache с помощью команды `services.msc`. Нажмите кнопку **Пуск** и введите команду `services.msc`. Отыщите в списке служб Apache 2.2 и нажмите кнопку **Перезапуск службы** (рис. 1.12). В случае возникновения ошибки перезагрузите компьютер.

Теперь вам осталось скачать и установить сервер баз данных MySQL (<http://mysql.com/>). Его установка проходит вообще без каких-либо нюансов, поэтому ее мы подробно рассматривать не будем.

ПРИМЕЧАНИЕ

В каталоге *software* сопровождающего книгу электронного архива (см. приложение 4) вы найдете MySQL версии 5.5 (файл *mysql-5.5.22-win32.msi*). Это не самая новая версия MySQL, но зато проверенная и стабильная, а самая новая (сейчас это 5.6 для Windows и 5.7 для UNIX) нам не нужна.

В главе 2 мы напишем тестирующий созданную вами конфигурацию сценарий `test.php`, который нужно будет поместить в подкаталог `C:\WWW\htdocs`. А в конце

данной главы будет написан более простой проверочный сценарий, и перед тем, как приступить ко *второй главе*, убедитесь, что этот простой сценарий у вас выполняется.

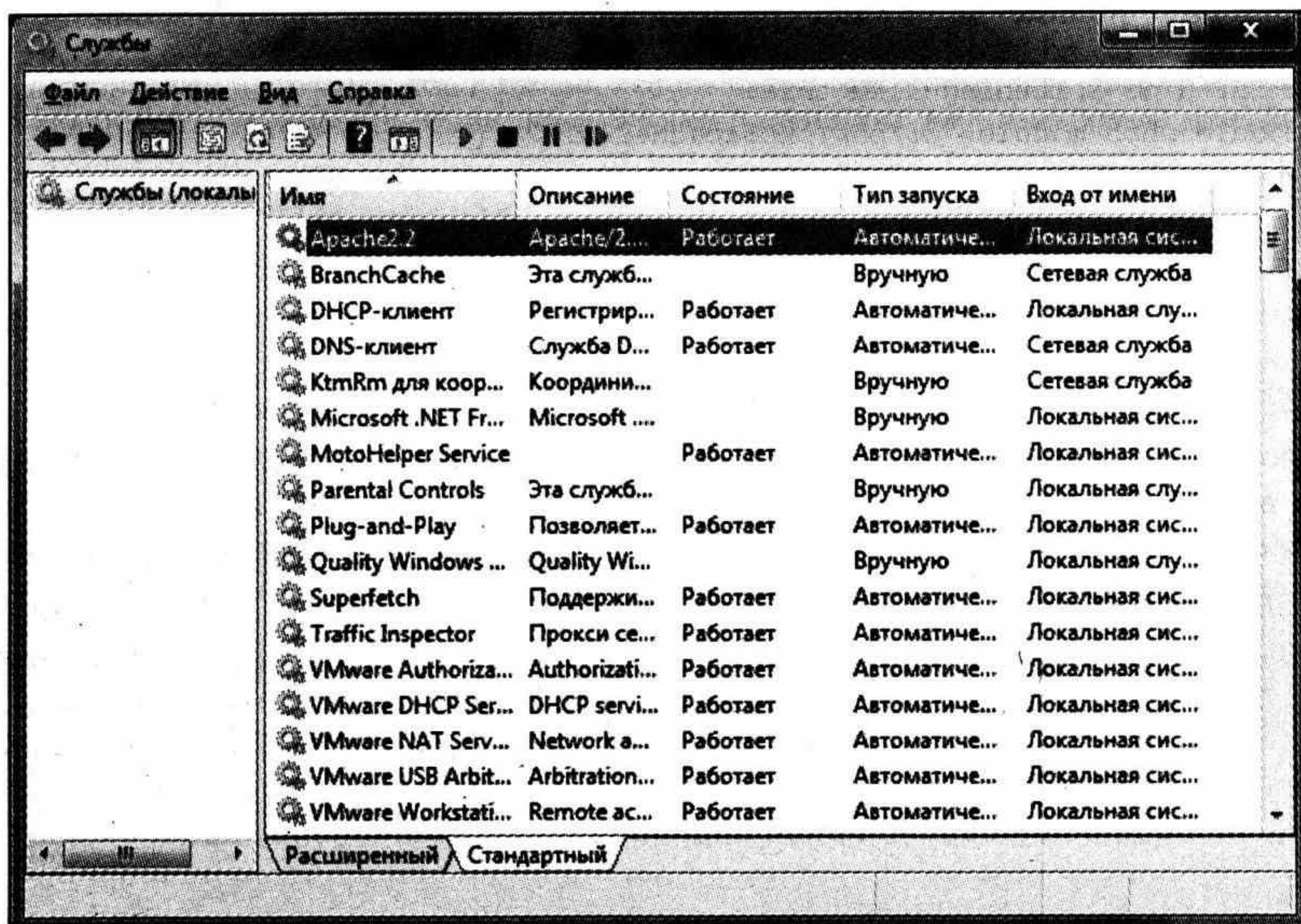


Рис. 1.12. Управление службами Windows

На мой взгляд установка в Windows всех компонентов, необходимых для организации связки Apache + PHP + MySQL, в настоящее время напоминает установку такой связки в Linux лет эдак 5 или 7 назад. В Linux же, наоборот, все стало гораздо проще — там достаточно только установить необходимые пакеты (apache, php, mysql), о чем далее.

1.3.2. Установка для новичков (рекомендуется)

Если вам не с руки устанавливать все компоненты связки отдельно, вы можете установить пакет XAMPP, включающий Apache, PHP и MySQL "в одном флаконе". Скачать последнюю версию XAMPP можно по адресу <http://www.apachefriends.org/en/xampp.html>.

ПРИМЕЧАНИЕ

В каталоге *software* сопровождающего книгу электронного архива (см. приложение 4) вы найдете последнюю (на момент написания этих строк) версию XAMPP — 1.8.3 (файл *xampp-win32-1.8.3-2-VC11-installer.exe*).

Основное преимущество XAMPP — простота установки и управления всеми сервисами, входящими в его состав (веб-сервер, FTP-сервер и MySQL). Начиная с вер-

сии 1.8.3, в комплект поставки входит PHP версии 5.5 (в предыдущих сборках XAMPP присутствовали версии PHP 5.4 и более ранние).

Установка XAMPP проблем не вызывает. Программа нормально работает в актуальных версиях Windows: Vista и Windows 7/8. В Windows Vista и Windows 7/8 программу установки XAMPP нужно запускать от имени администратора.

ВНИМАНИЕ!

PHP 5.5 больше не поддерживает Windows XP, поэтому вы не сможете установить XAMPP в этой операционной системе.

Желательно устанавливать XAMPP в корневой каталог диска C:\, а не в каталог C:\Program Files, чтобы не возникло каких-либо проблем с правами доступа — так рекомендуют разработчики XAMPP. Да и вам добираться до своих файлов будет проще.

При установке Apache и MySQL вас спросят, как вы хотите запускать эти программы: в качестве сервисов (служб) Windows или вручную. Если вы собираетесь программировать на PHP каждый день, тогда удобнее запускать Apache и MySQL как сервисы. Но помните, что даже когда они вам не нужны, они все равно занимают память и потребляют ресурсы процессора. Поэтому, если вы собираетесь программировать на PHP время от времени, лучше запускать эти программы вручную. Для этого выполните команду меню **Пуск | Все программы | XAMPP for Windows | XAMPP Control Panel** — откроется панель управления XAMPP. Она позволяет запустить, остановить и настроить подконтрольные сервисы (рис. 1.13):

- ☐ **Apache** — это и есть веб-сервер;
- ☐ **MySQL** — сервер баз данных;
- ☐ **FileZilla** — простейший FTP-сервер для Windows;
- ☐ **Mercury** — простейший почтовый сервер (точнее, агент доставки почты — MTA) для Windows;
- ☐ **Tomcat** — программа-контейнер сервлетов, написана на языке Java и реализует спецификацию сервлетов и спецификацию JavaServer Pages (JSP).

Все сервисы запускать не надо — вам хватит первых двух. Третий сервис нужно запустить, если вы хотите написать PHP-сценарий, взаимодействующий с FTP-сервером — вам не придется искать какой-то FTP-сервер, все будет у вас под рукой. Система **Mercury** подойдет для проектов, подразумевающих отправку почты, — без запуска Mercury ваши проекты не смогут отправлять почту стандартной функцией `mail()`, или же вам придется использовать дополнительные классы вроде **PHPMailer** для отправки почты через сторонние SMTP-серверы.

Для запуска сервиса нажмите кнопку **Start** напротив его имени. В случае успешного запуска кнопка **Start** превратится в кнопку **Stop** (служит для останова сервиса), а между названием сервиса и кнопкой **Stop** появится надпись **Running**.

При первом запуске каждой из служб вы увидите окно брандмауэра, предлагающее разрешить или запретить приложению доступ в сеть (рис. 1.14). Установите флажок **Общественные сети** и нажмите кнопку **Разрешить доступ**.

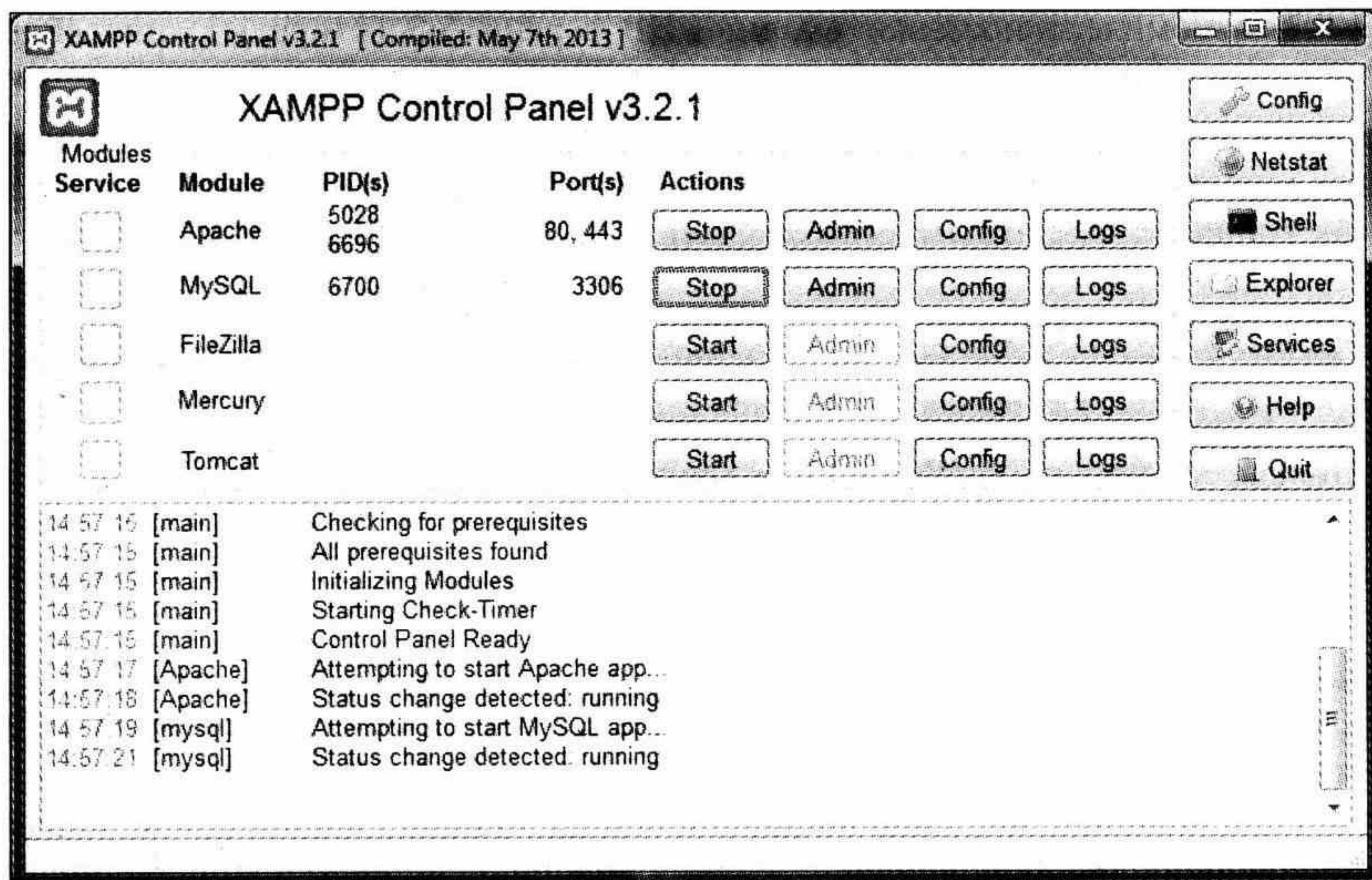


Рис. 1.13. Панель управления XAMPP

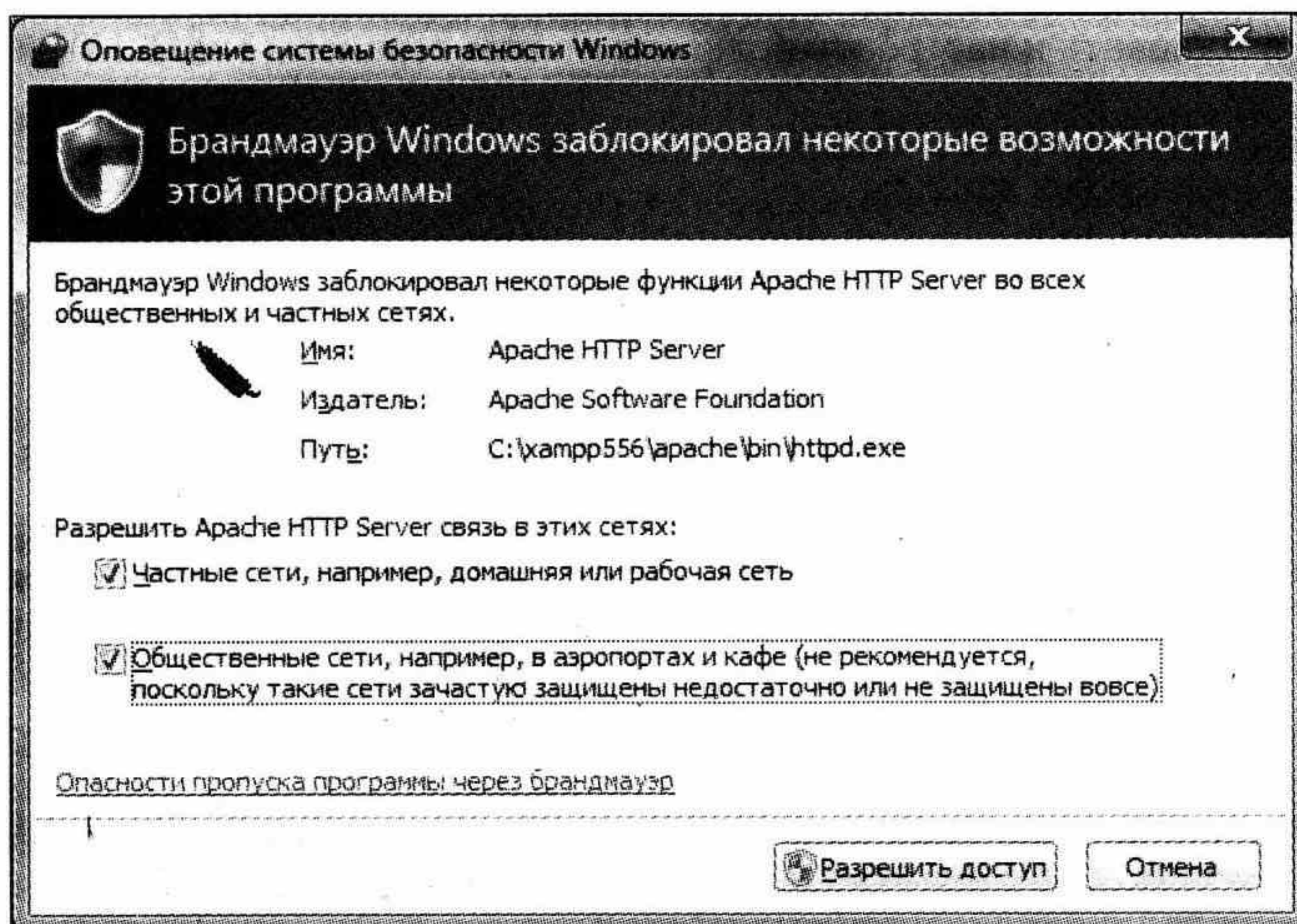


Рис. 1.14. Оповещение системы безопасности Windows

Кнопка **Admin** (см. рис. 1.13), которая становится активной после запуска сервиса, позволяет настраивать выбранный сервис. На рис. 1.15 изображена страница настройки Apache, а нажатие кнопки **Admin** для сервиса **MySQL** приводит к вызову программы phpMyAdmin (рис. 1.16), которая предоставляет удобный веб-интер-

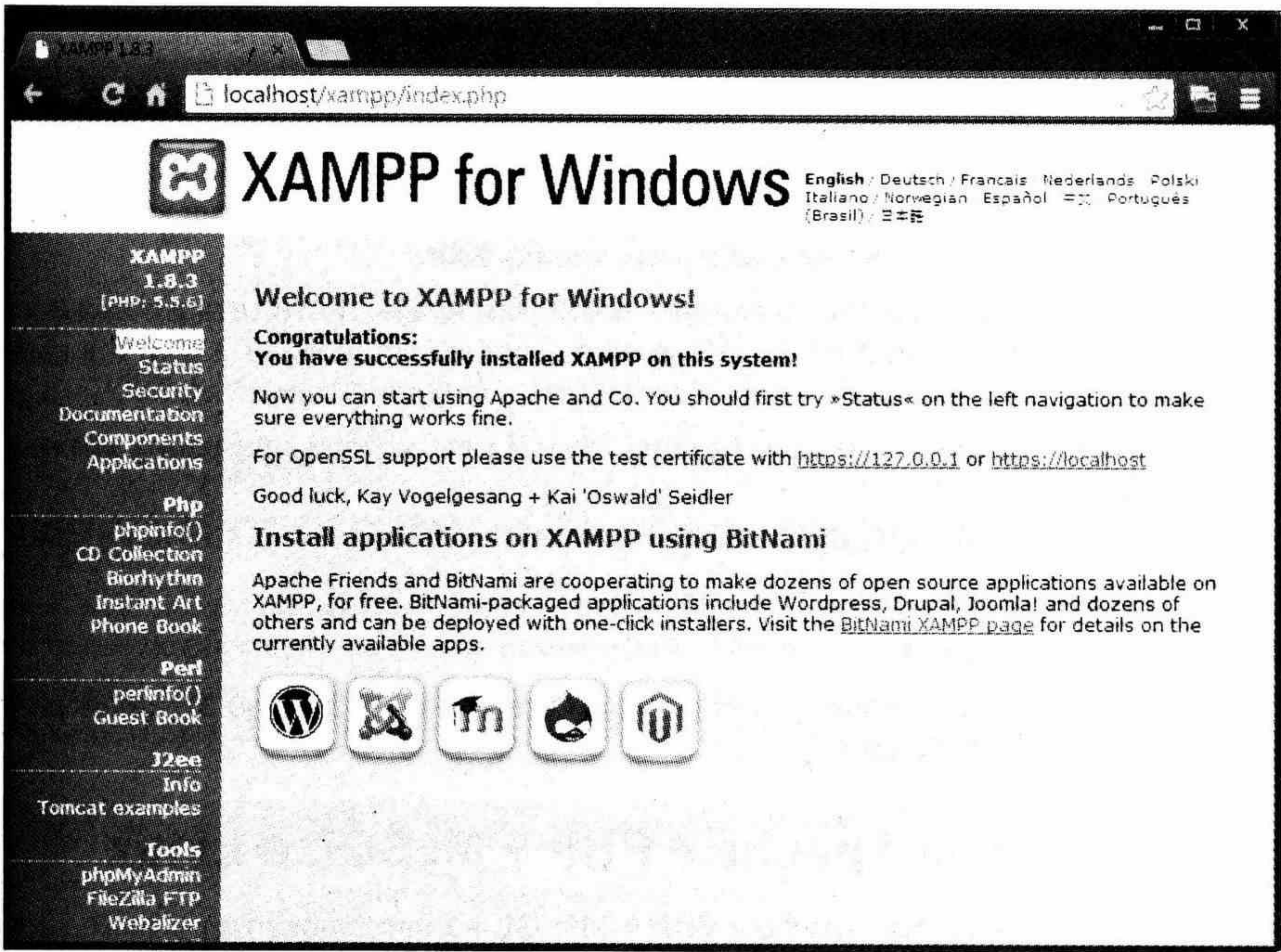


Рис. 1.15. Страница настройки Apache

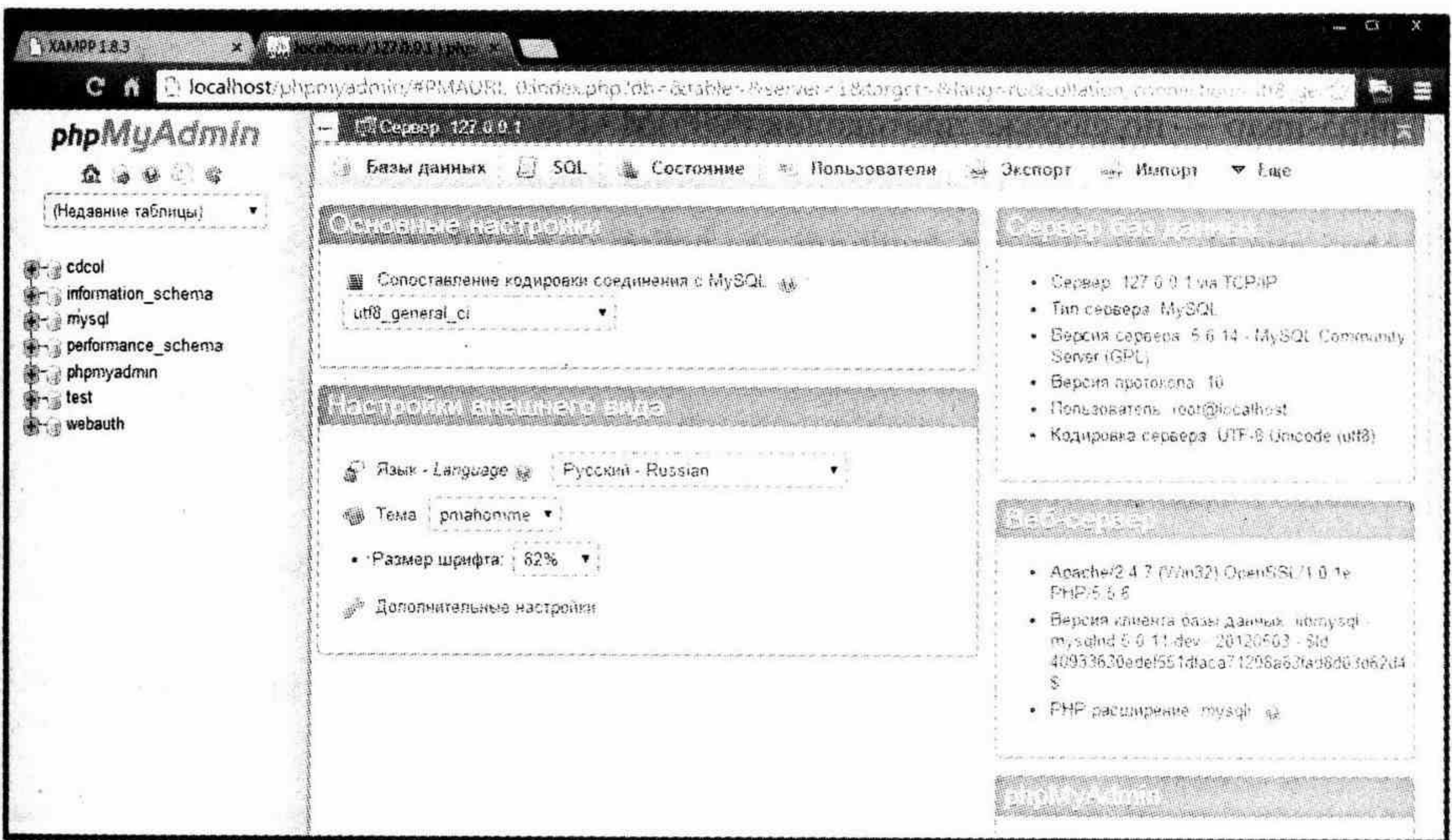


Рис. 1.16. Программа phpMyAdmin

фейс для управления вашими базами данных. Подробно программа описана в *приложении 1*.

Для выхода из панели управления XAMPP служит кнопка **Exit** (см. рис. 1.13). Помните, что при выходе из панели запущенные сервисы не будут остановлены. Если вы хотите завершить работу сервисов, нажмите кнопку **Stop** напротив каждого запущенного сервиса, а затем уже нажимайте кнопку **Exit**.

Свои PHP-сценарии при использовании XAMPP вам нужно поместить в подкаталог `htdocs` каталога установки XAMPP. Например, если вы установили XAMPP в каталог `C:\xampp`, то PHP-сценарии следует поместить в каталог `C:\xampp\htdocs`.

При написании сценариев, использующих MySQL, вы можете применить следующие параметры доступа:

- ☐ сервер MySQL — `localhost`;
- ☐ имя пользователя — `root`;
- ☐ пароль — отсутствует;
- ☐ база данных по умолчанию — `test` (но вы можете создать другую БД с помощью программы `phpMyAdmin`).

1.4. Установка Apache + PHP + MySQL в Linux

Процесс установки связки Apache + PHP + MySQL в Linux зависит от вашего дистрибутива. В общем случае вам нужно установить следующие пакеты:

- ☐ `apache2` — веб-сервер Apache;
- ☐ `php5-cli` — интерпретатор PHP, работающий в режиме командной строки (`command-line interpreter`);
- ☐ `php5-imap` — поддержка протоколов POP/IMAP для PHP;
- ☐ `php5-gd` — поддержка графических функций PHP;
- ☐ `php5-mysql` — поддержка функций для работы с базой данных MySQL;
- ☐ `mysql-server-5` — MySQL-сервер;
- ☐ `mysql-client-5` — MySQL-клиент.

Например, в Fedora для установки пакета служит команда `yum install`:

```
yum install apache2
```

Сразу после установки MySQL нужно задать пароль администратора:

```
# mysqladmin -u root password ваш_пароль
```

Для управления веб-сервером Apache можно применять следующие команды:

- ☐ `# service httpd start` — запуск сервера;
- ☐ `# service httpd stop` — останов сервера;
- ☐ `# service httpd restart` — перезапуск сервера.

Если в вашем дистрибутиве нет команды `service`, используйте следующие команды:

```
sudo /etc/init.d/apache2 start
sudo /etc/init.d/apache2 stop
sudo /etc/init.d/apache2 restart
```

Аналогично для управления MySQL служат следующие команды:

```
# service mysql start
# service mysql stop
# service mysql restart
```

1.5. Конфликты с другим программным обеспечением

Дело в том, что стандартные порты Apache (80 и 443) могут использоваться другими приложениями, в частности популярной программой Skype. Чтобы на вашем компьютере работал и Skype, и веб-сервер Apache, откройте Skype, выполните команду меню **Инструменты | Настройки**. Затем перейдите в раздел **Дополни-**

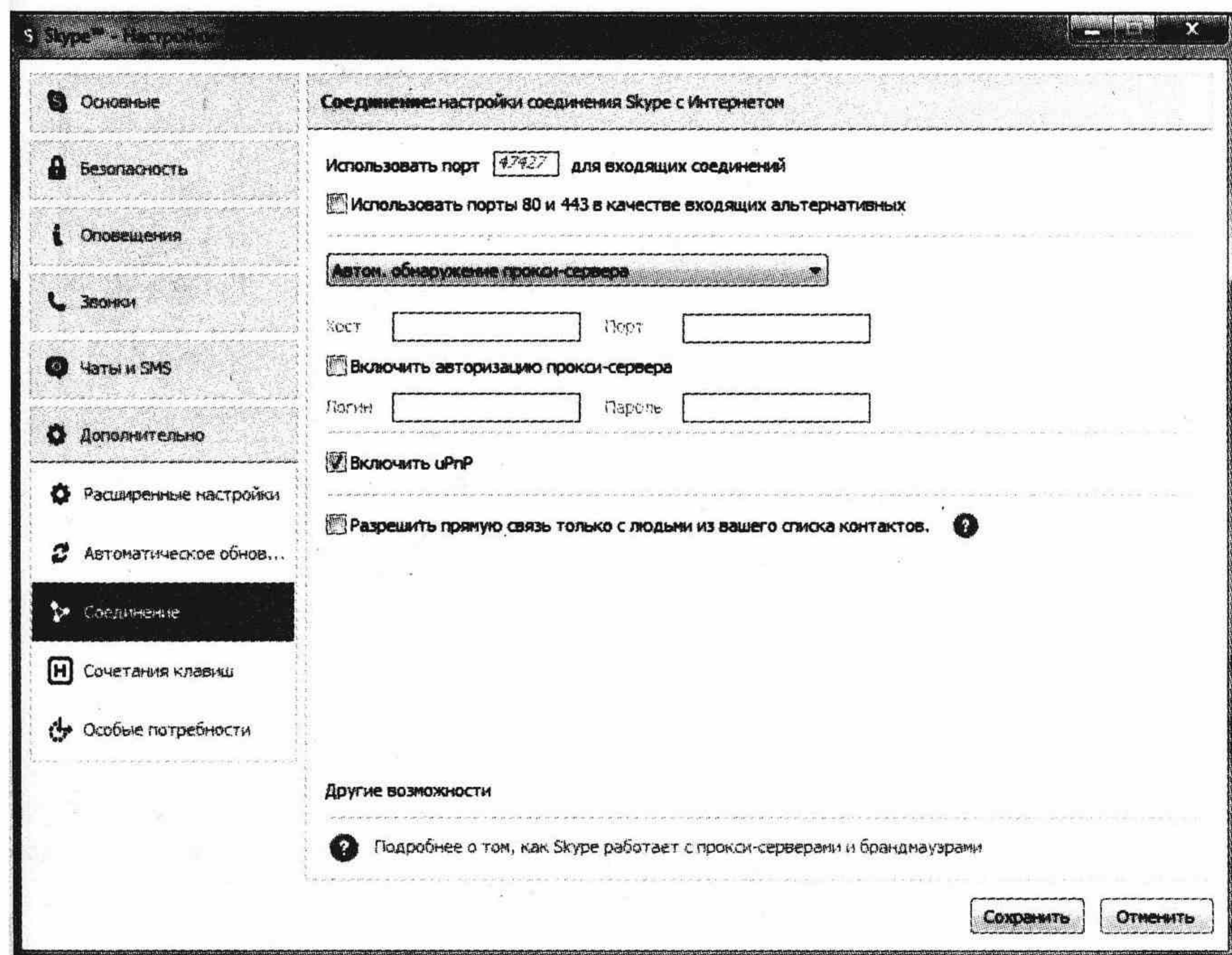


Рис. 1.17. Настройка Skype

тельно | Соединение и снимите флажок **Использовать порты 80 и 443 в качестве входящих альтернативных**. Нажмите кнопку **Сохранить**.

Аналогично, программа uTorrent использует порт 8080 для своего веб-интерфейса — и к этому же порту обращается программа Tomcat, входящая в состав XAMPP. Чтобы можно было использовать обе эти программы одновременно, я рекомендую перенести веб-интерфейс uTorrent на другой порт — например, на 8088 (рис. 1.18).

Интересно, что даже при выключенном веб-интерфейсе uTorrent все равно обращается к порту 8080, поэтому данную настройку придется выполнить, даже если вы вообще не используете веб-интерфейс.

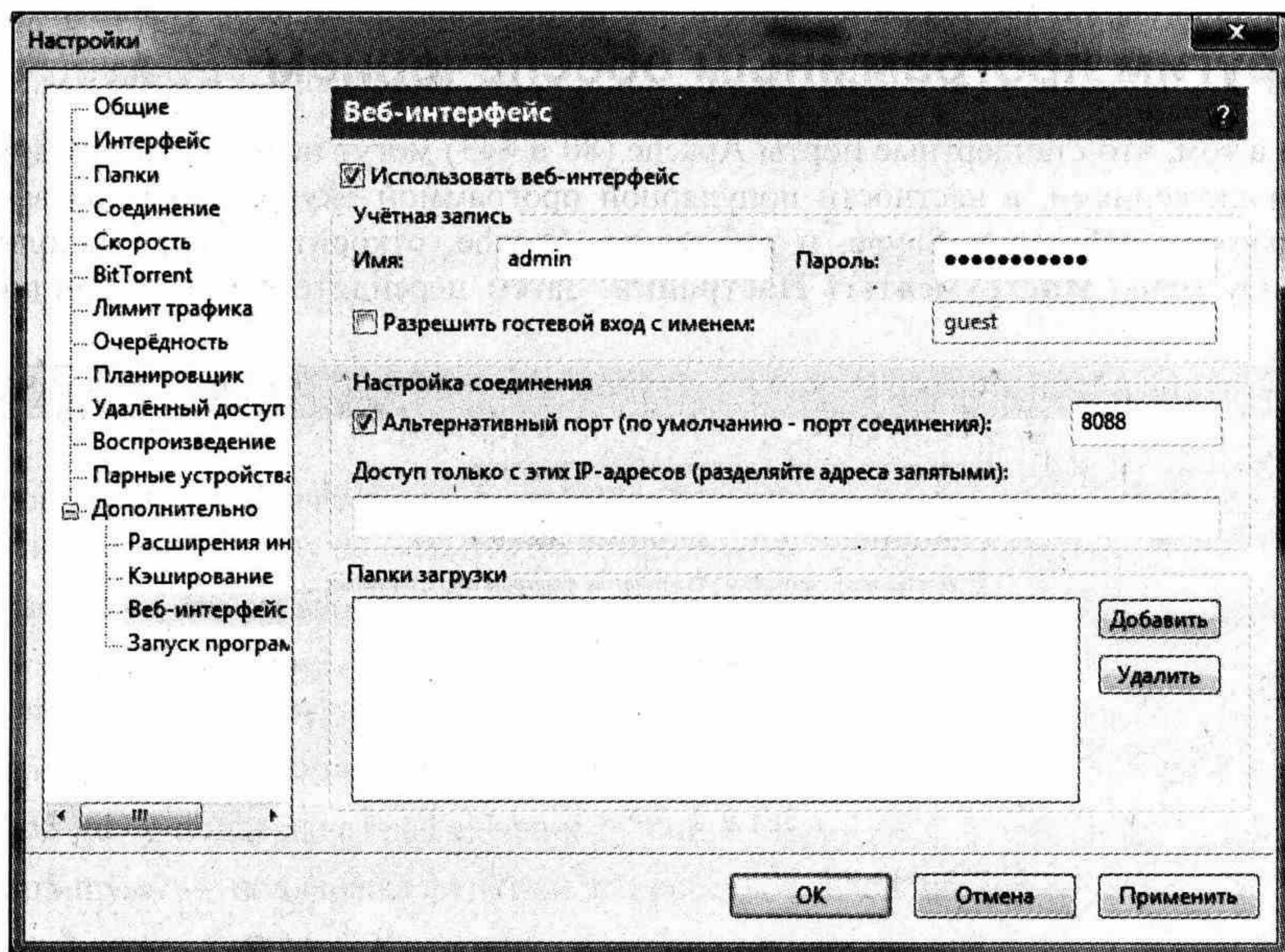


Рис. 1.18. Настройка uTorrent

1.6. Функция *phpinfo()*: получение информации о PHP

В главе 2 мы напишем более сложный сценарий, позволяющий выполнить проверку как работоспособности PHP и модуля Apache, так и конфигурацию MySQL. А сейчас я предлагаю создать файл `info.php` со следующим содержимым:

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


Этот файл — простейший PHP-сценарий, вызывающий функцию `phpinfo()`, которая сообщает множество полезной информации о вашем веб-сервере и о самом PHP. Поместите этот файл в каталог документов (`C:\WWW\htdocs` или `C:\xampp\htdocs`), откройте браузер и введите адрес: `http://localhost/info.php`. Если вы все сделали правильно, то должны увидеть страницу, подобную изображенной на рис. 1.19. У вас может отличаться только версия PHP.

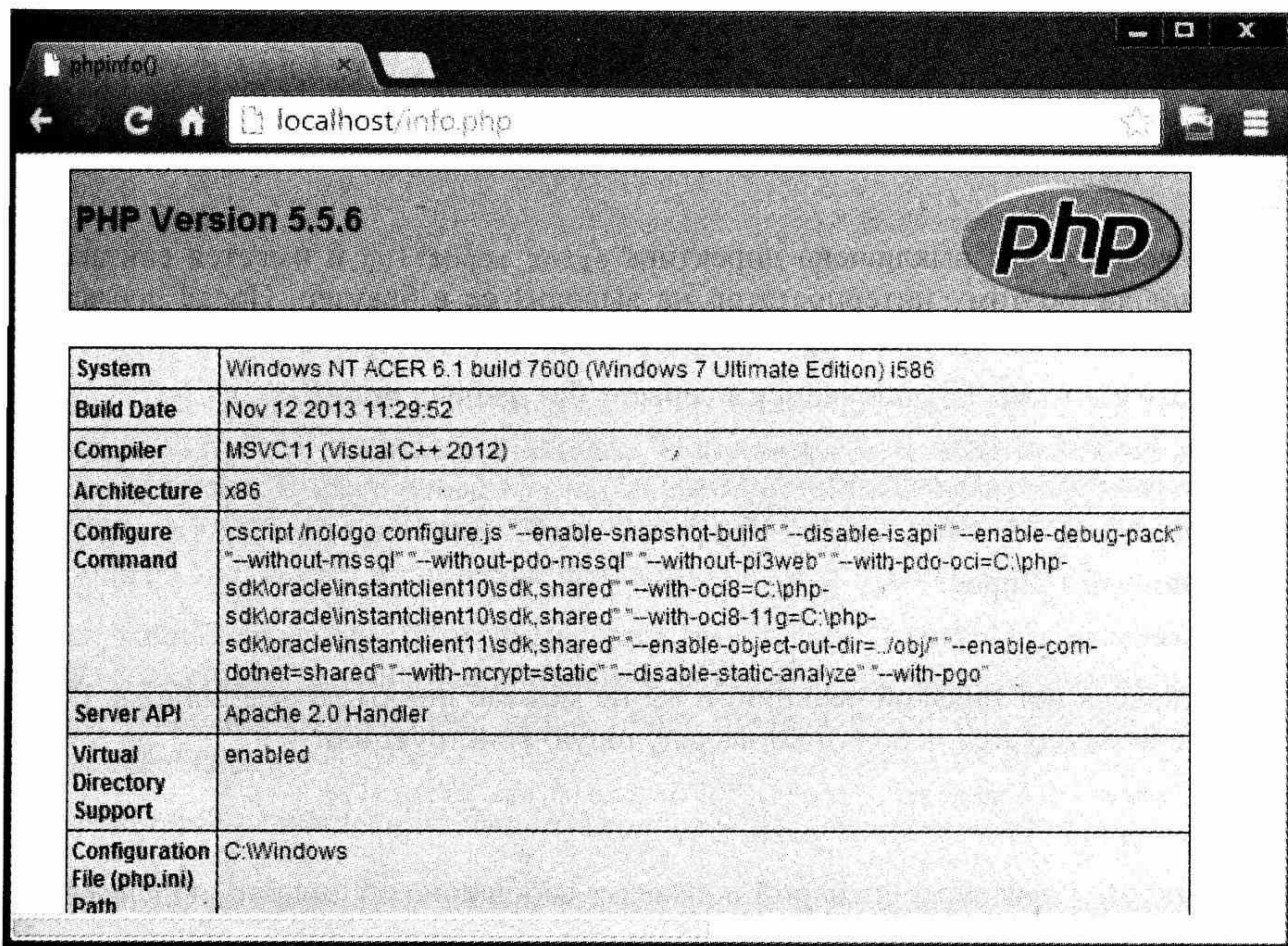


Рис. 1.19. Информация о веб-сервере и PHP

Если же вы вместо информационной таблицы увидели содержимое файла `info.php`, то PHP не работает вместе с Apache. Чтобы исправить это, выполните следующие действия:

1. Перезапустите Apache (ранее было показано, как это сделать) и повторите попытку. Возможно, вы просто забыли перезапустить веб-сервер. Если же перезапуск сервера не дал результатов, перейдите к следующему шагу.
2. Ранее было показано, какие строки нужно было скопировать из файла `install.txt` и вставить в файл `httpd.conf`. Убедитесь, что вы все сделали без ошибок, и снова перезапустите веб-сервер. Если и сейчас у вас ничего не получилось, скачайте и установите XAMPP заново — в 99.99 % у вас не будет никаких проблем, и вы сразу получите все необходимое программное обеспечение, готовое к использованию.

1.7. Несколько советов

Перед тем как приступить к написанию сценариев, позволю себе дать несколько советов. Итак, если вы при запуске сценария ничего не видите, хотя сценарий должен осуществлять вывод, — нет даже сообщений об ошибках, то добавьте в начало сценария следующую строку:

```
error_reporting(E_ALL);
```

Например:

```
<?php  
error_reporting(E_ALL);  
...
```

Скорее всего, у вас выключена директива `error_reporting` и имеется синтаксическая ошибка, поэтому интерпретатор не выводит ее в браузер. После добавления этой директивы вы научите PHP "говорить".

Представим, что мы подключаемся к серверу баз данных `mysql`:

```
// имя узла — localhost, пользователь — root, без пароля  
mysql_connect("localhost", "root", "");  
mysql_select_db("test");
```

или выполняем запрос:

```
mysql_query($q);
```

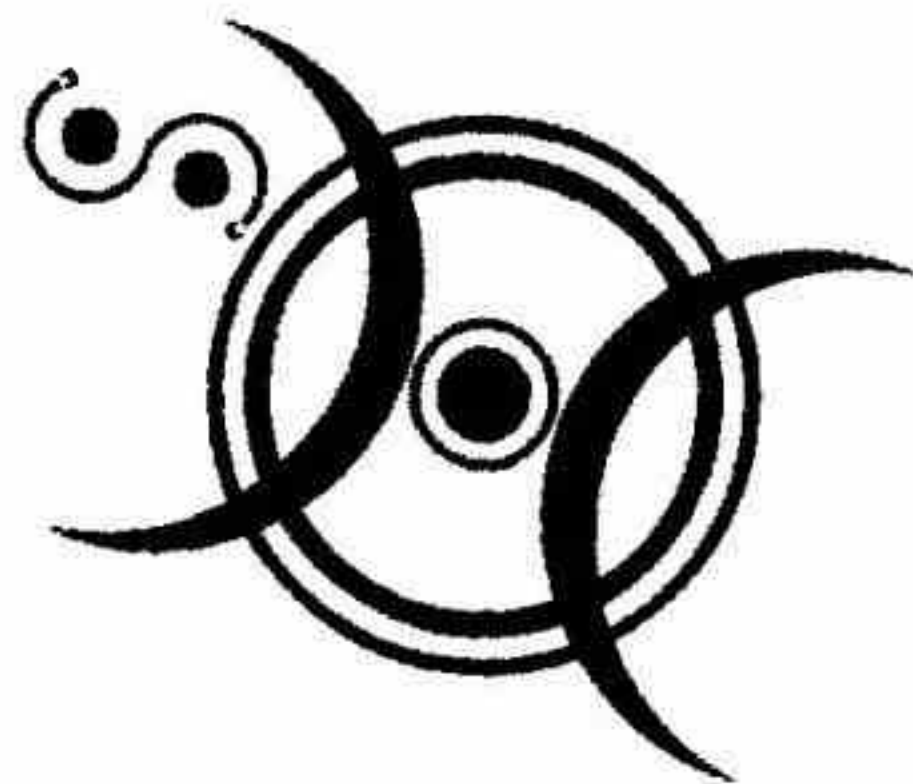
Но от сервера нет никакой реакции, и вы не можете понять, в чем ошибка. В этом случае рекомендуется использовать следующую конструкцию:

```
mysql_connect("localhost", "root", "") or die(mysql_error());  
mysql_query($q) or die(mysql_error());
```

Функция `die()` завершит сценарий и выведет сообщение об ошибке, полученное от функции `mysql_error()`.

Вот теперь можно перейти ко *второй главе*, в которой мы напишем вашу первую программу.

ГЛАВА 2



Программа на PHP

2.1. Ваша первая программа

Традиционно многие книги по программированию начинаются с создания простейшей программы, выводящей строку "Hello world!". В этой книге такой программы не будет. Вместо этого мы напишем простейшую программу, тестирующую наше MySQL-соединение. Этим мы сразу "убьем двух зайцев": напишем первую PHP-программу и проверим созданную в *главе 1* конфигурацию.

Логика нашей будущей программы проста — если удалось подключиться к MySQL-серверу, то все настроено правильно, иначе будет выведено сообщение об ошибке. В последующих главах книги будут приведены более сложные примеры работы с MySQL.

Итак, приступим к написанию нашего первого PHP-сценария (листинг 2.1).

Листинг 2.1. Первая PHP-программа

```
<?php

// параметры доступа к MySQL, возможно, придется изменить
if (!mysql_connect('localhost','root','')) die('error');

/* это пример многострочного
   комментария */

# пример комментария в стиле UNIX bash
/* если программа не остановлена die(), тогда выводим
   сообщение об успешном подключении */

echo "Соединение установлено успешно";

?>
```


ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *2-1.php*), представленный в каталоге *Glava_2* сопровождающего книгу электронного архива (см. приложение 4).

Разберемся, что мы узнали, написав эту программу:

- ❑ PHP-код заключается в теги `<?php` и `?>`. При желании теги можно сократить до вида `<?` и `?>`;
- ❑ инструкция `die()` выводит текст **error** и останавливает выполнение сценария, когда нам это нужно (в нашем случае, если не получилось подключиться к MySQL-серверу);
- ❑ комментарии в PHP-программах могут быть трех типов. Лично мне удобнее использовать однострочные комментарии в стиле C: `// Комментарий`;
- ❑ для вывода текста служит инструкция `echo`;
- ❑ после каждого оператора нужно указывать точку с запятой.

О КОРОТКИХ PHP-ТЕГАХ

Теги вида `<? ?>` называются короткими, или *short tags* (полная версия тегов выглядит так: `<?php ?>`). Чтобы можно было использовать сокращенную версию тегов, в файле *php.ini* нужно включить директиву `short_open_tag` (присвоить ей значение `On`). По умолчанию директива `short_open_tag` выключена (значение — `Off`). Некоторые хостеры включают эту директиву, некоторые нет. В XAMPP данная директива выключена по умолчанию. Старайтесь привыкать к полной версии PHP-тегов — так ваши сценарии станут более универсальными. В этой книге мы иногда будем использовать полную версию, иногда — сокращенную. Но не удивляйтесь, если вдруг сценарии, использующие сокращенную версию тегов, не станут работать — нужно или включить упомянутую директиву, или использовать полную версию тегов.

2.2. Запуск PHP-программы

Запустить программу очень просто. Сохраните ее как PHP-файл (например, *test.php*) и загрузите на ваш веб-сервер (предположим, в корневой каталог веб-сервера). После чего откройте браузер и введите URL:

`http://сервер/test.php`

ПРИМЕЧАНИЕ

Файл *test.php* нужно загрузить именно в корневой каталог веб-сервера, а не в корневой каталог вашей учетной записи на FTP-сервере хостинг-провайдера. Обычно это каталог *public_html* или *www*. За более точной информацией обратитесь к вашему хостинг-провайдеру.

Если у вас локальный веб-сервер, то запустить программу можно так:

`http://localhost/test.php`

Если вы при запуске сценария из листинга 2.1 увидели сообщение **error**, то проверьте параметры доступа к MySQL: имя сервера, имя пользователя и пароль. В большинстве случаев проблема именно в них. Если вы сами устанавливали сервер MySQL, то нужно использовать следующие параметры доступа:

- ❑ имя сервера — localhost;
- ❑ имя пользователя — root;
- ❑ пароль — пустой (отсутствует).

Если же вы запускаете сценарий на сервере хостинг-провайдера, то параметры доступа будут другими.

2.3. Вывод текста без *echo*

Создайте на основании листинга 2.2 файл `html.php`.

Листинг 2.2. Сценарий `html.php` (вариант 1)

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

Как видите, данный сценарий ничем не отличается от PHP-кода. Попробуйте запустить этот сценарий — PHP не найдет PHP-кода в файле (поскольку в нем нет тегов `<?php` и `?>`) и выведет HTML-код "как есть". Но самое интересное далее. PHP-сценарии могут содержать как HTML-код, так и PHP-код. Модифицируем наш файл `html.php` (листинг 2.3).

Листинг 2.3. Сценарий `html.php` (вариант 2)

```
<html>
<head><title>Заголовок</title></head>
<body>
<?php
echo "<h1>Выводим текст</h1>";
?>
</body>
</html>
```

В результате выполнения этого сценария будет сформирована следующая HTML-страница (рис. 2.1).



Рис. 2.1. Результат выполнения листинга 2.3

Пойдем дальше. Вы можете включать в HTML-код неограниченное количество PHP-кода (листинг 2.4).

Листинг 2.4. Сценарий html.php (вариант 3)

```
<html>
<head><title>Заголовок</title></head>
<body>
<?php
echo "<h1>Выводим текст</h1>";
?>
</body>
<?php
echo "</html>";
?>
```

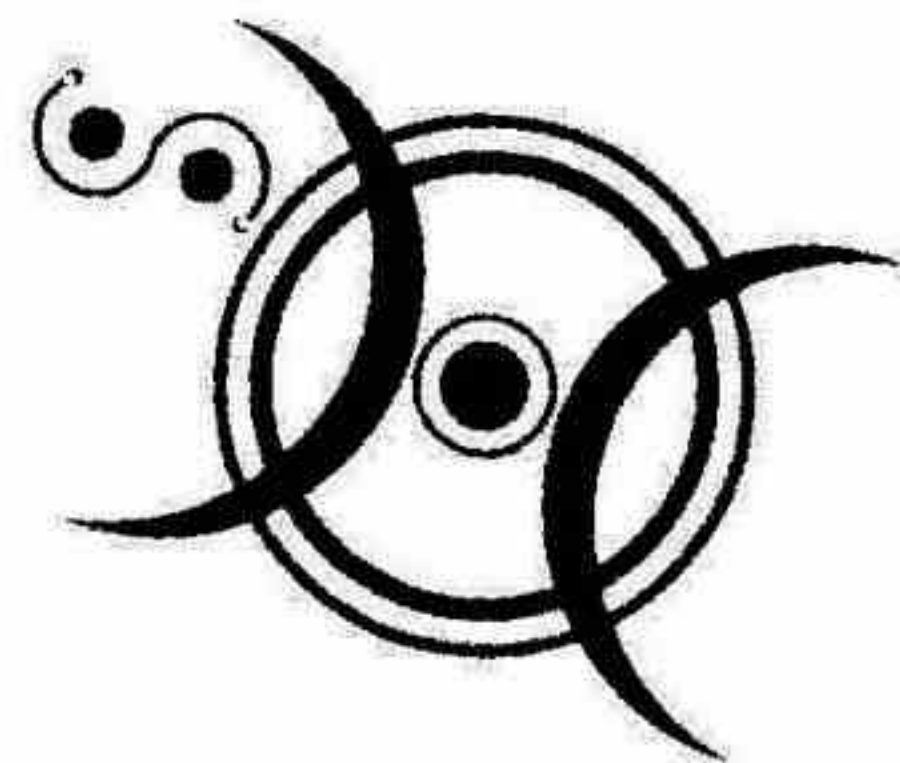
ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *2-4.php*), представленный в каталоге *Glava_2* сопровождающего книгу электронного архива (см. приложение 4).

Сценарий *html.php* демонстрирует, как можно включать HTML-код в состав PHP-сценариев. Но я вам советую избегать подобной практики. HTML-код и PHP-код должны храниться отдельно. Ведь когда созданный сайт будет передан заказчику, его станет поддерживать веб-дизайнер, который может не иметь представления о PHP и легко допустить ошибку, из-за которой сценарий вообще не запустится. Поэтому лучше всего хранить HTML-код в TPL-шаблонах, заполняемых нашим PHP-сценарием. Работа с шаблонами будет рассмотрена в главах 22 и 23.

Забегая вперед, скажу, что HTML-код за пределами PHP-тегов таит в себе угрозу. Если до PHP-тегов будет обнаружен любой текст (даже пробелы или пробельные символы), то он будет выведен до выполнения самого кода в браузер, а потом PHP-сценарий не сможет установить файлы Cookies, поскольку они должны быть установлены до первого вывода в браузер.

ГЛАВА 3



ОСНОВЫ синтаксиса RHP

3.1. Переменные

3.1.1. Правила объявления переменных. Имена переменных

Если вы работали до этого с другими языками программирования, то наверняка знакомы с понятием *переменной*, которое присутствует и в RHP. Говоря формально, переменная — это поименованная область памяти. Области памяти присваивается выбранное программистом имя, по которому потом программа обращается к данным. Все данные, с которыми вы станете работать, будут храниться в переменных.

В других языках программирования переменные нужно объявлять до их первого использования — например, в языке Pascal так:

```
var  
    I : integer;
```

RHP не требует явного объявления переменных. Но если переменные не объявлены, как тогда RHP знает, что есть переменная, а что — обычный символ? Для выделения переменных из кода программы RHP требует, чтобы перед именем переменной указывался символ \$. Например: \$i, \$string.

То, что RHP не требует объявления переменной, не освобождает вас от ее инициализации, т. е. от задания переменной начального значения. Инициализация не обязательна, но весьма желательна. Предположим, вы используете переменную \$i как счетчик с шагом 2, т. е. после определенного события — например, после каждой итерации цикла, выполняется следующий оператор:

```
$i = $i + 2;
```

Если вы забыли присвоить переменной \$i начальное значение до выполнения этого оператора, результат будет непредсказуемым. Поэтому возьмите за правило инициализировать каждую переменную, которую вы собираетесь использовать, например:

```
$i = 0;
```


Вернемся к именам переменных. Как уже было сказано, перед именем переменной нужно обязательно указывать символ доллара. Например:

```
i = $i + 2;           // этот код неправильный!  
i = i + 2;            // этот код неправильный!  
$i = $i + 2;          // этот код ПРАВИЛЬНЫЙ
```

Имя переменной может состоять из латинских символов, знака подчеркивания и цифр, причем не может начинаться с цифры.

Имена переменных чувствительны к регистру символов, т. е. `$a` и `$A` — две разные переменные:

```
$a = 1;  
$A = 2;  
  
echo $a;  
echo $A;
```

Наш код выведет, как и следовало ожидать, 12. Если бы переменные не были чувствительны к регистру символов, код вывел бы 11.

3.1.2. Типы данных переменных

Обратите внимание, что при объявлении переменной на языке Pascal были указаны имя переменной и ее тип. РНР не требует явного объявления типа переменной, но это не означает, что в РНР нет типов переменных. Просто РНР по-особому работает с переменными.

Предположим, вы объявили две переменные. Одна переменная (`$a`) содержит число, а вторая переменная (`$b`) — массив. Присвоим второй переменной значение первой переменной:

```
$b = $a;
```

Вы только задумайтесь, что происходит. Вы присваиваете число массиву! В любом другом языке программирования вы бы получили ошибку несовместимости типов переменных, но не в РНР. РНР возьмет да и скопирует в переменную `$b` не только значение переменной `$a`, но и тип (структуру) переменной. Да, в результате массив будет потерян, а `$b` станет обычной целой переменной, содержащей значение переменной `$a`.

Такая "молчаливость" РНР довольно удобна, но, с другой стороны, если вы невнимательны, она может стать причиной трудно отслеживаемых ошибок.

Вот основные типы переменных в РНР:

- ❑ `int` — целой переменной вы можете присвоить значения от `-2 147 483 648` до `2 147 483 647`;
- ❑ `double` — переменная, содержащая вещественное число большой точности;
- ❑ `string` — наверное, самый важный тип данных в РНР, поскольку очень много операций связано именно с обработкой строк;

- `array` — массивам и спискам далее посвящен целый раздел книги;
- `object` — PHP является объектно-ориентированным языком программирования, что, впрочем, требуется далеко не всегда — вы можете написать превосходный сценарий, работающий и без объектов;
- `bool` — логическая переменная (может принимать значения `true` или `false`);
- `link` — ссылка, особый тип, о котором мы поговорим отдельно.

Для проверки типа переменной служат функции, приведенные в табл. 3.1.

Таблица 3.1. Функции проверки типа переменной

Функция	Описание
<code>is_int(\$x)</code>	Возвращает <code>true</code> , если указанная переменная — целое число
<code>is_double(\$x)</code>	Возвращает <code>true</code> , если указанная переменная — вещественное число
<code>is_string(\$x)</code>	Проверяет, является ли переменная строкой
<code>is_array(\$x)</code>	Проверяет, является ли переменная массивом
<code>is_object(\$x)</code>	Возвращает <code>true</code> , если указанная переменная — объект
<code>is_bool(\$x)</code>	Проверяет, является ли переменная логической переменной
<code>gettype(\$x)</code>	Возвращает строку, которая описывает тип переменной: "boolean" — логическое значение; "integer" — целое значение; "double" — вещественное значение; "string" — строка; "array" — массив; "object" — объект; "resource" — ресурс (например, результат обработки MySQL-запроса); "NULL" — переменная не имеет значения (не инициализирована), следовательно, невозможно установить ее тип; "unknown type" — неизвестный тип (тип определить не удалось)

3.1.3. Булевы переменные

Отдельного разговора заслуживают логические переменные. Мы привыкли, что в других языках программирования значения `true` и `false` ассоциируются соответственно со значениями 1 и 0. Но в PHP значение `false` ассоциируется с пустой строкой, а `true` — с любым ненулевым значением, например:

```
$a = 10001;
if ($a == true) echo 'Переменная $a — истинна';
```


ПРИМЕЧАНИЕ

Обратите внимание, что в операторе `echo` используются одинарные кавычки, а не двойные.

Поскольку со значением `true` ассоциируется любое ненулевое значение, то мы увидим следующую строку:

Переменная `$a` — истинна

Рассмотрим еще один пример:

```
$a = 1001; $b = true;
if ($a == $b) echo '$a = $b';           // сравниваем переменные
echo "\n".gettype($b);                  // выводим тип переменной $b
```

Самое интересное здесь — сообщение сценария, а он сообщит, что переменные `$a` и `$b` равны (рис. 3.1). Это происходит потому, что переменной `$b` присвоен логический тип (это так и есть), и PHP считает, что и переменная `$a` тоже логического типа.

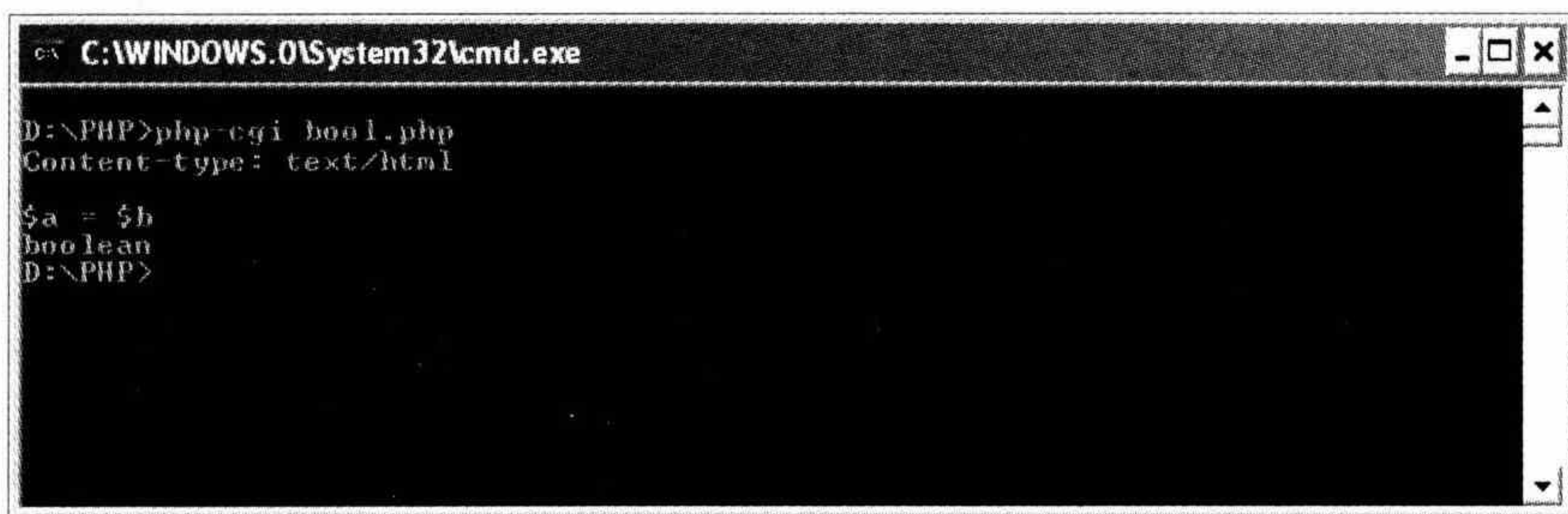


Рис. 3.1. Сравнение логической и целой переменных

3.1.4. Операции над переменными

Над переменной вы можете выполнить всего три операции:

- присвоить значение переменной — константу, результат выполнения функции или выражения:

```
$x = 1;
$x = pi;
$x = func($y);
$x = ($b * 2) / 100;
```

- уничтожить переменную — для этого служит функция `unset`:

```
unset($a);
```

- проверить существование переменной — используется функция `isset`:

```
if (isset($x)) echo 'Переменная $x установлена';
```


3.1.5. Ссылки

Если вы знакомы с программированием на языке C, то, очевидно, знакомы с указателями. В PHP нет указателей, зато есть ссылки. Ссылки можно расценивать как псевдонимы переменных. Предположим, что у вас есть переменная-массив `$arr1` объемом 1 Мбайт. Если вы присвоите ее значение другой переменной, `$arr2`, то у вас окажутся два одинаковых массива общим размером 2 Мбайт.

Ссылки позволяют обращаться к одним и тем же данным под разными именами:

```
$a = 100;
$link_on_a = &$a;

echo $link_on_a;

echo " ";      // пробел

$link_on_a = 5;
echo $a;
```

Сначала мы объявляем переменную `$a`. Затем объявляем ссылку `$link_on_a` на переменную `$a`. Синтаксис объявления ссылки следующий:

```
ссылка = &переменная;
```

После чего выводим переменную `$link_on_a`. Она равна 100, как и следовало ожидать. Теперь мы присваиваем ссылке `$link_on_a` значение 5 и выводим значение переменной `$a`. Сценарий выведет значение 5 (рис. 3.2).

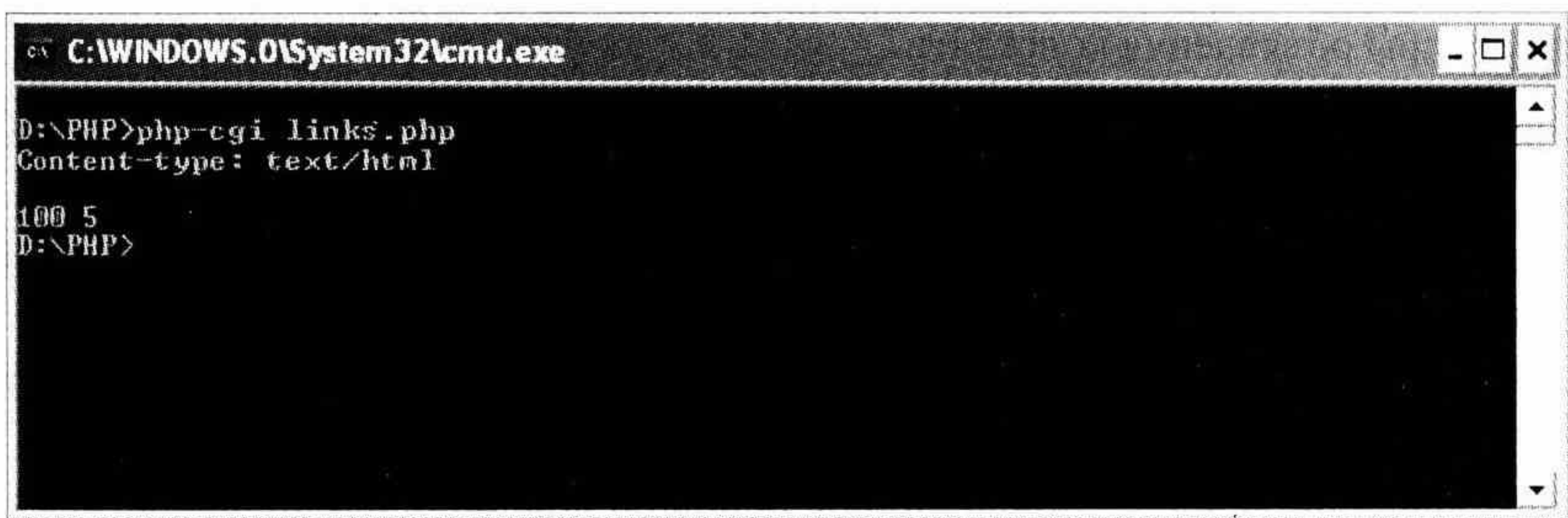


Рис. 3.2. Пример использования ссылок

Приведенные ссылки называются *жесткими*. Существуют еще *мягкие* (символические) ссылки. Символическая ссылка — это не псевдоним переменной, а переменная, содержащая имя другой переменной.

Работать с мягкими ссылками нужно следующим образом. Сначала надо создать переменную, на которую будет указывать ссылка. Затем другой переменной присвоить имя первой переменной. Чтобы создать мягкую ссылку, следует обратиться ко второй переменной так:

```
$$переменная
```


Рассмотрим следующий фрагмент кода:

```
$a = 100;                // исходная переменная

$soft_link = "a";        // эта переменная будет играть роль
                          // ссылки

echo $$soft_link;        // это уже ссылка, выведет 100

echo " ";                // выводим пробел

echo $soft_link;         // выведет просто 'a' (без кавычек)

$$soft_link = 5;         // присваиваем ссылке значение 5

echo " ";                // еще один пробел

echo $a;                 // выводим переменную $a, будет выведено 5
```

3.2. Константы

Константы содержат постоянные значения, изменить которые в процессе выполнения программы вы не можете. Для объявления констант используется функция `define()`, синтаксис ее таков:

```
define(имя, значение, чувствительность_к_регистру);
```

Вот примеры объявления констант:

```
define("const", "константа", true);
define("A", "1");
```

Первый параметр функции `define()` — имя константы, второй — значение. Третий параметр особый. Если он `true`, имя константы будет чувствительно к регистру. По умолчанию константы чувствительны к регистру, т. е. `CONST` и `const` — это две разные константы.

При использовании константы указывать знак доллара не нужно:

```
// выводим константы
echo const;
echo A;
```

Вы можете использовать следующие стандартные константы:

- ❑ `__FILE__` — имя текущего сценария (обратите внимание: до и после `FILE` — по два знака подчеркивания);
- ❑ `__DIR__` — каталог, в котором находится сценарий;
- ❑ `__LINE__` — строка, обрабатываемая в данный момент;

❑ `PHP_OS` — название операционной системы, под которой выполняется PHP;

❑ `PHP_VERSION` — версия PHP.

Ознакомиться с другими PHP-константами можно по адресу:

<http://php.net/manual/en/reserved.constants.php>.

Просмотреть список констант и их значений можно с помощью функции `get_defined_constants()`:

```
print_r(get_defined_constants());
```

3.3. Выражения и операции

3.3.1. Что такое выражение

Выражение — это часть инструкции или сама инструкция, возвращающая значение. Практически все инструкции программы являются выражениями. Даже оператор присваивания:

```
$a = 1;
```

Результат этой операции — 1, он будет присвоен переменной `$a`. Само выражение в этом случае выглядит так:

```
1
```

Выражением может быть не только значение, но и инициализированная переменная:

```
$a = 1;  
$b = $a;
```

В PHP оператор присваивания может выглядеть довольно необычно:

```
$a = $b = 1;
```

Интересно, что в выражении можно даже инициализировать переменную:

```
$a = 50 * ($c = 7) * $c;
```

3.3.2. Арифметические операции

Как и в любом другом языке программирования, вы можете выполнять следующие арифметические операции:

❑ `+` — сложение;

❑ `-` — вычитание;

❑ `*` — умножение;

❑ `/` — деление;

❑ `A % B` — остаток от деления `A` на `B`.

Кроме того, можно использовать операции инкремента и декремента:

- `$a++` — увеличить переменную `$a` на 1;
- `$b--` — уменьшить переменную `$b` на 1.

3.3.3. Логические выражения

Логическим называется выражение, результатом которого является истина или ложь. Обычно в логических выражениях применяются следующие операторы сравнения:

- `<` — меньше;
- `>` — больше;
- `!=` — не равно;
- `==` — равно;
- `<=` — меньше или равно;
- `>=` — больше или равно.

Также вы можете использовать следующие сугубо логические операции:

- `&&` — бинарная операция И (AND) истинна, если оба операнда истинны;
- `||` — бинарная операция ИЛИ (OR) истинна, если один из операндов `true`;
- `!` — унарная операция отрицания (NOT).

Примеры использования логических операций:

```
$a = true;
$b = false;
$c = $a || $b; // $c = true;
$d = $a && $b; // $d = false;
$e = !$a;      // $e = false;
```

3.3.4. Приоритеты операций

Операции в PHP выполняются в соответствии с их приоритетом, приведенным в табл. 3.2. Для изменения порядка выполнения операций можно использовать скобки, например:

```
$a = (2 + 2) * 2;
```

Таблица 3.2. Приоритет операций

Приоритет	Порядок выполнения	Операторы
13	справа налево	<code>=</code> , <code>+=</code> , <code>-=</code> , <code>*=</code> , <code>/=</code> , <code>%=</code> , <code>>>=</code> , <code><<=</code> , <code>&=</code> , <code>^=</code> , <code> =</code>
12	слева направо	<code> </code>
11	слева направо	<code>&&</code>
10	слева направо	<code> </code>

Таблица 3.2 (окончание)

Приоритет	Порядок выполнения	Операторы
9	слева направо	^
8	слева направо	&
7	слева направо	==, !=
6	слева направо	<, <=, >, >=
5	слева направо	<<, >>
4	слева направо	+, -
3	слева направо	*, /, %
2	справа налево	++(префикс), --(префикс)
1	слева направо	(постфикс)++, (постфикс)--

ПРИМЕЧАНИЕ

Приоритет операций в табл. 3.2 представлен от низшего к высшему.

ПРИМЕЧАНИЕ

Операции >> и << называются *битовыми*. Мы их не рассматриваем, поскольку при веб-разработке они используются очень редко, а на роль системного языка программирования (как C) PHP явно не тянет!

3.3.5. Операторы эквивалентности == и ===

Для сравнения значений служит оператор равенства ==, например:

```
if ($a == $b) ...
```

Иногда этот оператор работает не совсем так, как мы ожидаем. Рассмотрим следующий код:

```
$x = 1;
$y = 1;

$a = 0;
$b = "";

if ($x == $y) echo 'Переменные $x и $y равны';
if ($a == $b) echo 'Переменные $a и $b равны';
```

Переменные \$x и \$y действительно равны, поэтому мы увидим сообщение:

```
Переменные $x и $y равны
```

Но переменные \$a и \$b вообще не могут быть равны, они даже разных типов, однако PHP сообщает, что они равны:

```
Переменные $a и $b равны
```


Почему так происходит? Просто RНР воспринимает переменные `$a` и `$b` как логические. А 0 и пустая строка соответствуют `false`, поэтому RНР и сообщает, что переменные равны.

Чтобы избежать подобной неразберихи, нужно использовать оператор сравнения `===` (три знака равенства):

```
if ($a === $b) echo 'Переменные $a и $b равны';
```

В этом случае строка 'Переменные `$a` и `$b` равны' не будет выведена.

Помните, что для оператора `===` не предусмотрен обратный оператор `!==`, вместо него вы должны использовать следующую конструкцию:

```
if (!( $a === $b )) echo '$a <> $b';
```

3.3.6. Операции со строками

Первым делом нужно поговорить о кавычках, в которые заключаются строковые значения:

```
$a = 10;  
$string1 = "Значение переменной: $a";  
$string2 = 'Значение переменной: $a';
```

После выполнения этих операторов значение переменной `string1` будет следующим:

```
Значение переменной: 10
```

А значение переменной `string2` таким:

```
Значение переменной: $a
```

Как видите, между двойными и одинарными кавычками большая разница. Если строка заключена в одинарные кавычки, то все символы строки воспринимаются как есть, кроме последовательностей `\'` и `\\` (первая последовательность означает апостроф, а вторая — слэш). Строка, заключенная в двойные кавычки, позволяет вывести значение переменной.

В строках вы можете использовать следующие специальные символы:

- ☐ `\n` — символ новой строки;
- ☐ `\t` — символ табуляции;
- ☐ `\r` — символ возврата каретки;
- ☐ `\$` — знак доллара.

Операций над строками всего две: обращение к символу строки и конкатенация (слияние строк). Обратиться к определенному символу строки можно так:

```
$имя[номер]
```

Например:

```
$string[1]           // обращение к первому символу строки
```


Конкатенация (слияние) строк выполняется с помощью оператора `.`:

```
$s1 = "Hello, ";  
$s2 = "world!";  
$string = $s1 . $s2; // $string = "Hello, world!"
```

При желании вы можете считать в строку вывод системной команды. Делается это с помощью обратных апострофов (которые находятся на одной клавише с тильдой `~`):

```
$dir = `dir`; // в Windows  
$dir = `ls`;  // в Linux  
echo $dir;
```

3.4. Условный оператор

Полный вид синтаксиса условного оператора в PHP выглядит следующим образом:

```
if (условие1)  
    оператор1;  
elseif (условие2)  
    оператор2;  
...  
elseif (условиеN)  
    операторN;  
else  
    операторN+1;
```

Условие — это логическое выражение. Как работает данный оператор, думаю, понятно. Если истинно первое условие, то выполняется оператор1, и действие условного оператора прекращается. Если истинно условие2, то выполняется оператор2 и т. д. Если ни одно условие не истинно, тогда выполняется операторN+1.

Полная форма оператора используется далеко не всегда. Чаще она сокращается за счет блоков `elseif` до следующего вида:

```
if (условие1)  
    оператор1;  
else  
    оператор2;
```

Минимальная форма условного оператора выглядит следующим образом:

```
if (условие1)  
    оператор1;
```

Рассмотрим небольшой пример использования условного оператора:

```
$a = 10;  
$b = 7;  
  
if ($a > $b)  
    echo "A > B";  
else  
    echo "A < B";
```


3.5. Циклы

PHP поддерживает четыре вида циклов:

- ❑ цикл со счетчиком `for`;
- ❑ цикл с предусловием `while`;
- ❑ цикл с постусловием `do-while`;
- ❑ специальный цикл переработки массивов `foreach`.

В этой главе мы обсудим первые три цикла, а последний цикл будет рассмотрен в главе 9.

3.5.1. Цикл со счетчиком *for*

Цикл со счетчиком `for` нужно использовать тогда, когда известно количество итераций цикла. Синтаксис цикла `for` следующий:

```
for (оператор_инициализация; условие; поститерационная_команда) {  
    тело_цикла;  
}
```

Перед первой итерацией выполняется оператор инициализации, обычно это инициализация счетчика цикла.

Затем проверяется условие. Если оно истинно, выполняется тело цикла, а после него — поститерационная команда.

Рассмотрим небольшой пример цикла `for`:

```
for ($i=1; $i <= 5; $i++) echo $i;
```

Цикл выведет: 12345.

Если нужно выполнить несколько команд до запуска цикла и после итерации, их можно разделить запятыми:

```
for ($i=1, $d = 1; $i <= 5; $i++, $t--) echo $i;
```

3.5.2. Цикл *while*

Цикл `while` еще называется циклом с предусловием, поскольку сначала вычисляется условие, а потом выполняются операторы (тело цикла). Рассмотрим синтаксис цикла `while`:

```
while (условие) {  
    тело;  
}
```

Сначала вычисляется условие цикла. Если оно истинно, то выполняется тело цикла. Вот пример цикла `while`:


```
$i = 1;
while ($i < 6) {
    echo $i;
    $i++;
}
```

Как и в предыдущем случае, цикл напечатает 12345.

3.5.3. Цикл *do-while*

В отличие от цикла *while*, цикл *do-while* сначала выполняет тело цикла, а потом проверяет условие. Вот синтаксис цикла *do-while*:

```
do {
    тело;
} while (условие);
```

Особенность этого цикла заключается в том, что тело цикла будет выполнено хотя бы один раз.

Рассмотрим небольшой пример:

```
$i = 1;
do {
    echo $i;
    $i++;
} while ($i <= 5);
```

Как вы уже догадались, наш цикл снова выведет строку 12345.

3.5.4. Принудительное завершение цикла и пропуск итерации

Предположим, что у нас есть некая функция `func()`. Давайте напишем цикл, который будет вызывать эту функцию некоторое количество раз. Если возвращенное функцией `func()` значение будет равно 5, тогда нам нужно завершить цикл. Если же `func()` возвратила 4, то нам нужно пропустить итерацию, т. е. ничего не делать, и сразу перейти к следующей итерации цикла.

```
$i = 1;
while ($i < 10) {

    $k = func();

    if ($k == 4) continue;           // условие перехода на следующую итерацию
    if ($k == 5) break;              // условие останова цикла

    echo $k;                         // выводим $k

}
```


Оператор `break` принудительно завершает работу цикла. Обратите внимание на условие нашего цикла — он должен выполняться, пока `$i` меньше 10. Поскольку переменная `$i` равна 1 и не изменяется в процессе выполнения цикла, то наш цикл выполнялся бы вечно, если бы не условие останова цикла и оператор `break`.

Оператор `continue` завершает не весь цикл, а только текущую итерацию, и переходит к следующей итерации цикла.

3.6. Оператор выбора *switch-case*

Условный оператор довольно удобен, но когда условий слишком много и все они одного типа, тогда намного удобнее использовать оператор выбора *switch-case*. Рассмотрим синтаксис этого оператора:

```
switch (выражение) {  
  case значение1: операторы1; [break;]  
  case значение2: операторы2; [break;]  
  . . .  
  case значениеN: операторыN; [break;]  
  [default: операторы_по_умолчанию; [break;]]  
}
```

Разберемся, как работает оператор *switch-case*. Сначала вычисляется значение выражения. Затем просматривается список значений. Если найдено совпадение, то выполняются соответствующие значению операторы. Если после этих операторов не будет указан оператор `break`, будут выполнены все последующие операторы и операторы по умолчанию.

Если совпадений не найдено, будут выполнены операторы по умолчанию, если они были заданы.

Теперь рассмотрим два примера кода. Первый — это пример обычного условного оператора `if`:

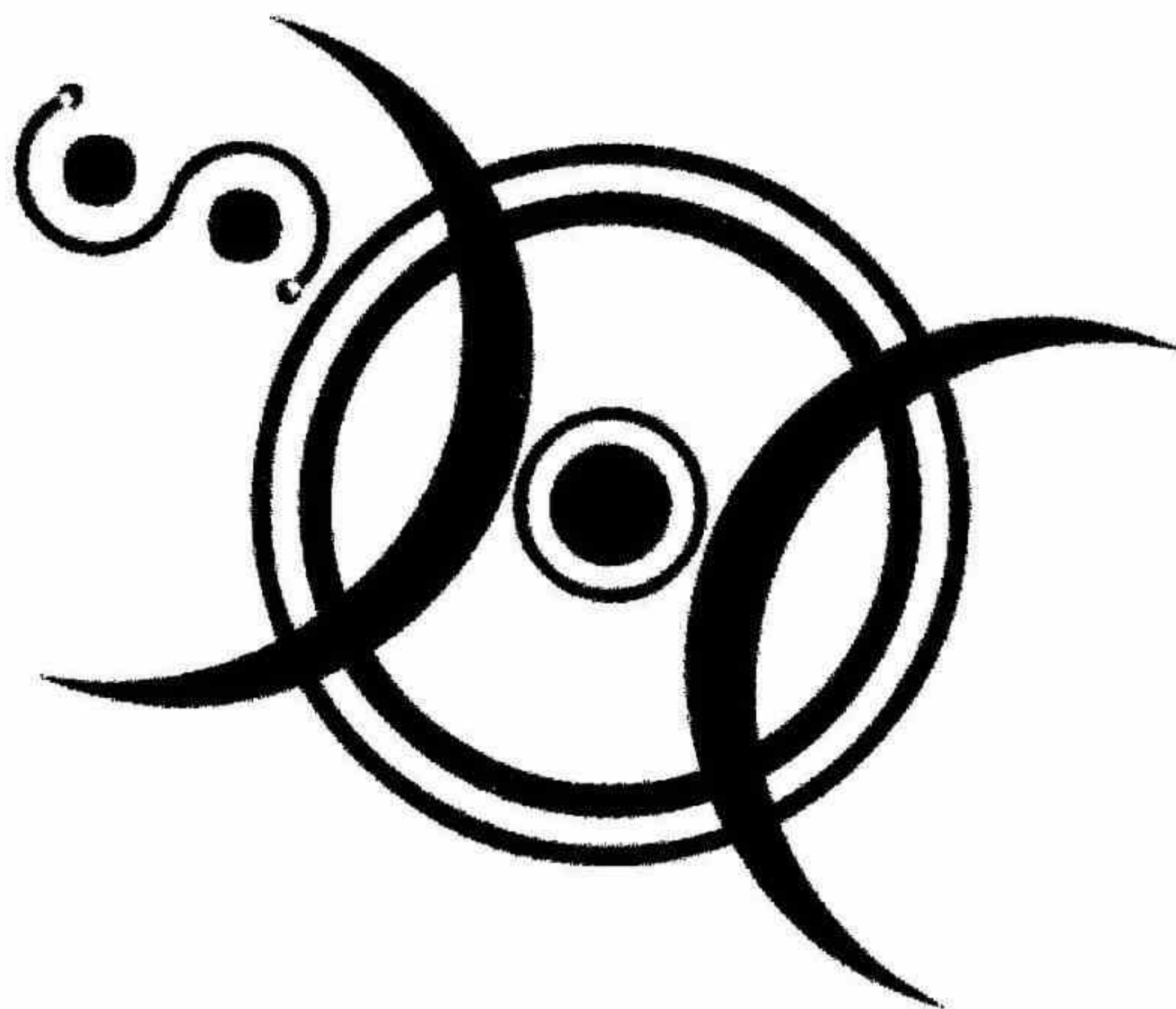
```
$x=2;  
  
if ($x == 0) {  
    echo "x=0";  
} elseif ($x == 1) {  
    echo "x=1";  
} elseif ($x == 2) {  
    echo "x=2";  
}
```

Второй пример — это аналогичный ему оператор *switch-case*:

```
$x=2;  
switch ($x) {  
  case 0:  
    echo "x=0";  
    break;
```



```
case 1:
    echo "x=1";
    break;
case 2:
    echo "x=2";
    break;
}
```

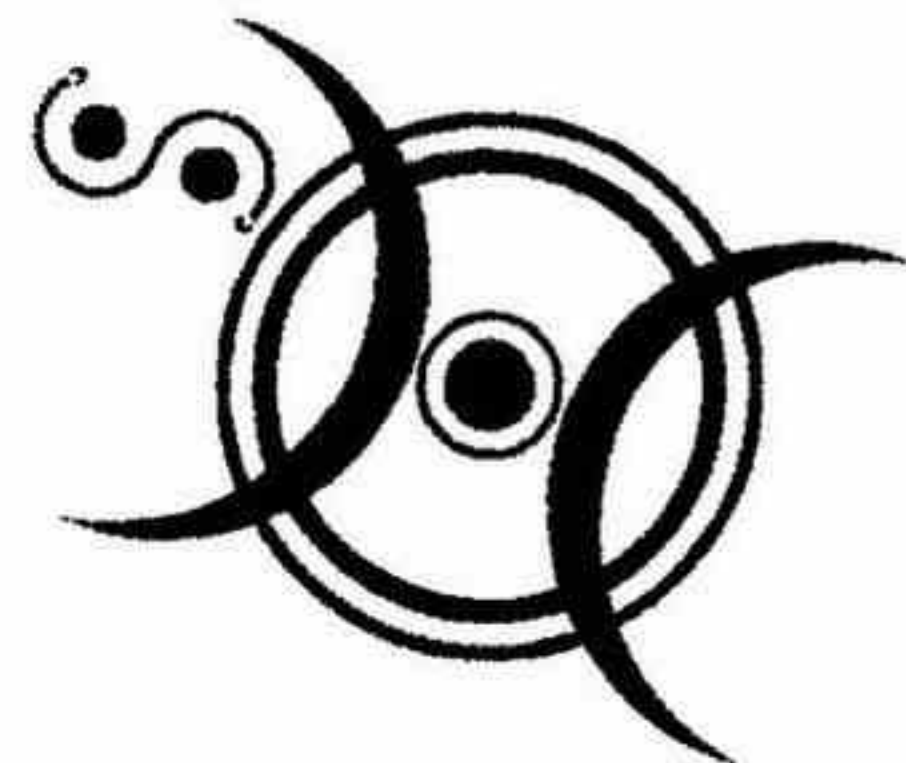



I. РАЗДЕЛ 2

Передача параметров PHP-программам

Глава 4.	Методы <i>GET</i> и <i>POST</i>
Глава 5.	Протокол HTTP и интерфейс CGI
Глава 6.	Передача параметров посредством HTML-формы
Глава 7.	Запоминание параметров с помощью Cookies и сессий
Глава 8.	Отдельное слово о директиве <i>register_globals</i>

ГЛАВА 4



Методы *GET* и *POST*

4.1. Интерфейс CGI

CGI (Common Gateway Interface) — общий интерфейс шлюза, используемый для "общения" веб-клиентов (браузеров) с веб-серверами. В этой главе мы рассмотрим интерфейс CGI вкратце, поскольку полученные знания при программировании на PHP вам пригодятся лишь частично.

Итак, предположим, что у нас есть веб-сервер. Какой именно и с какой операционной системой он работает — не имеет значения. Это может быть как Apache, работающий под управлением FreeBSD, так и Microsoft Internet Information Server, запущенный в Windows 2003 Server.

В каталог `cgi-bin` мы помещаем простейшую программу, работающую со стандартным вводом и выводом. Эта программа не имеет графического интерфейса, все входные данные вводятся пользователем с клавиатуры. Вывод программа производит на экран. С такой программой можно работать только в режиме командной строки, иначе вы попросту не увидите результат ее работы.

В любой операционной системе есть средства перенаправления ввода/вывода. То есть вы можете вывод одной программы перенаправить на ввод другой. Ну, или запустить программу и передать данные на ее стандартный ввод. Следовательно, веб-сервер может запустить нашу программу, передать ей информацию, затем прочитать вывод этой программы и передать его в браузер.

На каком языке написана программа, не имеет значения. Это может быть исполняемый файл (`exe`) или обычный сценарий (расширение `bat` в Windows). В Linux/UNIX тоже нет ограничений на используемый язык программирования. Для большей определенности представим, что наша программа называется `example.cgi`.

Вот теперь у вас уже почти вырисовалась картина работы интерфейса CGI. Пользователь запускает браузер и вводит следующий адрес (URL):

```
http://localhost/cgi-bin/example.cgi?name=denis
```

Этот адрес означает, что пользователь пытается запустить нашу программу и передать ей параметр `name` со значением `denis`.

Получив такой URL, наш веб-сервер пытается найти и запустить программу `example.cgi`, передав ей на стандартный ввод параметры, указанные пользователем. Программа запускается и обрабатывает полученные параметры. Затем она выводит результат обработки параметров на стандартный вывод. Веб-сервер перехватывает вывод программы и отправляет его в браузер пользователя. Пользователь видит результат выполнения программы!

Нашей программе кажется, что параметры передал ей не сервер, а пользователь. И она просто выводит данные, ничего не подозревая о том, что они будут перехвачены и отправлены еще куда-то.

Благодаря такой "прозрачности" CGI-приложение можно написать на любом языке программирования. Для написания приложений не нужны какие-либо дополнительные библиотеки. С одной стороны — это преимущество, а с другой — недостаток. Но если у языка программирования есть средства для разработки CGI-приложений, то их разработка становится намного проще. Язык программирования PHP обладает такими средствами. Вы только взгляните на листинг 4.1 — в нем представлено самое простое CGI-приложение, которое всего лишь выводит переданные ему параметры.

Листинг 4.1. Пример простого CGI-приложения

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *Query_String = getenv("QUERY_STRING");

    printf("Content-type: text/html\n\n");
    printf("<p>Переданные сценарию параметры: %s", Query_String);
    return 0;
}
```

Переменная окружения `QUERY_STRING` содержит строку параметров, т. е. (в нашем случае):

```
name=denis
```

Если параметров больше, то строка параметров будет выглядеть сложнее, например:

```
name=denis&age=25
```

Наше приложение несовершенно. Здесь еще предстоит сделать разбор параметров, т. е. разобрать строку параметров на пары "параметр–значение".

В PHP все это делать не нужно. Вам достаточно обратиться к параметру `name` вот так:

```
$_GET['name']
```


Существуют два метода передачи сценариям параметров: GET и POST. Далее мы рассмотрим оба метода.

4.2. Метод GET

Сначала мы рассмотрим метод GET, как более простой. Предположим, что у нас есть сценарий `hello.php`, которому нужно передать параметр `name`:

```
http://server.ru/hello.php?name=denis
```

Браузер выделяет из введенного URL протокол (`http`), имя сервера (`server.ru`), имя сценария. Затем браузер устанавливает подключение с сервером по заданному протоколу и передает вот такой запрос:

```
GET hello.php HTTP/1.0\n
различные заголовки
\n\n
```

У метода GET есть одна особенность — все параметры передаются через строку адреса браузера, и пользователь их видит невооруженным глазом. А поэтому данный метод нельзя использовать для передачи секретной информации — например, паролей. Да и максимальная длина строки в методе GET ограничивается 255 символами, поэтому вряд ли вы будете использовать его для передачи больших объемов информации — файлов или длинных текстов.

Получить доступ к параметру, переданному методом GET, можно так:

```
$_GET['имя']
```

Например:

```
// присваиваем переменной $name значение параметра name
$name = $_GET['name'];
```

4.3. Метод POST

При использовании метода POST передаваемые данные не отображаются в окне браузера. Запрос выглядит так:

```
POST hello.php HTTP/1.0\n
Content-Length: 10\n
\n
0123456789
```

Как видите, передаваемые сценарию данные отображаются уже после самого запроса. Заголовок запроса `Content-Length` указывает, сколько байтов нужно прочесть серверу.

Метод POST обычно используется для передачи больших объемов данных — например, файлов. А поскольку передаваемые данные не отображаются в браузере, его можно использовать для передачи паролей.

Получить доступ к параметрам, переданным методом POST, можно через массив

`$_POST`:

`$_POST['имя']`

Например:

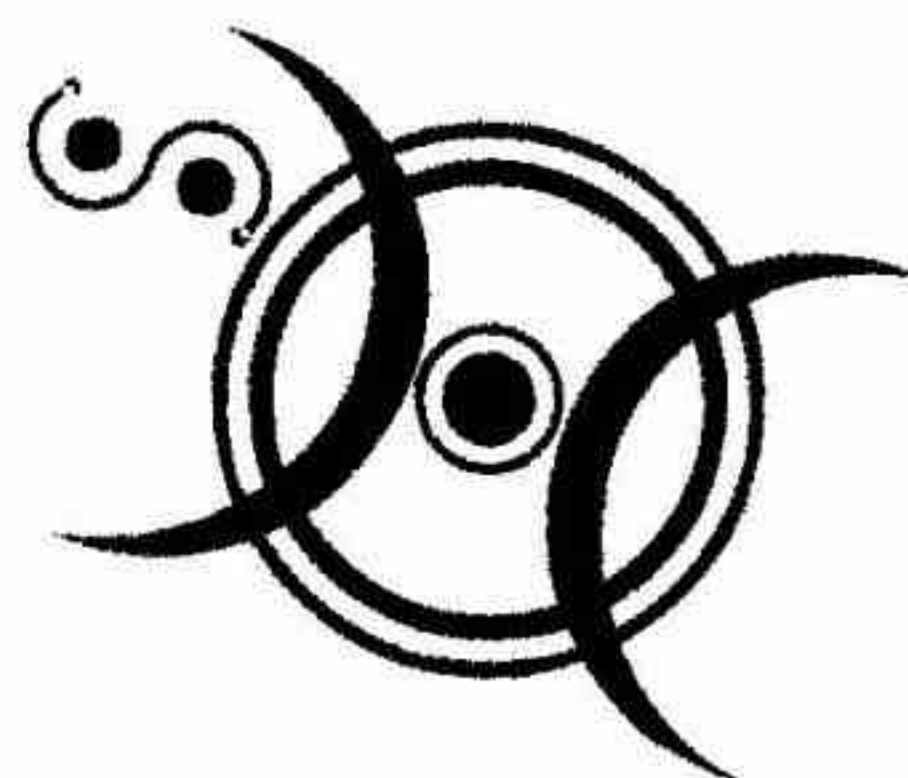
```
// присваиваем переменной $name значение параметра name
$name = $_POST['name'];
```

О передаче файлов методом POST мы еще поговорим в *главе 26*. Также рекомендую внимательно прочитать *главы 6* и *8*. В *главе 6* рассказано об использовании форм для передачи параметров (к слову, POST-параметры можно передать только с помощью формы или специальных программ), а в *главе 8* — о директиве `register_globals`, которая была исключена из шестой версии PHP, но о ней вы просто должны знать!

Массив `$_REQUEST`

Кроме массивов `$_GET` и `$_POST` в PHP есть массив `$_REQUEST`, объединяющий в себе массивы `$_GET`, `$_POST` и `$_COOKIE`. Если вы не знаете, откуда "пришла" переменная (была передана методом GET или POST или получена через Cookies), тогда используйте этот массив: `$_REQUEST['имя']`. Подробно о массиве `$_REQUEST` мы поговорим в *главе 8*.

ГЛАВА 5



Протокол HTTP и интерфейс CGI

5.1. Специальные переменные окружения CGI

При создании серьезных PHP-сценариев не обойтись без специальных переменных окружения CGI. Например, переменная `REMOTE_ADDR` содержит IP-адрес удаленного пользователя. Получить значение переменной окружения можно с помощью функции `getenv()`:

```
echo getenv("REMOTE_ADDR");
```

Самые полезные переменные окружения представлены в табл. 5.1.

Таблица 5.1. Полезные переменные окружения

Переменная	Значение
<code>HTTP_REFERER</code>	Адрес сайта, с которого пользователь пришел на ваш сайт
<code>HTTP_USER_AGENT</code>	Браузер пользователя
<code>REMOTE_HOST</code>	Доменное имя удаленного компьютера
<code>REMOTE_PORT</code>	Удаленный порт (по этому порту браузер будет ждать ответа от сервера)
<code>QUERY_STRING</code>	Строка параметров, переданная сценарию (используется редко)
<code>SCRIPT_NAME</code>	Путь к сценарию
<code>SERVER_ADDR</code>	IP-адрес сервера
<code>SERVER_NAME</code>	Доменное имя сервера
<code>SERVER_PORT</code>	Порт сервера, обычно 80
<code>SERVER_PROTOCOL</code>	Версия HTTP-протокола

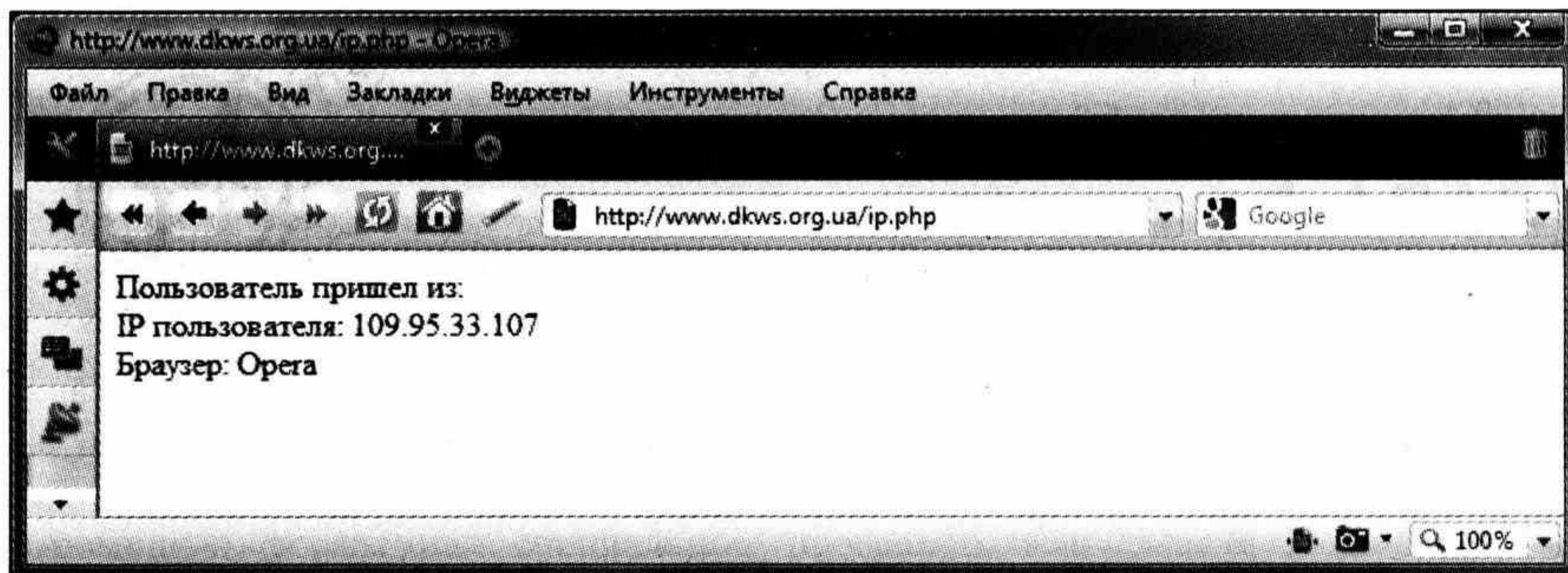


Рис. 5.1. Определение IP-адреса и браузера пользователя

Рассмотрим пример использования этих переменных окружения (рис. 5.1):

<?

```
echo "Пользователь пришел из: ".getenv("HTTP_REFERER")."<BR>";
```

```
echo "IP пользователя: ".getenv("REMOTE_ADDR")."<BR>";
```

```
$browser=getenv("HTTP_USER_AGENT");
```

```
if(strpos($browser,"Opera")!=false) echo "Браузер: Opera<BR>";
```

```
else echo "Другой браузер ( $browser )<BR>";
```

?>

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *ip.php*), представленный в каталоге *Glava_5* сопровождающего книгу электронного архива (см. приложение 4).

5.2. Заголовки протокола HTTP

Познакомимся с заголовками протокола HTTP, которые вам рано или поздно придется использовать в своих сценариях. Например, заголовок *Location* служит для перенаправления пользователя на другую страницу, а заголовки *GET/POST* — для получения файлов по протоколу HTTP (об этом мы поговорим в главе 16).

Вывести заголовки можно с помощью функции *Header()*, например:

```
// перенаправляем браузер пользователя на сайт http://dkws.org.ua
Header("Location: http://dkws.org.ua");
```

В табл. 5.2 представлены основные заголовки протокола HTTP.

Таблица 5.2. Заголовки протокола HTTP

Заголовок	Значение
Accept	Отправляется браузером серверу для информирования последнего о поддерживаемых MIME-типах, например <code>Accept: text/plain</code> . В данном случае поддерживается только обычный текст. Этот заголовок используется редко, и если вы не собираетесь разрабатывать собственный браузер, он вам не понадобится
Content-type	Информирует браузер о формате выводимых данных. Часто используемые форматы: • <code>text/plain</code> — обычный текст; • <code>text/html</code> — текст в формате HTML; • <code>image/jpeg</code> — изображение в формате JPEG; • <code>image/png</code> — изображение в формате PNG; • <code>image/gif</code> — изображение в формате GIF
Content-Length	Если используется передача данных серверу методом POST, то этот заголовок содержит длину передаваемых данных
Cookie	Содержит все Cookies, о них мы поговорим в главе 7
GET	Используется для получения файла методом GET, вот синтаксис запроса: <code>GET файл_или_сценарий?параметры HTTP/1.0</code>
Location	Перенаправление на другой сайт
POST	Используется для получения файла методом POST. Синтаксис такой же, как у заголовка GET
Pragma	Используется для всевозможных целей, чаще всего для запрета кэширования страницы: <code>Pragma: no-cache</code>
Referer	Содержит адрес сайта, откуда пользователь пришел на ваш сайт
User-Agent	Содержит информацию об используемом браузере, например: <code>Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) Opera 7.54 [en]</code>

5.3. Коды ответов протокола HTTP

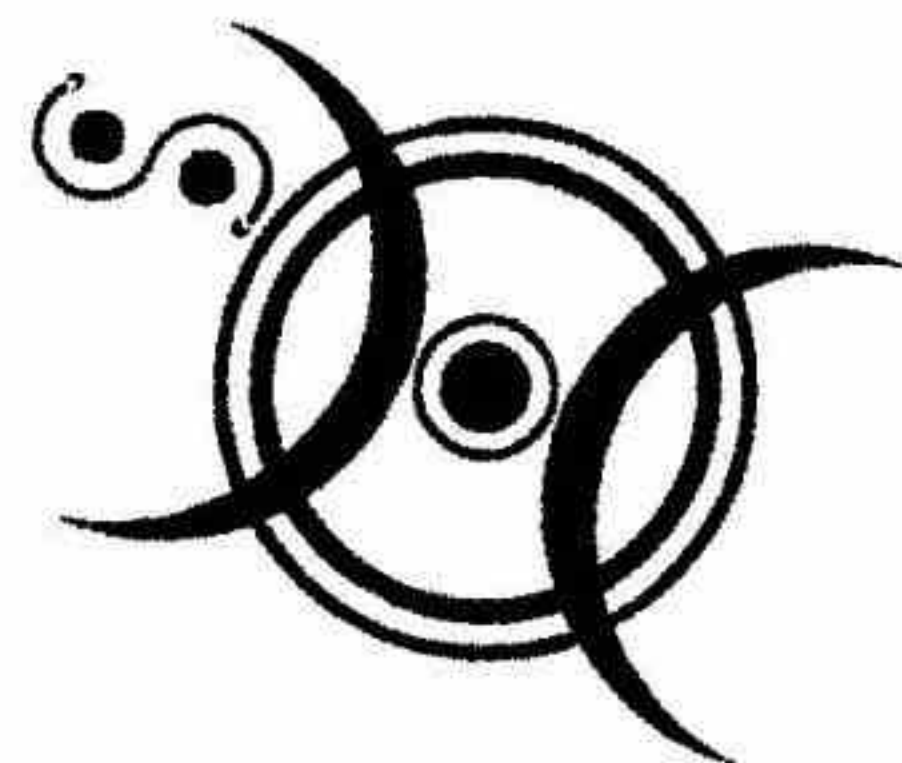
Наверняка каждый пользователь Интернета сталкивался с ошибкой 404, которая генерируется, если запрашиваемая страница не найдена. Вообще-то, код 404 — не ошибка, а обычный ответ сервера, который сообщает вам, что запрошенной страницы не существует. Тем не менее, коды из диапазона 4xx называются *ошибками клиента*. Код 404 попадает в эту категорию — пользователь ошибся при вводе адреса страницы сайта. Код 402 означает, что доступ к сайту платный. Иногда все ответы 4xx интерпретируются как ошибка 404 — все зависит от настроек сервера.

Кроме ответов 4xx есть и другие коды — например, коды из диапазона 2xx являются *кодами успеха*. Вы никогда их не увидите, поскольку если от сервера пришел код 200, значит, страница существует и отправлена в браузер. Понятное дело, что браузер выводит в этом случае не код ответа, а сразу полученную страницу.

Коды 5xx — это *ошибки сервера*. Например, сервер выводит ошибку 500 в случае, когда вы пытаетесь запустить сценарий, при выполнении которого возникла ошибка, и сервер больше не может продолжать его нормальное выполнение.

Кстати, коды 1xx называются *информационными* — они передают различную информацию, которая так или иначе относится к запросу, а коды 3xx — это коды *перенаправления*.

ГЛАВА 6



Передача параметров посредством HTML-формы

6.1. Создание простейшей формы и ее обработка в сценарии

В чем заключается цель любой программы? Правильно — в обработке данных. Программа получает исходные данные, затем обрабатывает их и передает результат. Как программа передаст результат: выведет в браузер пользователя, отправит по e-mail или поместит в базу данных — нас сейчас не интересует, равно как и способы обработки данных (что может программа с ними делать). Но чтобы получить результат, нужно как минимум иметь входные данные. Вот они нас сейчас и интересуют.

Одним из часто используемых способов получения данных является ввод данных пользователем. HTML-форма (далее просто форма) предназначена для передачи введенных пользователем параметров PHP-сценарию. Форма описывается с помощью тега `FORM`:

```
<FORM action=сценарий method=метод>  
<элементы формы>  
</FORM>
```

Мы указали два обязательных параметра формы: `action` и `method`. Вообще-то, и эти параметры не обязательны, но без них форма теряет свой смысл. Описание этих двух, а также остальных параметров формы представлено в табл. 6.1.

ПРИМЕЧАНИЕ

HTML-теги можно писать как прописными, так и строчными буквами — разницы никакой нет. Это и иллюстрируют примеры в данной главе.

Таблица 6.1. Параметры формы

Параметр	Описание
action	Задаёт сценарий, которому наша форма передает параметры
method	Определяет метод передачи параметров (GET или POST)

Таблица 6.1 (окончание)

Параметр	Описание
enctype	Тип кодирования данных формы. Обычно этот параметр определять не нужно. Но если вы собираетесь передавать файлы, то для этого параметра нужно установить значение <code>multipart/form-data</code>
name	Задаёт имя формы, через которое можно обращаться к ее элементам в сценариях JavaScript/VBScript. При работе на PHP определять этот параметр не нужно
target	Позволяет определить окно, в котором будет загружаться обработчик формы (сценарий, указанный в параметре <code>action</code>): • <code>_blank</code> — открывает страницу в новом окне браузера; • <code>_self</code> — открывает страницу в текущем окне; • <code>_parent</code> — открывает страницу во фрейме-родителе; • <code>_top</code> — игнорирует все фреймы и открывает страницу в новом окне браузера; если фреймов нет, то данный параметр работает как <code>_self</code>

Предположим, что на сервере `localhost` имеется сценарий `hello.php`, которому нужно передать параметр `username`. В строке браузера это можно сделать так:

```
http://localhost/hello.php?username=denis
```

Теперь напишем HTML-форму `hello.html`, передающую сценарию параметр `username`:

```
<form action=http://localhost/hello.php method=get>
Введите ваше имя <input type=text name=username>
<input type=submit value=OK name=send>
</form>
```

В окне браузера данная форма будет выглядеть так, как на рис. 6.1.⁶

Как только пользователь введет свое имя и нажмет кнопку **ОК**, форма методом `GET` передаст параметр `username` сценарию `hello.php`.

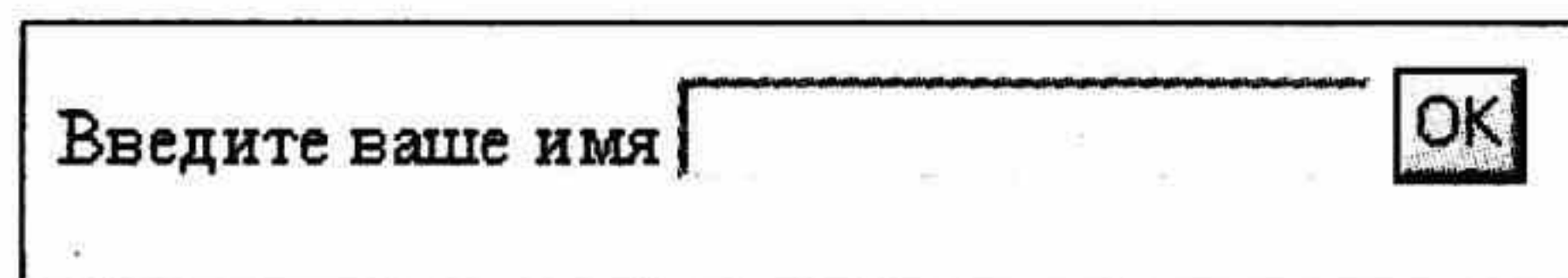


Рис. 6.1. Форма в окне браузера

Обратимся к сценарию `hello.php`. Нам нужно получить параметр `username`. Это можно сделать одним из способов:

```
echo $_GET['username'];
echo $_REQUEST['username'];
```

Первый случай нужно использовать, если вы точно знаете, что параметр передан методом `GET`. А вот если вы забыли, каким методом передаете параметры, или же просто хотите сделать более универсальный сценарий, чтобы он работал с обоими методами передачи параметров, примените второй способ.

В старых версиях PHP можно было использовать директиву `register_globals` (см. главу 8). Если она включена, то параметр `username` будет автоматически зарегистрирован как глобальная переменная `$username`:

```
echo $username;
```

Теперь напишем сценарий, который проверяет, передан ему параметр или нет. Если параметр не передан, то сценарий выводит форму. А вот если форма уже была отображена, и пользователь передал параметр, тогда мы его обработаем (просто выведем обратно в браузер). Проверить, существует ли параметр `username`, можно так:

```
if (isset($_GET['username'])) ...
```

Но лучше проверять наличие не параметра `username`, а параметра `send` — это наша кнопка. Так мы точно будем знать, что пользователь передал параметр через форму, а не вручную (листинг 6.1).

Листинг 6.1. Пример обработки параметра

```
<?php

if (!isset($_GET['send'])) {
    // выводим форму из hello.html
    echo join('', file('hello.html'));
    // и прерываем выполнение сценария
    die();
}

// обрабатываем параметры
echo $_GET['username'];

?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *6-1.php*), представленный в каталоге *Glava_6* сопровождающего книгу электронного архива (см. приложение 4).

Обратите внимание, что для получения содержимого файла мы применяем следующую конструкцию:

```
join('', file('hello.html'));
```

Функция `file()` получает содержимое файла в массив строк. Но чтобы его вывести одной строкой, т. е. для объединения массива, используется функция `join()`. Вместо этой конструкции можно применить более современную функцию `file_get_contents()`:

```
echo file_get_contents('hello.html');
```


Приведенная нами в листинге 6.1 конструкция `join('', file('hello.html'))`; будет работать на самых древних версиях PHP, а вот функция `file_get_contents()` появилась только в PHP 5. Однако учитывая, что на современных хостах работают последние версии PHP, для получения содержимого файла вы можете использовать и функцию `file_get_contents()`.

Можно было бы вывести форму оператором `echo`, а не загружать из файла, но старайтесь, чтобы ваш PHP-код содержал минимум HTML-кода. Во-первых, так PHP-код будет компактнее и его станет проще редактировать. Во-вторых, если ваш сценарий будет поддерживаться кем-то еще, например дизайнером, которому со временем придется изменить HTML-код, лучше разделять код HTML и PHP.

6.2. Создание пользовательского интерфейса с помощью формы

Совсем недавно мы познакомились с двумя элементами формы: текстовым полем ввода и кнопкой отправки параметров (она называется **Submit**). Это далеко не все элементы формы. Вы можете использовать текстовые области (`textarea`), списки, зависимые и независимые переключатели и многое другое.

У каждого элемента формы должен быть параметр `name`, иначе вы не сможете к нему обратиться в PHP-сценарии:

```
<input type=text name=my_text>
```

Желательно все значения атрибутов HTML указывать в кавычках, например:

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

Однако это создает определенные проблемы при выводе формы оператором `echo` — кавычки нужно экранировать:

```
echo "<input type=\"text\" name=\"my_text\">";
```

Одинарные кавычки использовать нельзя, потому что довольно часто нам бывает нужно задать значение по умолчанию, которое хранится в какой-нибудь переменной, а в одинарных кавычках вместо значения переменной вы увидите ее имя:

```
$val = "777";
```

```
// элемент формы будет содержать значение "777" (без кавычек)
```

```
echo "<input type=\"text\" name=\"my_text\" value=\"\$val\">";
```

```
// элемент формы будет содержать значение "$val" (без кавычек)
```

```
echo "<input type=\"text\" name=\"my_text\" value=\"$val\">";
```

Впрочем, если значения HTML-параметров не содержат пробелов, от использования кавычек можно отказаться. Но значения, выводимые из сценария в форму, нужно всегда заключать в кавычки, поскольку они могут содержать пробелы:


```
$val = "два слова";
```

```
// элемент формы будет содержать значение "два слова" (без кавычек)
echo "<input type=text name=my_text value=\"\$val\">";
```

```
// элемент формы будет содержать значение "два" (без кавычек)
echo "<input type=text name=my_text value=$val>";
```

Вот теперь можно приступить к рассмотрению элементов формы.

6.2.1. Ввод текста. Теги *INPUT* и *TEXTAREA*

Тег *INPUT* является одним из самых часто используемых. Параметры тега *INPUT* представлены в табл. 6.2.

Таблица 6.2. Параметры тега *INPUT*

Параметр	Описание
type	Задает тип текстового поля: • text — обычный текст; • hidden — скрытое поле, содержит значение, но не отображается на форме, и пользователь не может его изменить; • password — поле для ввода пароля
value	Задает значение по умолчанию
size	Размер поля (в знаках)
maxlength	Максимальная длина строки, которую можно ввести в поле
name	Имя, по которому можно обращаться к элементу ввода в сценарии

Рассмотрим небольшой пример:

```
<input type=password size=8 maxlength=16>
```

Данный тег создает поле для ввода пароля (введенный текст будет заменяться звездочками) длиной восемь знаковых мест и максимальной длиной значения — 16 символов. То есть наше поле — небольшое, оно отображает только 8 символов (8 звездочек, точнее), хотя в него можно ввести до 16 символов.

Теперь поговорим о теге *TEXTAREA*. В отличие от тега *INPUT*, который позволяет ввести всего одну строку, тег *TEXTAREA* дает возможность вводить многострочный текст:

```
<textarea параметры name=txt>
текст по умолчанию
</textarea>
```

Обратите внимание — если для тега *INPUT* не нужно было указывать закрывающий тег */INPUT*, то для тега *TEXTAREA* закрывающий тег (*/TEXTAREA*) обязателен, иначе браузер не поймет, где заканчивается форма, и будет считать текстом поля *TEXTAREA* код формы.

Параметров у тега `TEXTAREA`, если не считать стандартного параметра `name`, всего три: `cols`, `rows` и `wrap`. Первые два — это соответственно ширина и высота текстового поля в знаках. Последний параметр задает тип переноса текста:

- ☐ `virtual` — самый оптимальный вариант переноса текста с автоматической полоской прокрутки и автоматическим переносом;
- ☐ `physical` — зависит от браузера, и предсказать, что появится в том или ином браузере, невозможно;
- ☐ `none` — вообще без переноса, текст вводится как есть; не очень удобно, но иногда полезно.

6.2.2. Зависимые и независимые переключатели

Независимые переключатели потому и называются независимыми, что состояние (включен/выключен) одного из переключателей формы не зависит от всех остальных.

Значения зависимых переключателей зависят друг от друга — включенным может быть только один из переключателей. На рис. 6.2 изображена форма с зависимыми и независимыми переключателями.

Независимые переключатели	Зависимые переключатели
☒ Да, я хочу получать рассылку ☒ Да, я хочу получать сообщения от других пользователей	Пол: ☒ Мужчина ☐ Женщина

Рис. 6.2. Зависимые и независимые переключатели

Для создания переключателей служит уже знакомый нам тег `INPUT`. Тип переключателя регулируется параметром `type`:

- ☐ `checkbox` — независимый переключатель;
- ☐ `radio` — зависимый переключатель.

Кроме параметра `type` вы можете задать следующие параметры:

- ☐ `name` — имя переключателя;
- ☐ `value` — значение переключателя;
- ☐ `checked` — переключатель активен.

Рассмотрим пример создания независимых переключателей:

```
<input type=checkbox value=yes name=subscr checked> Да, я хочу получать рассылку
```

```
<input type=checkbox value=yes name=email checked> Да, я хочу получать сообщения от других пользователей
```

Если пользователь включил переключатель, то сценарию будет передан параметр: `имя_переключателя=значение`

Например, если пользователь хочет получить рассылку сайта, но не хочет получать сообщения от других пользователей, то сценарию будет передан параметр `subscr` со значением `yes` (значение устанавливается параметром `value`).

```
if (isset($_REQUEST['subscr']))
echo "Пользователь хочет получать рассылку сайта";

if (isset($_REQUEST['email']))
echo "Пользователь хочет получать сообщения пользователей";
```

Зависимые переключатели, подобные изображенным на рис. 6.2, создаются так:

```
<input type=radio value=male name=sex checked> Мужчина
<input type=radio value=female name=sex> Женщина
```

Зависимые переключатели работают немного иначе. Обратите внимание, что у обоих переключателей одно и то же имя — `sex`. Первый переключатель (со значением `male`) включен по умолчанию. Когда пользователь нажмет кнопку **ОК**, форма передаст сценарию параметр `sex`, но его значение будет зависеть от выбранного переключателя:

```
// проверяем, был ли передан параметр
if (isset($_REQUEST['sex'])) {
    // параметр передан, проверяем значение
    if ($_REQUEST['sex'] == "male") echo "Мужчина";
    else echo "Женщина";
}
```

6.2.3. Списки выбора

Тег `SELECT` позволяет организовать списки выбора. Параметр `size` задает размер списка в строках, параметр `multiple` разрешает выбор нескольких значений списка. Рассмотрим два списка (листинг 6.2, рис. 6.3).

Листинг 6.2. Два списка выбора

```
<p>Выберите год:
<select name=year size=1>
<option value=2014>2014</option>
<option value=2013>2013</option>
<option value=2012>2012</option>
<option value=2011>2011</option>
<option value=2010>2010</option>
<option value=2009>2009</option>
<option value=2008>2008</option>
<option value=2007>2007</option>
<option value=2006>2006</option>
<option value=2005>2005</option>
</select>
```



```

<p>Какая операционная система у вас установлена?
<p><select name=os[] size=3 multiple>
<option value=xp>Windows XP</option>
<option value=vista>Windows Vista</option>
<option value=linux>Linux</option>
</select>

```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *6-2.html*), представленный в каталоге *Glava_6* сопровождающего книгу электронного архива (см. приложение 4).

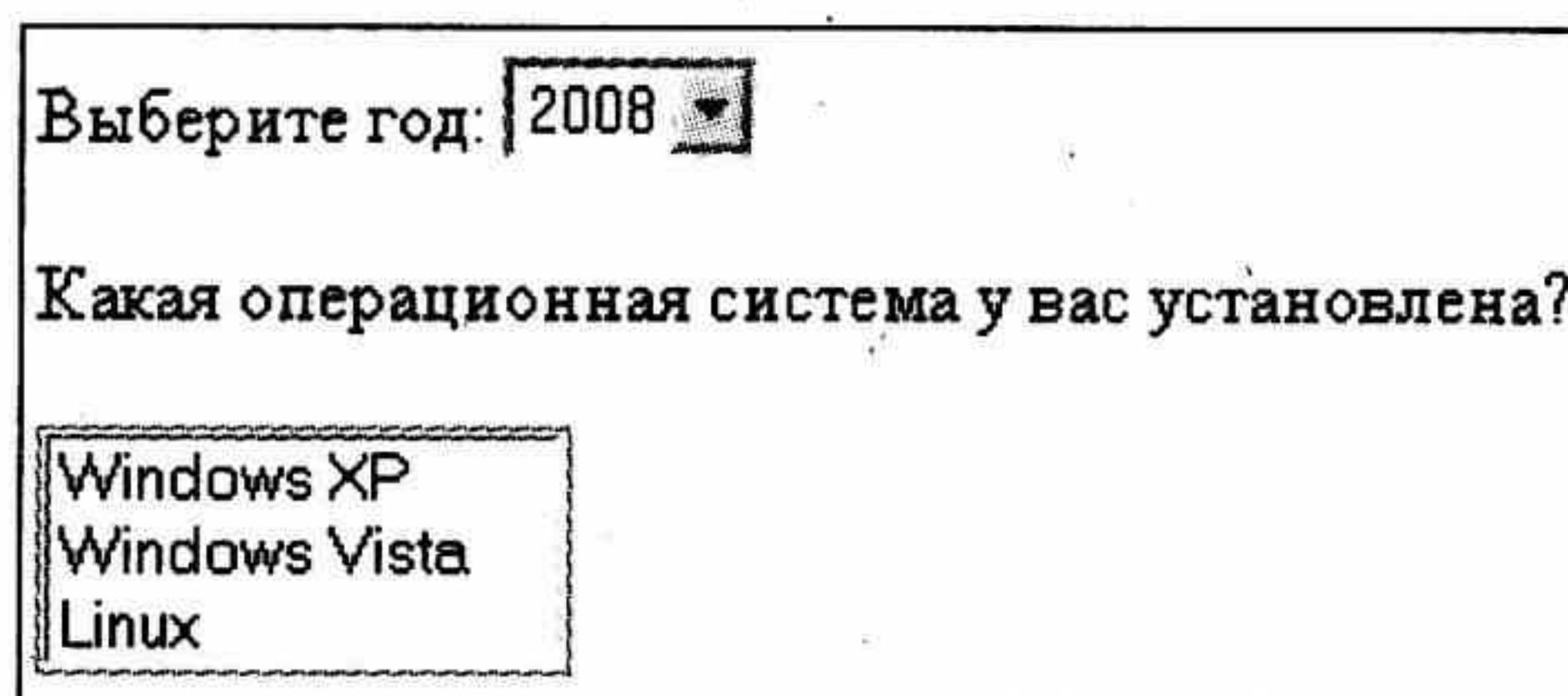


Рис. 6.3. Два списка выбора

Первый список не представляет ничего интересного — это обычный выпадающий список. Форма передает нашему сценарию параметр `year` со значением, совпадающим с одним из значений `value` из тега `OPTION`. Например, если пользователь выберет **2005**, то будет передан параметр:

```
year=2005
```

Обработать список можно так:

```

if (isset($_REQUEST['year']))
echo $_REQUEST['year'];

```

У нас значения параметров довольно простые и не содержат пробелов, поэтому мы могли бы вообще упростить объявление списка до следующего вида:

```

<select name=year size=1>
<option>2014</option>
<option>2013</option>
<option>2012</option>
<option>2011</option>
<option>2010</option>
<option>2009</option>
<option>2008</option>
<option>2007</option>
<option>2006</option>
<option>2005</option>
</select>

```


В этом случае в качестве значения `year` будет передан текст между тегами `OPTION` и `/OPTION`.

А теперь представим, что нам нужно сформировать список, возвращающий номер месяца, но пользователю не очень удобно работать с числами, ему проще выбрать название месяца. Поэтому параметр `value` тега `OPTION` нам очень пригодится:

```
<select name=month size=1>
<option value=1>Январь</option>
<option value=2>Февраль</option>
<option value=3>Март</option>
...
```

Во втором списке (см. рис. 6.3), как вы уже заметили, значения параметра `value` тоже отличаются от текста, заключенного в тег `OPTION`. Второй список разрешает множественный выбор благодаря параметру `multiple`. Обратите внимание, что после имени второго списка идут две квадратные скобки. Это означает, что мы передаем массив параметров, т. е. несколько значений, которые может выбрать пользователь.

Вот пример обработки такого списка:

```
$os = $_REQUEST['os'];

// проверяем, является ли $os массивом
if (is_array($os)) {
    foreach ($os as $key => $value) {
        echo "$value<br>";
    }
}
```

Если пользователь выбрал Windows XP и Linux, то сценарий выведет:

```
xp
linux
```

6.2.4. Форма для передачи файлов

С помощью тега `INPUT` можно создать поле загрузки файлов на сервер, но об этом мы поговорим в *главе 29*.

6.2.5. Кнопки

После того как пользователь установит все параметры, он нажмет кнопку отправки параметров — уже знакомую нам кнопку **Submit**. Существуют два варианта кнопки: кнопка отправки формы и кнопка сброса (устанавливает параметры по умолчанию):

```
<input type=submit name=send value=OK>
<input type=reset value=Сброс>
```


Иногда, чтобы реализовать несколько действий, нам нужны несколько кнопок — например, кнопки **Добавить** и **Удалить**. А как определить, какую кнопку нажал пользователь? Для этого нужно не забыть определить параметр `name` кнопок:

```
<input type=submit name=add value=Добавить>
<input type=submit name=delete value=Удалить>
```

В сценарии кнопки можно обработать так:

```
// определяем, какую кнопку нажал пользователь
if (isset($_REQUEST['add'])) {
    // пользователь нажал кнопку "Добавить"
    ...
}
elseif (isset($_REQUEST['delete'])) {
    // пользователь нажал кнопку "Удалить"
    ...
}
else die('Пользователь не нажимал кнопки формы, а пытается запустить сценарий
напрямую');
```

Наш сценарий не только сообщает, какую кнопку нажал пользователь, но и определяет, пытается ли пользователь запустить сценарий напрямую (возможно, он пытается взломать сайт?). Конечно, в случае использования метода GET такую проверку легко обойти, добавив к строке запроса параметр `add=Добавить`. Но в случае метода POST проделать такое будет сложнее. Правда, в этом случае нужно искать параметр не в массиве `$_REQUEST`, а в массиве `$_GET`.

6.3. Проверка параметров формы

Вы никогда не должны доверять параметрам, полученным из формы. Виной тому человеческий фактор. Кто-то может по ошибке ввести неправильные данные (например, ошибиться при вводе номера кредитной карты), а кто-то вводит неправильные данные умышленно с целью взлома сайта. О защите сайта мы поговорим в *главе 47*, а сейчас рассмотрим основные принципы проверки параметров формы:

1. Обязательно проверяйте существование параметров формы (с помощью функции `isset()`) перед обращением к ним, чтобы не попасть в ситуацию запуска пользователем сценария без параметров.
2. Если проверять значения всех параметров вам не хочется (их может быть столь много, что такая проверка отрицательно скажется на читабельности сценария), поможет инициализация переменных, т. е. установка параметров по умолчанию. Этот способ представляется более уместным — даже если форма не передаст параметр, у вас будут параметры по умолчанию, и вы сможете продолжить работу с этими параметрами вместо вывода сообщения об ошибке.

Рассмотрим второй способ. Предположим, что мы ожидаем два параметра: год рождения (`$year`) и имя пользователя (`$username`). Установим значения по умолчанию, а затем получим значения этих параметров:

```
$year = 2008;  
$username='user';  
$year = $_REQUEST['year'];  
$username = $_REQUEST['username'];
```

Существует еще одна стратегия проверки — проверка типа значения. Если вы ожидаете строку, то нужно убедиться, что переданный параметр — строка, а не что-то иное (число, например). Если вы, наоборот, ждете число, тогда нужно убедиться, что пользователь ввел число, а не строку.

Для проверки типов вы можете использовать следующие функции:

- ❑ `is_numeric()` — возвращает `true`, если переданный ей аргумент — число;
- ❑ `is_string()` — возвращает `true`, если переданный ей аргумент — строка;
- ❑ `is_array()` — возвращает `true`, если переданный ей аргумент — массив.

Иногда нужно проверить, соответствует ли введенная строка определенному формату — например: пользователь ввел e-mail или произвольную строку? Далее мы займемся проверкой корректности e-mail и номера кредитной карточки.

6.3.1. Проверка корректности e-mail

Напишем функцию `is_email()`, которая будет проверять, является ли введенный пользователем текст электронным адресом. Если формат e-mail правильный, то функция возвращает `true`. В противном случае — `false`. Обратите внимание, что функция не проверяет существование домена пользователя, и если пользователь введет адрес **user@domain.com**, то функция сообщит, что адрес правильный, поскольку он соответствует правилам написания электронных адресов. Однако функция не гарантирует, что домен **domain.com** существует (листинг 6.3).

Листинг 6.3. Проверка корректности e-mail

```
function is_email($email) {  
    if (!preg_match('/^[A-Za-z0-9!#$%&\'*+-\./=?^_`{|}~]+@[A-Za-z0-9-]+\.[A-Za-z0-9-]+[A-Za-z]$/', $email)) {  
        return false;  
    } else {  
        return true;  
    }  
}
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *6-3.php*), представленный в каталоге *Glava_6* сопровождающего книгу электронного архива (см. приложение 4).

Для проверки формата адреса функция использует *регулярные выражения*. Подробно о них мы поговорим в *главе 33*, а пока вам нужно лишь знать, как эту функцию применять:

- ❑ добавьте описание функции в начало PHP-сценария или вынесите в отдельный PHP-файл и подключите с помощью инструкции `require` (см. *главу 22*);
- ❑ вызовите функцию так:

```
$email = $_REQUEST['email'];  
  
if (is_email($email)) echo "E-mail правильный";  
else echo "Введите правильный e-mail!";
```

6.3.2. Проверка правильности номера кредитной карты

Кредитные (банковские) карты все больше входят в обиход, и даже у нас теперь не боятся оплачивать покупки в интернет-магазинах кредиткой. Обработкой транзакции, т. е. снятием определенной суммы со счета клиента, занимается отдельная платежная система (с которой вы должны заключить договор). Выбор системы зависит от многих факторов, но сейчас речь не об этом. До выполнения запроса на транзакцию вы должны хотя бы убедиться, что клиент ввел номер кредитной карточки, а не свой домашний адрес. Выполнить проверку можно так же с помощью регулярных выражений, зная формат номера кредитной карточки, — вам он понадобится для создания регулярного выражения.

Листинг 6.4. Функция проверки корректности номера кредитной карточки

```
function is_cc_num($cc_num) {  
  
    $card_type = "";  
  
    // массив регулярных выражений, который используется для  
    // определения типа карты  
    $cards = array(  
        '/^4\d{12}(\d\d){0,1}$/' => 'visa',  
        '/^5[12345]\d{14}$/' => 'mastercard'  
    );  
    // определение типа карты (visa или mastercard)  
    foreach ($cards as $reg_ex => $type) {  
        if (preg_match($reg_ex, $cc_num)) {  
            $card_type = $type;  
            break;  
        }  
    }  
  
    // мы не можем определить тип карты, возвращаем false  
    if (!$card_type) {
```



```
return false;
}

// проверка контрольной суммы карты
$code = strrev($cc_num);
$src = 0;

for ($i = 0; $i < strlen($code); $i++) {
    $current_num = intval($code[$i]);
    if ($i & 1) {
        $current_num *= 2;
    }
    $src += $current_num % 10;
    if ($current_num > 9) {
        $src += 1;
    }
}

if ($src % 10 == 0) {
    return $card_type;
} else {
    return false;
}
}
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *6-4.php*), представленный в каталоге *Glava_6* сопровождающего книгу электронного архива (см. приложение 4).

В начале функции мы задаем массив регулярных выражений, позволяющих определить тип кредитной карточки: VISA или MasterCard (если вы знаете форматы номеров карточек других систем, то можете расширить эту функцию после прочтения главы 33). Затем мы, используя этот массив, устанавливаем тип карточки. Если тип карточки установить не удалось, мы возвращаем *false*.

Далее мы проверяем контрольную сумму карточки. Процедура проверки контрольной суммы карточки стандартна, поэтому в большинстве случаев вы сможете использовать ее для проверки корректности номера карточек других систем. Если контрольная сумма не делится на 10, значит, мы возвращаем *false*. Если карточка прошла все проверки, значит, мы возвращаем тип карточки ("visa" или "master-card").

Использовать функцию очень просто:

```
echo is_cc_num("4111 1111 1111 1111");
```


6.3.3. Удаление лишних пробелов

Чего только не вводят пользователи в поля формы. Довольно часто они почему-то вводят пробелы до и после строки. Лишние пробельные символы можно удалить функцией `trim()` — например, так:

```
$username = trim($username);
```

Об остальных строковых функциях мы поговорим в *главе 13*.

6.4. Защита от спама с помощью формы CAPTCHA

Любой проект, предусматривающий ввод данных пользователем (будь то форма регистрации на сайте, гостевая книга или просто форма контактов или отзывов о сайте), требует защиты от спама. Рано или поздно спамеры найдут ваш сайт, и каждый день вы будете получать спам: предложения куда-то что-то инвестировать, что-то увеличить на 3 дюйма или поздравления с наследством в пару миллионов долларов. Поначалу это просто раздражает, а потом даже мешает сайту — ведь если не удалять спам из базы данных, вы в конечном итоге превысите квоту на ее максимальный размер, и работа вашего сайта будет приостановлена хостером.

Бороться со спамерами можно путем добавления формы CAPTCHA. Такая форма представляет изображение с немного искаженными символами (буквами и цифрами) и поле, позволяющее ввести опознанные вами символы изображения. Человек, как правило, может узнать знакомые символы и ввести правильный ответ. Нужно отметить, что некоторые формы так искажают символы, что понять, какой символ изображен, бывает сложно даже человеку. Но именно такие формы и являются максимально эффективными. Ведь спамеры тоже не сидят без дела, и их программы уже справляются с менее эффективными формами.

В этой главе мы продемонстрируем одну из самых эффективных CAPTCHA-форм — сервис Google reCAPTCHA. На рис. 6.4 показано, как будет выглядеть наша форма. Кнопка **Заказать** к форме не относится — иллюстрация представляет

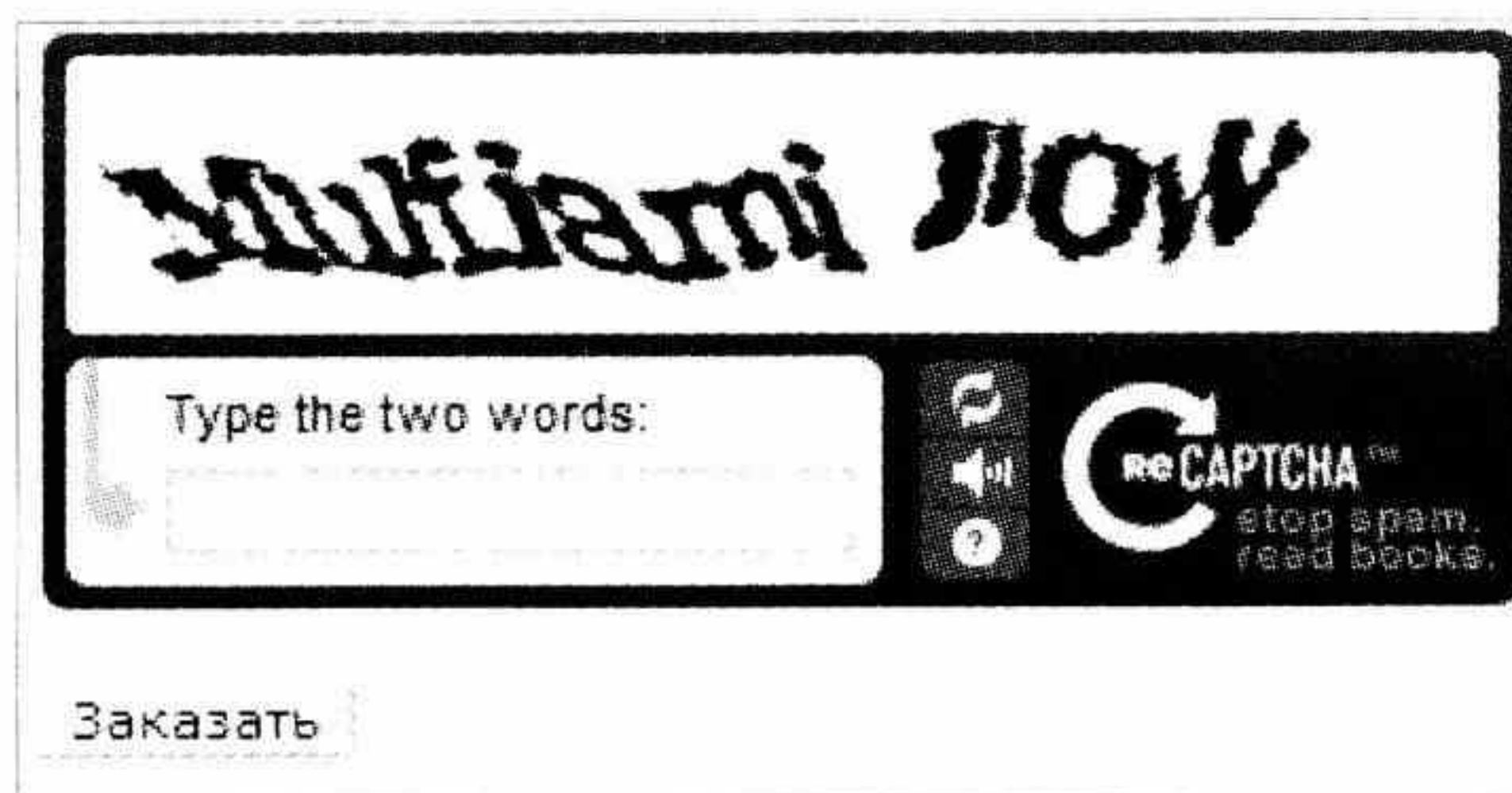


Рис. 6.4. Форма от проекта reCAPTCHA

собой фрагмент скриншота рабочего окна одного из разработанных мною интернет-магазинов. А кнопка попала на иллюстрацию, чтобы вам было понятно, что форма CAPTCHA представляет собой только картинку и поле для ввода, а кнопка отправки данных (кнопка **Submit**) в каждом случае своя. То есть вам практически не нужно изменять свои формы отправки данных — только добавить в нужном месте (перед кнопкой **Submit**) вызов формы CAPTCHA.

Зарегистрируйтесь на сайте <http://www.google.com/recaptcha> (регистрация бесплатна). Вы получите архив с файлом `recaptchalib.php` и два ключа: `public key` (публичный ключ) и `private key` (частный ключ). Эти два ключа необходимы для правильной работы формы reCAPTCHA. Для каждого сайта, на который вы собираетесь поместить форму reCAPTCHA, нужно создать отдельные ключи. На рис. 6.5 показана моя учетная запись в сервисе reCAPTCHA. Свои ключи по понятным причинам я скрыл.

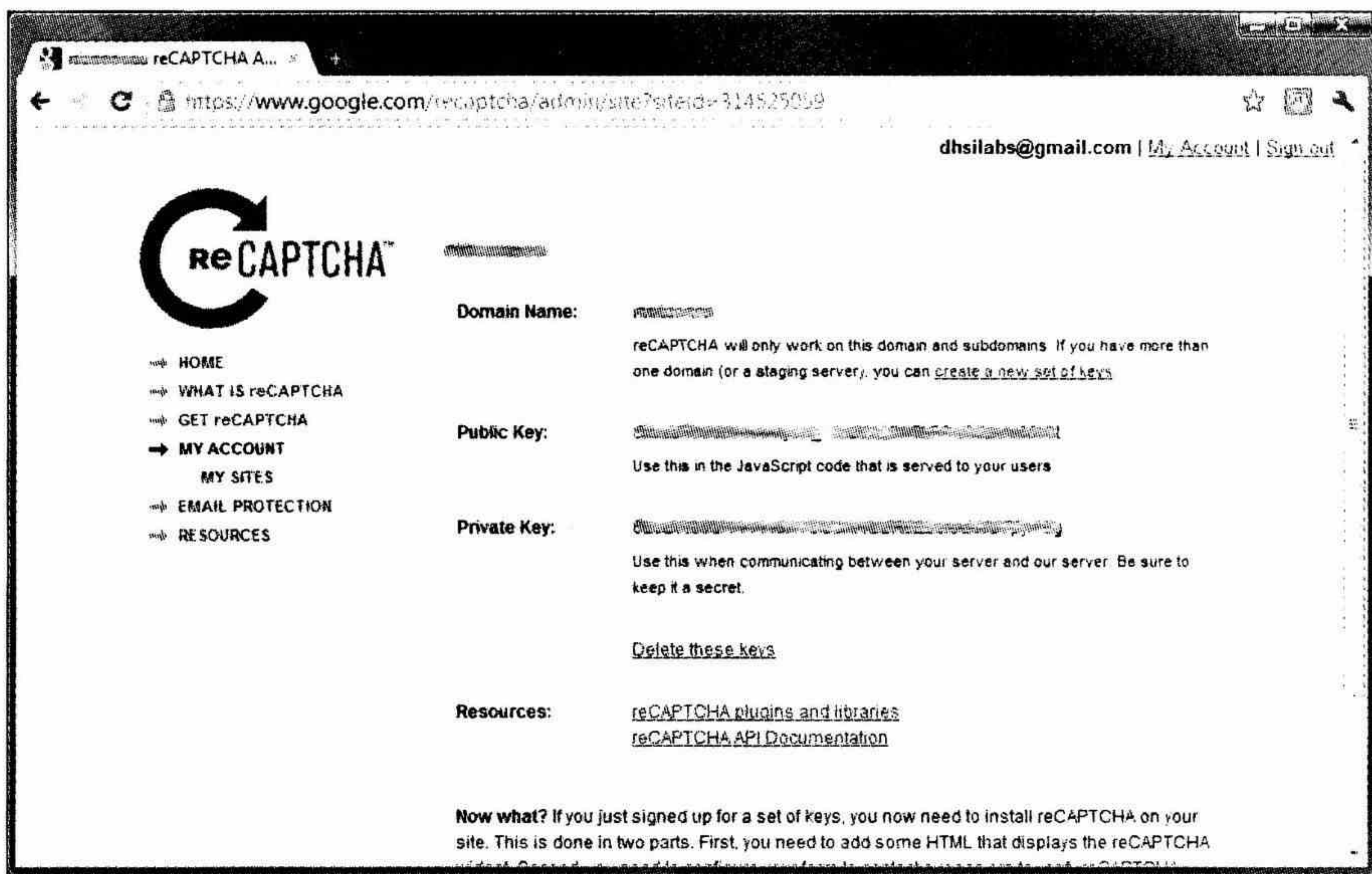


Рис. 6.5. Учетная запись пользователя (проект reCAPTCHA)

Теперь модифицируем код PHP-файла, выводящего форму отправки данных (форма добавления нового сообщения, комментария, форма регистрации на сайте — тип формы зависит от вашего проекта):

```
// подключаем сценарий recaptchalib.php
require_once('recaptchalib.php');
// указываем ваш public key
$publickey = "public key";
```


Затем в нужном месте (перед выводом кнопки **Submit**) выводим форму reCAPTCHA:

```
recaptcha_get_html($publickey);  
echo "<input type=submit value=OK name=ok>";
```

Теперь следует отредактировать код сценария, который принимает ввод пользователя из формы и добавляет его в базу данных. Задача этого кода — проверить правильность введенной пользователем CAPTCHA-фразы:

```
require_once('recaptchalib.php');  
$privatekey = "private key";  
$resp = recaptcha_check_answer ($privatekey,  
                                $_SERVER["REMOTE_ADDR"],  
                                $_POST["recaptcha_challenge_field"],  
                                $_POST["recaptcha_response_field"]);  
  
if (!$resp->is_valid) {  
    // действие сценария, если введенная фраза неправильна  
    // я просто завершаю сценарий  
    die ("Введенная фраза неправильная!");  
}  
// далее следует обычный код обработки полученных данных  
// например, добавление данных в БД или отправка данных по e-mail
```

Вот и все — как видите, совсем нет ничего сложного.

6.5. Форма поиска в Яндексе

Отвлечемся от проектов Google, тем более что есть отечественная поисковая машина Яндекс (<http://www.yandex.ru/>), которая не менее популярна, чем Google. Поэтому сейчас мы добавим на наш сайт форму поиска Яндекса. Сделать это довольно просто — достаточно добавить следующий HTML-код в шаблон вашего сайта или в PHP-код:

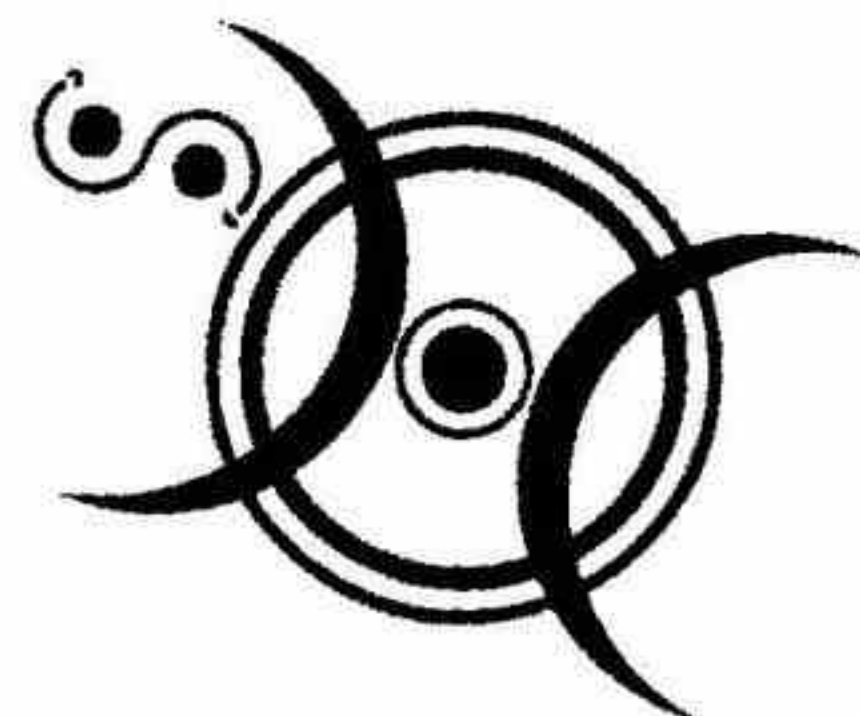
```
// закрываем PHP-код, для вывода HTML-кода:  
?>  
<form name="web" method="get"  
action="http://www.yandex.ru/yandsearch"> <input type="text" name="text"  
size="18" value="" maxlength="160"> <input type="hidden" name="serverurl"  
value="ваш URL"> <input type="hidden" name="server_name" value="ваш URL">  
<input type="hidden" name="referrer1" value="ваш URL"> <input type="hidden"  
name="referrer2" value="ваш URL ">  
<input type="submit" value="Поиск"> </form>  
<?php  
// PHP-код продолжается
```

Результаты поиска изображены на рис. 6.6.



Рис. 6.6. Результаты поиска с помощью Яндекса

ГЛАВА 7



Запоминание параметров с помощью Cookies и сессий

7.1. Что такое Cookies и как с ними работать?

Cookies — это данные небольшого объема, хранящиеся на стороне клиента, т. е. в браузере. Cookies можно использовать, например, для хранения выбранной пользователем цветовой гаммы сайта, элементов корзины заказов.

Поскольку Cookies хранятся на стороне клиента, они могут быть легко подделаны. Следовательно, не нужно помещать в Cookies важную (конфиденциальную) информацию.

Как у всего на свете, у Cookies есть преимущества и недостатки. Начнем с преимуществ.

- ❑ Cookies могут храниться годами, пока пользователь их не удалит, не переустановит операционную систему или пока не истечет срок жизни Cookies, установленный программистом, т. е. вами.
- ❑ Поскольку Cookies хранятся на стороне клиента, они не создают дополнительную нагрузку на сервер. Вы только представьте, что у нас есть большой сервер, у которого 100 000 пользователей (имеются в виду не зарегистрированные пользователи, а только те, которые посетили наш сайт и не поленились выбрать тему дизайна). Нам нужно хранить в Cookies всего одну настройку сайта — цветовую тему сайта. Название темы — это строка длиной до 20 символов. А теперь посчитаем: $20 \times 100\,000 = 2\,000\,000$ байтов = 1,95 Мбайт. Вроде бы немного для большого сервера, но, как правило, серьезные сайты не ограничиваются хранением только одной переменной в Cookies, а могут сохранять довольно внушительные объемы данных.

А вот и недостатки Cookies.

- ❑ Браузер может не принимать Cookies (некоторые пользователи отключают их прием), и вы даже об этом не узнаете. Максимум, что можно сделать, — это проверить существование того или иного Cookie с помощью функции `isset()`.
- ❑ Размер хранимой в Cookies информации не должен превышать 512 Кбайт (для каждой переменной), поэтому вы не можете хранить в Cookies большие объемы информации.

- ❑ Cookies привязаны к конкретному браузеру. Если пользователь зайдет на сайт с другого браузера или вообще с другого компьютера, вы не сможете использовать ранее установленные пользователем параметры.

Для установки Cookies служит функция `setcookie()`, которой нужно передать следующие параметры:

- ❑ имя — имя Cookie, потом вы будете обращаться к сохраненным данным по этому имени;
- ❑ значение — значение переменной Cookie;
- ❑ срок жизни — задает время жизни Cookie в секундах. Если этот параметр не указан, Cookie будет существовать до закрытия окна браузера;
- ❑ путь — путь Cookie. Если вы хотите, чтобы устанавливаемая переменная относилась только к страницам из каталога `/site/`, то установите `/site/` в качестве значения этой переменной;
- ❑ домен — домен Cookie. Предположим, у вас есть два сайта: **orders.example.ru** и **www.example.ru**, и вы хотите, чтобы устанавливаемая Cookie была доступна на обоих сайтах. Тогда в качестве значения этого параметра нужно указать `.example.ru`;
- ❑ защита — если этот параметр равен `true`, то Cookie должна передаваться по защищенному соединению (SSL).

Все параметры, кроме первого, необязательны. Рассмотрим пример установки Cookie:

```
setcookie("username", "denis", time()+36000);
```

В данном случае мы устанавливаем Cookie с именем `username`, значением `denis` и временем жизни — 10 часов (36 000 секунд).

Функция `setcookie` устанавливает HTTP-заголовки, поэтому до вызова этой функции ваш сценарий не должен производить никакого вывода в браузер. Если вы не можете установить Cookies, тому есть две причины:

- ❑ браузер их не поддерживает (что маловероятно — ведь ваш браузер их точно поддерживает, и они включены, раз вы надумали их использовать);
- ❑ сценарий производит вывод до вызова функции `setcookie()`, поэтому функцию `setcookie()` нужно вызывать в начале сценария.

Сосредоточимся на второй причине. Вы можете возразить: мол, какой вывод, если в сценарии вообще нет ни одного оператора `echo`?! Обязательно убедитесь, чтобы до и после тегов `<?php` и `?>` не было пробелов и других пробельных символов. Пробельные символы интерпретируются как HTML-код, и PHP выводит их перед выполнением кода сценария. Таким образом, получается, что перед выполнением `setcookie()` уже был произведен вывод.

После установки Cookie к ней можно обратиться через массив `$_COOKIE`:

```
echo $_COOKIE['username'];
```


Рассмотрим простой сценарий установки Cookie:

```
<?php

// проверяем существование Cookie
if(!isset($_COOKIE['username'])) {

// Cookie не установлена, устанавливаем ее
setcookie("username", "denis", time()+36000);
}
else echo $_COOKIE['username'];

?>
```

Наш сценарий предельно прост. Он проверяет установку Cookie. Если да, то сценарий выводит ее, а если нет — устанавливает. При первом запуске сценария вы ничего не увидите, а вот при втором запуске вы увидите значение установленной переменной.

7.2. Механизм сессий

7.2.1. Сессии и Cookies: преимущества и недостатки

Cookies — далеко не идеальный механизм, но PHP предлагает другой механизм — механизм сессий. Сессии, в отличие от Cookies, могут использоваться, даже если пользователь отключил браузер. К тому же, вы можете хранить столько данных, сколько вам нужно. Но самое главное отличие сессий от Cookies заключается в том, что информация, сохраненная в сессиях, хранится на сервере, а не на стороне клиента! Следовательно, вы можете использовать сессии для хранения важной, хотя и временной, информации. Пользователи не могут просматривать и редактировать данные сессии.

Но у сессии есть свои особенности. Их нельзя назвать недостатками, ведь без них не было бы у сессий их преимуществ. Одна из особенностей заключается в том, что сессии привязаны к серверу. Если у вас есть два сервера, и вы хотите использовать общую сессию, то вам нужно разработать собственный механизм хранения сессий, что, к сожалению, выходит за рамки этой книги.

Теоретически возможен перехват идентификатора сессии и, как результат, самой сессии. После этого один пользователь сможет работать от имени другого пользователя. Защититься от перехвата (частично) позволяет установка тайм-аута сессии. Если тайм-аут не установлен, тогда сессия будет действительна до закрытия окна браузера.

7.2.2. Для чего нужны сессии?

Рассмотрим вопрос о правильной аутентификации пользователя. Аутентификация нужна в любом серьезном проекте, будь то вход на форум или в панель администрирования сайта. Важно не только проверить имя пользователя и пароль, но проде-

лать это правильно и после этого обеспечить дальнейшую корректную работу сценария.

Допустим, у нас есть сценарий `login.php` (листинг 7.1), выводящий форму регистрации и проверяющий правильность имени пользователя и пароля, переданных пользователем с помощью формы.

Листинг 7.1. Сценарий аутентификации пользователя — `login.php`

```
<?php

if (!isset($_POST['login']))
{
    // выводим форму входа на сайт
    echo join('', file('login-form.html'));
}
else
{
    // проверяем имя пользователя и пароль
    include "config.php";
    if (!mysql_connect($HOST, $USER, $PASS)) die('Error');
    mysql_select_db($DB);

    $r = mysql_query("select * from users where login=\"{$login}\"");
    $f = mysql_fetch_array($r);

    if ($f['password'] === md5($pass))
    {
        // выводим меню администратора
        echo join('', file('admin-menu.html'));
    }
    else die('Неправильное имя пользователя или пароль');
}

?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл `7-1.php`), представленный в каталоге *Glava_7* сопровождающего книгу электронного архива (см. приложение 4).

Основную задачу наш сценарий выполнил — проверил имя пользователя и пароль и вывел меню администратора (HTML-код меню вынесен из сценария и помещен в отдельный HTML-файл). Наверняка наш код будет содержать ссылки на другие PHP-сценарии — например, на сценарии управления пользователями, статьями и т. п. Возникает вопрос — как эти дополнительные сценарии "узнают", что сценарий `login.php` проверил имя пользователя и пароль и разрешил доступ? Правильно. Никак. И что же, опять выводить форму для ввода имени пользователя и пароля?

Первое, что приходит в голову, — это использование Cookies. В случае корректной аутентификации мы устанавливаем Cookie-переменную `$AUTH`. Если такая переменная установлена, значит, пользователь аутентифицирован.

Это достаточно простое решение, но небезопасное. Ведь злоумышленник может написать сценарий, который установит переменную с таким же именем. Тогда он сможет запустить любой сценарий, входящий в состав панели администрирования, даже без ввода имени пользователя и пароля.

Да и вообще применение Cookies не очень удобно — что делать, если Cookies отключены? Поэтому такое решение нельзя назвать универсальным. Но в PHP есть механизм сессий более гибкий, чем Cookies. К тому же, если браузер не поддерживает Cookies (или они отключены), сценарии на основе сессий будут преспокойно работать.

7.2.3. Как работает механизм сессий?

В механизме сессий для идентификации пользователя имеется идентификатор сессии — `session identifier (SID)`. С его помощью PHP определяет, какой пользователь запустил сценарий. Идентификатор Cookies хранится в Cookies браузера, а все переменные сессии — на сервере. Получается, что SID — это номер хранилища (временного файла), в котором PHP хранит переменные сессии. Чувствуете разницу? Если в варианте с Cookies мы можем узнать имя переменной и установить ее в собственном браузере (для этого достаточно написать сценарий, вызывающий функцию `setCookie()`), то в случае с сессиями такой фокус не пройдет. Ведь переменные сессии хранятся на сервере, а не на нашем компьютере. Конечно, возможность перехвата сессии все же существует — например, когда пользователь по неосторожности оставит ссылку со своим SID где-то на форуме, но это равносильно тому, что он оставил там свои имя пользователя и пароль. Поэтому нужно соблюдать элементарную безопасность, господа!

Как уже было сказано, SID — это имя временного файла. Как и другие временные файлы, наши хранилища находятся на сервере в каталоге `/tmp`.

Инициализировать сессию можно с помощью функции `session_start()`:

```
session_start();
```

Для регистрации переменной в сессии предназначена функция `session_register()`:

```
bool session_register (mixed name [, mixed ...])
```

Эта функция принимает переменное число аргументов, т. е. позволяет за один вызов зарегистрировать несколько переменных в нашей сессии. Для аутентификации пользователя нам потребуется зарегистрировать три переменные: `$auth`, `$username`, `$password` (для чего они будут использоваться — уточним позже).

```
session_register("a", "username", "password");
```

Имена переменных следует указывать без знака доллара. В сессии можно несколько раз регистрировать одну и ту же переменную, не боясь, что ее значение обнулится. После регистрации с переменной сессии можно работать как с обычной пе-

ременной, а ее последнее значение при завершении работы сценария будет записано в сессию.

Функция `session_is_registered()` позволяет проверить, зарегистрирована ли в сессии некоторая переменная:

```
bool session_is_registered(string name)
```

Если вы хотите удалить из сессии переменную, тогда укажите функцию `session_unregister()`:

```
session_unregister(string $name)
```

Удалить все переменные из сессии можно, вызвав функцию:

```
session_unset();
```

Вы можете присвоить собственное имя сессии с помощью функции `session_name()`, которую нужно вызвать до инициализации сессии, т. е. до обращения к функции `session_start()`:

```
session_name('MySession');  
session_start();
```

Если имя сессии не задано (т. е. вы вообще не вызывали эту функцию), то по умолчанию в PHP будет задано имя `PHPSESSID`.

Не нужно путать `SID` и имя сессии. Имя сессии — это название группы сессий, к которой принадлежит тот или иной `SID`. Кстати, `SID` можно узнать с помощью функции `session_id()`:

```
string session_id([string sid])
```

Если параметр `sid` не задан, то функция возвращает `SID`, в противном случае она устанавливает заданное значение в качестве `SID`. Свой `SID` указывать не рекомендуется, поскольку вряд ли вы придумаете `SID` сложнее, чем это сделает за вас PHP.

Обратиться к переменным, сохраненным в сессиях, можно через массив `$_SESSION`:

```
echo $_SESSION['a'];
```

7.2.4. Обход Cookies

Некоторые пользователи из соображений безопасности (правда, это больше иллюзия безопасности) отключают Cookies. Тогда к ссылкам желательно дописывать константу `SID`, содержащую идентификатор сессии:

```
<?php  
session_start();  
session_register("auth", "username", "password");  
  
$login = "denis";  
  
?>  
<a href=admin.php?<?=SID?>>Ссылка</a>;
```


Но если вы забудете это сделать, ничего страшного — PHP выполнит это за вас. Другими словами, если Cookies выключены, все будет работать, причем вам для этого ничего не придется делать.

7.3. Массивы и Cookies

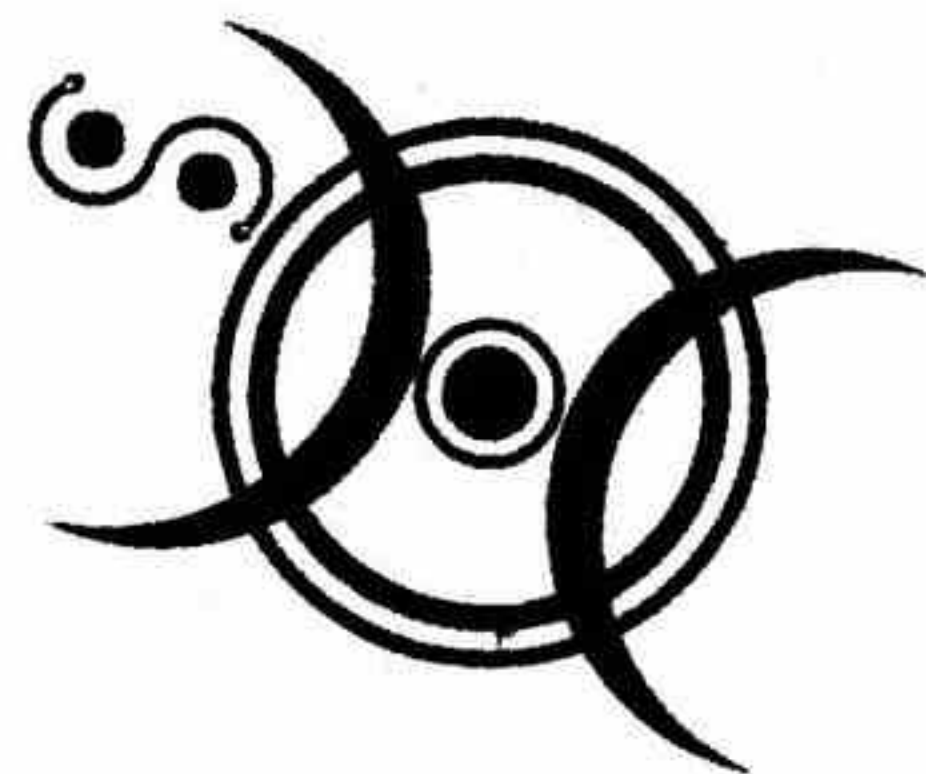
Понимаю, что массивы мы еще не рассматривали, но пока мы не забыли, что такое Cookies и сессии, нам нужно узнать, как поместить массивы в Cookies. Вы не можете поместить ни в Cookies, ни в сессии массив в "чистом" виде. Перед помещением массива в Cookies его нужно сериализовать — превратить в строку. Это можно сделать с помощью функции `serialize()`:

```
// $A — массив, $A_str — строка  
$A_str = serialize($A);
```

Чтобы превратить строку `$A_str` обратно в массив, следует воспользоваться функцией `unserialize()`:

```
$A = unserialize($A_str);
```


ГЛАВА 8



Отдельное слово о директиве *register_globals*

8.1. Почему опасно использовать *register_globals*?

В файле конфигурации `php.ini` имеется директива `register_globals`. Если она включена (`register_globals = On`), то переменные, переданные с помощью методов `GET` и `POST`, а также ряд других, будут зарегистрированы автоматически как глобальные. Например, у нас есть сценарий `index.php`. Если вы вызовете его так:

```
index.php?page=main
```

то автоматически будет создана переменная `$page` со значением `main`, которая станет доступна в любой части сценария `index.php`. Это очень удобно, поскольку не нужно больше анализировать массивы `_GET` и `_POST` на предмет переданных данных. Если же директива `register_globals` выключена (`register_globals = Off`), обратиться к переменной `$page` можно будет с помощью конструкции `$_GET['page']`.

Кроме массивов `$_GET` и `$_POST` вы можете использовать следующие массивы (подробно о суперглобальных массивах мы поговорим позже):

- ❑ `$_REQUEST` — если вы не знаете, каким методом был передан параметр, то можно попытаться найти его в массиве `$_REQUEST`, который представляет собой объединение массивов `$_GET`, `$_POST` и `$_COOKIE`;
- ❑ `$_COOKIE` — Cookies, установленные сценарием;
- ❑ `$_FILES` — информация о загружаемых файлах хранится в этом массиве.

Все мы не один раз слышали, что использовать директиву `register_globals` опасно. Именно поэтому директива `register_globals` по умолчанию всегда отключена. Когда-то даже шла речь о том, чтобы полностью избавиться от этой директивы (в версии 6, которая так и не вышла), а поведение PHP при работе с глобальными переменными сделать таким, как при выключенной директиве. То есть, выключить ее раз и навсегда. Однако до сих пор, даже в самой новой версии PHP, 5.5, эта директива в файле `php.ini` присутствует, но она выключена (значение `off`). Можно было бы, конечно, написать, что есть такая директива и что ни в коем случае не

включайте ее. Однако важно, чтобы читатель понимал, почему ее нельзя включать. Для этого и предназначена целая глава.

Давайте еще раз рассмотрим, как работает `register_globals`. Создайте следующий сценарий:

```
<?php

    echo $a . "<br>";
    echo $GLOBALS['a'] . "<br>";
    echo $_GET['a'] . "<br>";

?>
```

Запустите этот сценарий при включенной директиве `register_globals` и передайте переменной `$a` значение `'a'`:

`http://localhost/index.php?a=a`

Сценарий выведет три буквы `"a"`:

```
a
a
a
```

Получается, что когда директива `register_globals` включена, то создаются три копии переменной: локальная переменная, к которой можно обратиться просто — `$a`, копия в массиве `$GLOBALS` и копия в массиве `$_GET`, поскольку переменная была передана методом `GET`.

Если директиву `register_globals` выключить, то будет выведена только одна буква `"a"` — из массива `$_GET`. Поскольку локальная переменная `$a` создана не будет, не окажется и ссылки на нее в массиве `$GLOBALS`.

Узнав, как работает `register_globals`, можно разобраться и в ее потенциальной опасности. Рассмотрим небольшой сценарий проверки имени пользователя и пароля (листинг 8.1).

Листинг 8.1. Сценарий с уязвимостью, версия для `register_globals`

```
...
// получаем пароль пользователя из нашей БД
$password = GetPasswordFor($login);

// проверяем пароль пользователя
if ( md5($password) === $password ) $auth = TRUE;

if ($auth) {
    echo "Аутентификация прошла успешно!";
    ...
}
```


Наш сценарий работает так:

1. Пользователь передает (понятно, с помощью красивой формы на главной странице) имя пользователя (переменная `$login`) и пароль (`$password`).
2. Сценарий получает зашифрованный пароль из базы данных с помощью функции `GetPasswordFor()`.
3. Полученный пароль шифруется с помощью функции `md5()` и сравнивается с имеющимся у нас паролем.
4. Если пароли совпадают, переменная `$auth` устанавливается в `true`.
5. Если переменная `$auth` истинна, пользователь видит надпись **Аутентификация прошла успешно!** и получает доступ к своей панели управления.

Такой сценарий содержит огромную дыру в системе безопасности. Смотрите — если имя пользователя и пароль передаются методом `GET`, то адрес (URL) в строке браузера будет выглядеть так (предположим, что наш сценарий называется `login.php`):

`http://localhost/login.php?login=denis&password=123456`

Все, что нужно злоумышленнику, — это добавить в конец URL определение переменной `$auth`:

`http://localhost/login.php?login=denis&password=123456&auth=1`

Даже если переменные `login` и `password` передаются с помощью метода `POST`, все равно данный способ сработает, ведь `register_globals` одинаково работает как с методом `GET`, так и с методом `POST`. Вообще, достаточно просто указать адрес **`http://localhost/login.php?auth=1`**, и наш метод "обхода" аутентификации сработает — переменная `$auth` будет определена со значением `1`, что соответствует логическому значению `true`.

В результате наш сценарий сработает так:

1. Он попытается вычислить пароль пользователя.
2. Затем проверит пароли.
3. Отобразит надпись **Аутентификация прошла успешно!**

Да, сценарий не устанавливал переменную `$auth` в `true`, за него это сделала директива `register_globals`.

Если бы директива `register_globals` была выключенной, переменную `auth` нельзя было бы передать указанным способом. Следовательно, переменная `$auth` приняла бы значение `true` только при вводе правильных имени пользователя и пароля. Конечно, в этом случае не были бы переданы и переменные `$login` и `$password`, поэтому сценарий пришлось бы модифицировать так (листинг 8.2):

Листинг 8.2. Сценарий с уязвимостью, версия без `register_globals`

...

```
// получаем пароль пользователя из нашей БД
$pswd = GetPasswordFor($_POST[$login]);
```



```
// проверяем пароль пользователя
if ( md5($_POST[$password]) === $pswd ) $auth = TRUE;

if ($auth) {
    echo "Аутентификация прошла успешно!";
    ...
}
```

Итак, я доказал вам, что `register_globals` опасна, однако у вас все же может возникнуть вопрос: "А откуда злоумышленник знает, что в моем сценарии есть переменная `$auth`, и для чего я ее использую?" Да, если это ваш собственный сценарий, и вы ни с кем им не делились, тогда узнать данный факт будет очень сложно. Но ведь не всегда мы используем только свои сценарии, мы же не пишем "с нуля" форум или чат, а часто берем уже готовые продукты. А они доступны всем. Злоумышленник может скачать ту же заготовку форума, проанализировать ее на предмет подобных уязвимостей и взломать ваш форум. Вот так-то...

Далеко не все хостеры отключают `register_globals`. И даже если ваш сценарий не будет ориентирован на `register_globals` (см. листинг 8.2), но эта директива окажется включена, все равно останется возможность обойти систему аутентификации, передав переменную `$auth`. Выходит, что все, у кого включена `register_globals`, в опасности? Нет, не все. Стоит немного изменить сценарий, исключив из него уязвимость, и злоумышленнику придется искать какой-то другой способ для взлома вашего сценария. Вот версия нашего сценария без уязвимости (листинг 8.3).

Листинг 8.3. Сценарий без уязвимости

```
...
// получаем пароль пользователя из нашей БД
$pswd = GetPasswordFor($_POST[$login]);

// проверяем пароль пользователя
if ( md5($_POST[$password]) === $pswd ) $auth = TRUE;
else $auth = FALSE;

if ($auth) {
    echo "Аутентификация прошла успешно!";
    ...
}
```

Мы добавили всего одну строку, которая в случае неправильного пароля устанавливает переменную `$auth` в `false`. Даже если злоумышленник и передаст сценарию переменную `$auth` со значением 1, наш сценарий в случае неправильного пароля установит для нее значение `false`.

Выходит, что опасность заключается не в самой директиве `register_globals`, а в некорректно написанных сценариях. Но поскольку не все умеют писать безопасные сценарии, с выключенной директивой `register_globals` все же безопаснее.

8.2. Если `register_globals` отключена

Ситуация с данной директивой двойственна. С одной стороны, сами разработчики PHP отмечают, что выключение этой директивы на 90 % уменьшает уязвимость PHP-сценариев. С другой стороны, до версии PHP 4.3 данная директива была по умолчанию включена, и тогда все писали сценарии, ориентируясь на это ее состояние. Поверьте, было написано очень много сценариев. Выходит, что все их нужно переписать? С точки зрения безопасности — да, но на практике мало кто этим будет заниматься.

Что же делать, если ваш хостер выключил эту директиву, и рассчитанные на нее сценарии перестали работать?

При выключенной директиве `register_globals` получить доступ к внешним переменным (которые пользователь не определил явно в сценарии) возможно через ассоциативные массивы `HTTP_***_VARS` (`HTTP_POST_VARS`, `HTTP_GET_VARS` и т. д.) или через суперглобальные массивы, которые использовать немного удобнее. Да и разработчики PHP сами рекомендуют отключить `register_globals` и работать с переменными только через суперглобальные массивы. Все нужные вам суперглобальные массивы, а также соответствующие им ассоциативные массивы, представлены в табл. 8.1.

Таблица 8.1. Суперглобальные и ассоциативные массивы

Суперглобальный массив	Ассоциативный массив	Описание
<code>\$_GET</code>	<code>\$HTTP_GET_VARS</code>	Переменные, переданные с помощью метода GET
<code>\$_POST</code>	<code>\$HTTP_POST_VARS</code>	Переменные, переданные с помощью метода POST
<code>\$_SERVER</code>	<code>\$HTTP_SERVER_VARS</code>	Переменные, установленные веб-сервером
<code>\$_COOKIE</code>	<code>\$HTTP_COOKIE_VARS</code>	Переменные, переданные через Cookies
<code>\$_REQUEST</code>		Переменные, переданные пользователем (GET, POST, COOKIE)
<code>\$_ENV</code>	<code>\$HTTP_ENV_VARS</code>	Содержит переменные окружения
<code>\$_SESSION</code>	<code>\$HTTP_SESSION_VARS</code>	Переменные, определенные в текущей сессии сценария
<code>\$_FILES</code>	<code>\$HTTP_POST_FILES</code>	Переменные, используемые для загрузки файла по HTTP
<code>\$GLOBALS</code>		Содержит ссылки на все переменные, которые в данный момент доступны в глобальной области сценария

Обратиться к той или иной переменной очень просто. Предположим, что мы ожидаем переменную `$variable`, которая должна быть передана с помощью метода

POST. Тогда получить доступ к значению этой переменной (если она была передана) можно так:

```
$_POST['variable']
```

Если вам не хочется постоянно писать `$_POST['variable']`, можно в начале сценария поместить инструкцию:

```
$variable = $_POST['variable'];
```

После этого с переменной `$variable` можно работать как обычно.

Аналогично в начале сценария можно поместить подобные инструкции для всех переменных, которые вы ожидаете.

Чтобы просмотреть, какие переменные находятся в том или ином массиве, используется следующая простая конструкция:

```
foreach (<массив> as $k=>$v)
    echo "<br>$k = $v";
```

Например (см. рис. 8.1):

```
foreach ($_GET as $k=>$v)
    echo "<br>$k = $v";
```

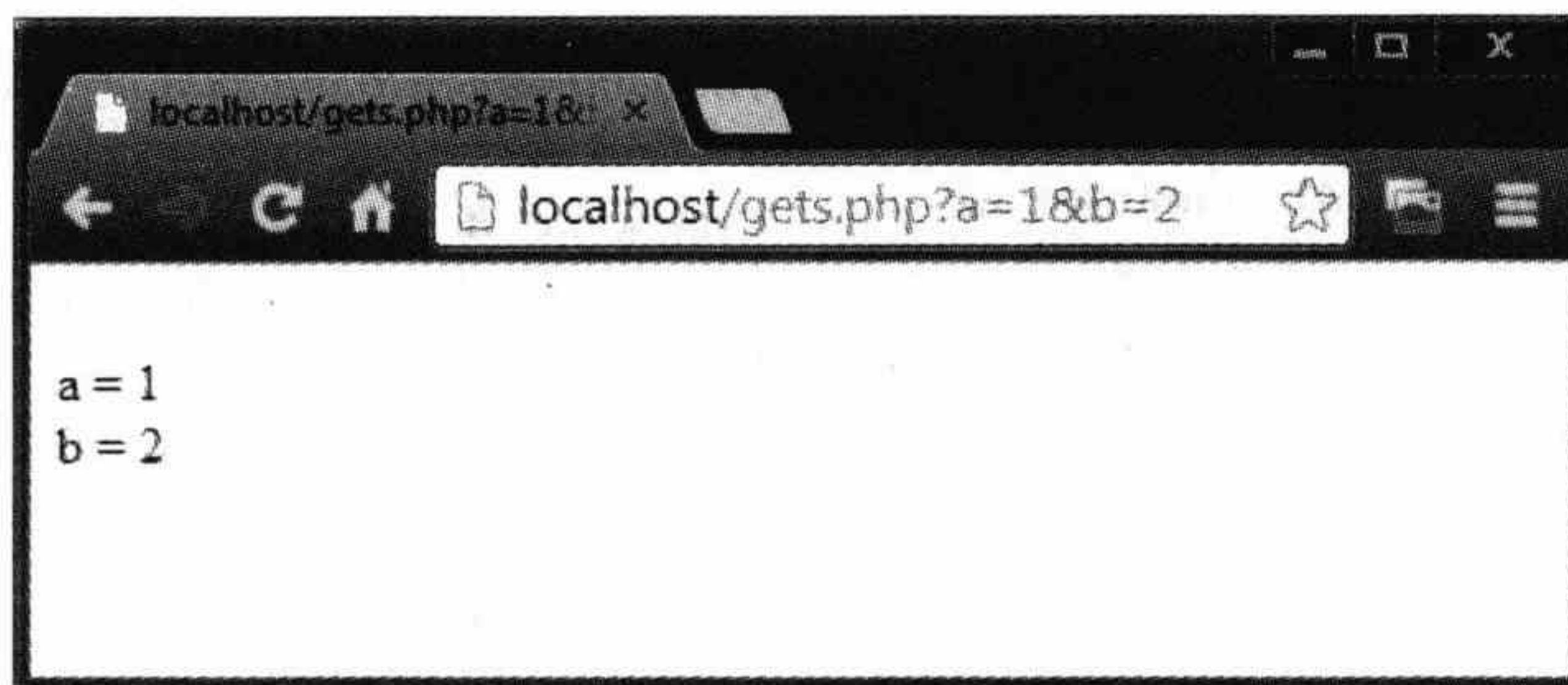


Рис. 8.1. Вывод содержимого массива `$_GET`

Есть один способ, позволяющий эмулировать работу `register_globals`, но использовать его нужно с осторожностью, ведь тогда ваши сценарии опять станут уязвимыми. И не говорите, что я вас не предупреждал! Вот код:

```
foreach($_POST as $key => $value)
    eval("$".$key."='".$value."'");
foreach($_GET as $key => $value)
    eval("$".$key."='".$value."'");
```

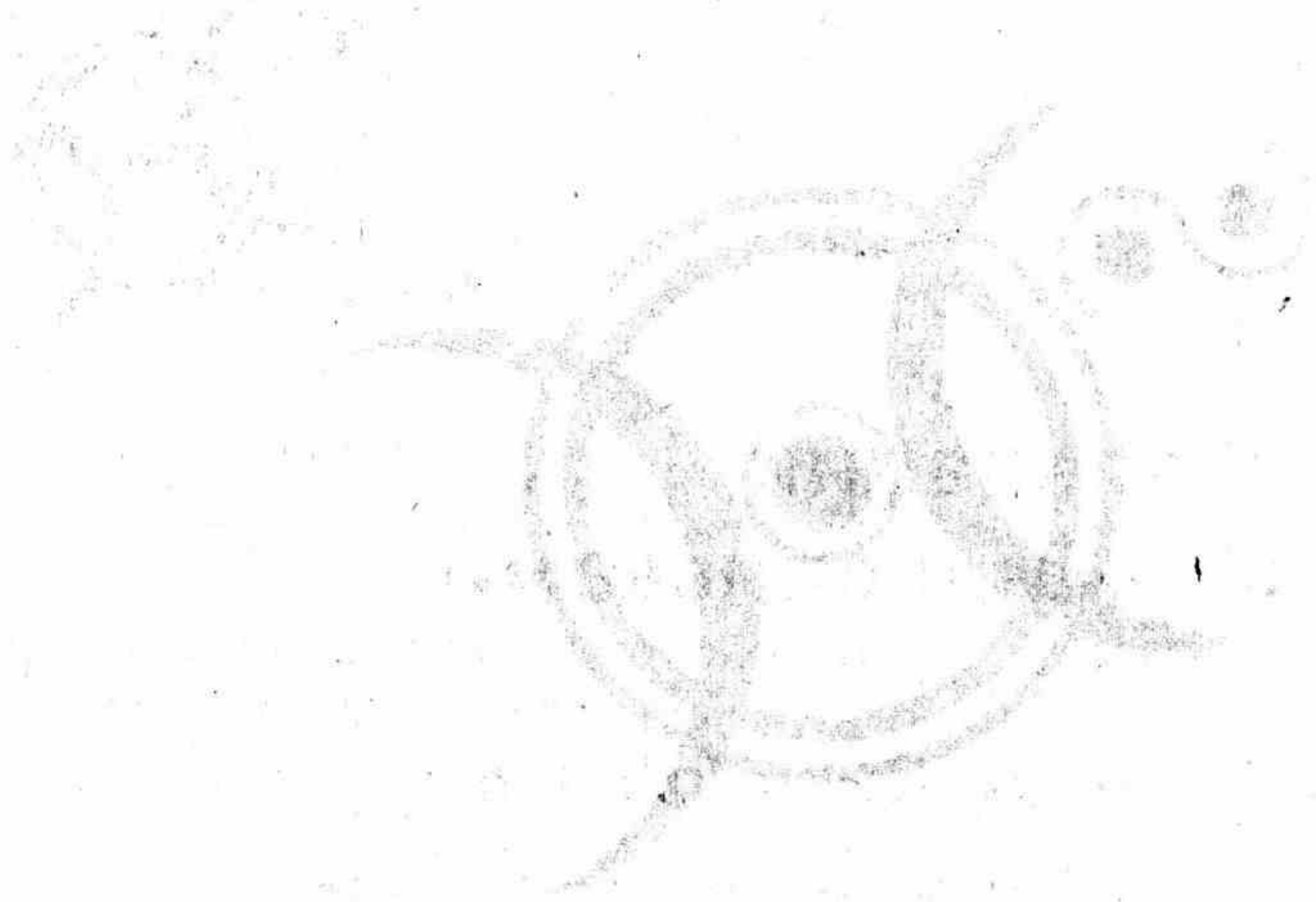
Что делает этот код, думаю, особо комментировать не нужно. Он просто устанавливает все переменные из массивов `$_POST` (ранее `$HTTP_POST_VARS`) и `$_GET` (ранее `$HTTP_GET_VARS`).



I. РАЗДЕЛ 3

Массивы и списки

Глава 9.	Основные операции над массивами и списками
Глава 10.	Функции сортировки массивов
Глава 11.	Особые операции над массивами

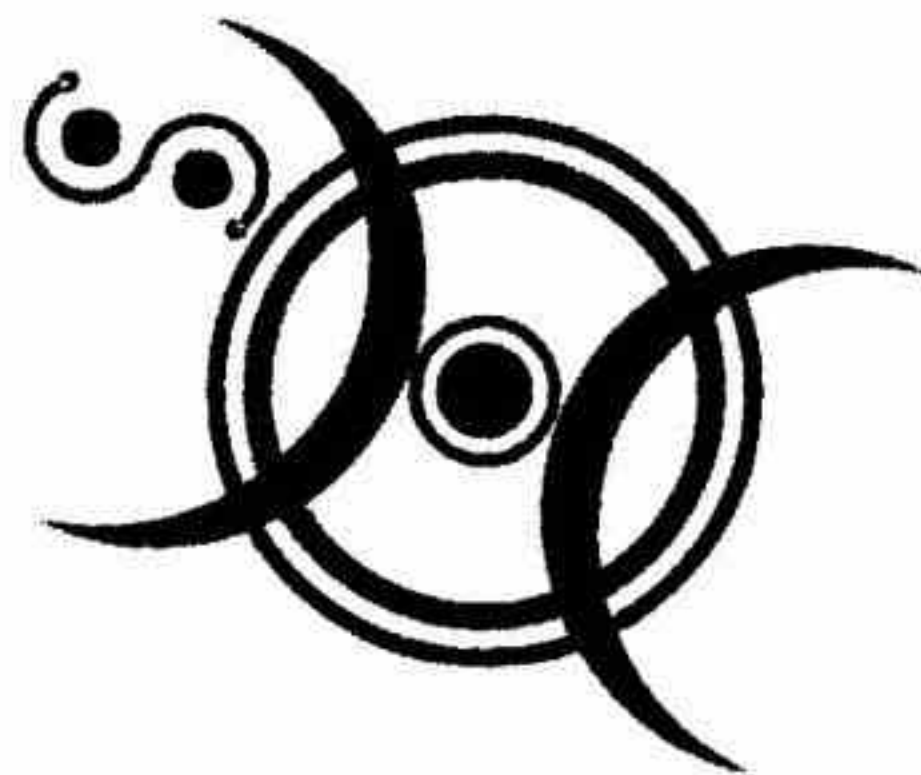


J. PASTOR

THE
CLINICAL
JOURNAL

Published by the
American Medical Association
535 North Dearborn Street
Chicago, Ill. 60610
Subscription price, \$10.00 per annum in advance.
Single copies, \$3.00 each.

ГЛАВА 9



Основные операции над массивами и списками

9.1. Массив и список. Цикл *foreach*

Ранее в этой книге мы уже сталкивались с массивами, поскольку без них нельзя было реализовать тот или иной фрагмент кода. И в дальнейшем с массивами вы будете встречаться очень часто.

Массив — это упорядоченный набор данных. У каждого элемента массива есть свой индекс, или ключ (*index*, *key*), который однозначно идентифицирует элемент массива, — таким образом, индекс уникален для каждого элемента.

Давайте создадим массив:

```
$os[0] = "UNIX";  
$os[1] = "Linux";  
$os[2] = "Windows XP";  
$os[3] = "Windows Vista";  
$os[4] = "Mac OS";
```

Массив `$os` содержит пять элементов, нумерация которых начинается с нуля. Поскольку после имени переменной мы указываем квадратные скобки, РНР понимает, что мы имеем дело с массивом, а не с обычной переменной.

Вывести все элементы массива можно так:

```
for ($i=0; $i < count($os); $i++) echo "$os[$i] <br>";
```

В этом цикле нет ничего необычного, кроме функции `count()`. Эта функция возвращает количество элементов массива. Для функции `count()` также определен псевдоним `sizeof()`. Какую функцию использовать: `count()` или `sizeof()` — разницы нет, используйте ту, название которой вам больше нравится.

РНР обладает очень удобной функцией — при добавлении в массив нового элемента совсем не обязательно указывать его индекс, РНР присвоит ему следующий свободный индекс. Приведенный ранее массив можно было бы создать так:

```
$os[] = "UNIX";  
$os[] = "Linux";  
$os[] = "Windows XP";
```



```
$os[] = "Windows Vista";  
$os[] = "Mac OS";
```

Нумерация элементов в нем также будет начата с нуля.

По типам массивы в PHP различаются на ассоциативные массивы и списки.

- ❑ **Список** — это обычный массив, где в качестве индексов элементов выступают числа. В нашем случае массив `$os` представляет собой список.
- ❑ **Ассоциативный массив** — это массив, в котором в качестве индекса элементов массива выступают произвольные строки. Вот пример создания ассоциативного массива:

```
$DB["host"] = "localhost";  
$DB["user"] = "root";  
$DB["password"] = "";  
$DB["db_name"] = "test";
```

Ассоциативный массив `$DB` содержит параметры доступа к базе данных: узел, имя пользователя, пароль и имя базы данных. Обратиться к любому из элементов массива можно по его индексу:

```
echo $DB["host"];
```

Очевидно, для вывода всех элементов массива оператор `for` не подходит. Специально для перебора ассоциативных массивов используется оператор цикла `foreach`. Синтаксис его таков:

```
foreach ($массив as $ключ => $значение) { тело; }
```

Например:

```
foreach ($DB as $key => $value)  
    echo "$key => $value <br>";
```

Впрочем, чисто технически можно было бы реализовать перебор ассоциативного массива циклом `for`, например:

```
for (reset($DB); ($k = key($DB)); next($DB))  
    echo "$k => " . current($DB) . "<br>";
```

Цикл `for` также удобно применять для обратного перебора ассоциативного массива:

```
for (End($DB); ($k = key($DB)); Prev($DB))  
    echo "$k = {$DB[$k]} <br>";
```

В последних двух примерах мы использовали функции `reset()`, `end()`, `next()`, `prev()`, `current()` и `key()`:

- ❑ функция `reset()` устанавливает внутренний указатель массива на первый элемент массива, функция `end()` — на последний;
- ❑ функция `next()` устанавливает указатель на следующий элемент массива, а функция `prev()` — на предыдущий;
- ❑ функция `key()` получает текущий ключ массива, а функция `current()` возвращает текущий элемент массива, т. е. значение текущего элемента. Вместо функции

`current()` можно использовать функцию `pos()`, которая является псевдонимом `current()`.

Для прямого (не обратного) перебора массива цикл `foreach` использовать гораздо удобнее. Вот пример перебора глобального массива `$_SERVER`, содержащего все глобальные переменные нашего сервера:

```
<?php

foreach($_SERVER as $key => $value)
    echo "$key => $value\n";

?>
```

Вывод сценария изображен на рис. 9.1. Особой ценности этот вывод не имеет, однако на примере данного массива удобно продемонстрировать работу цикла `foreach`.

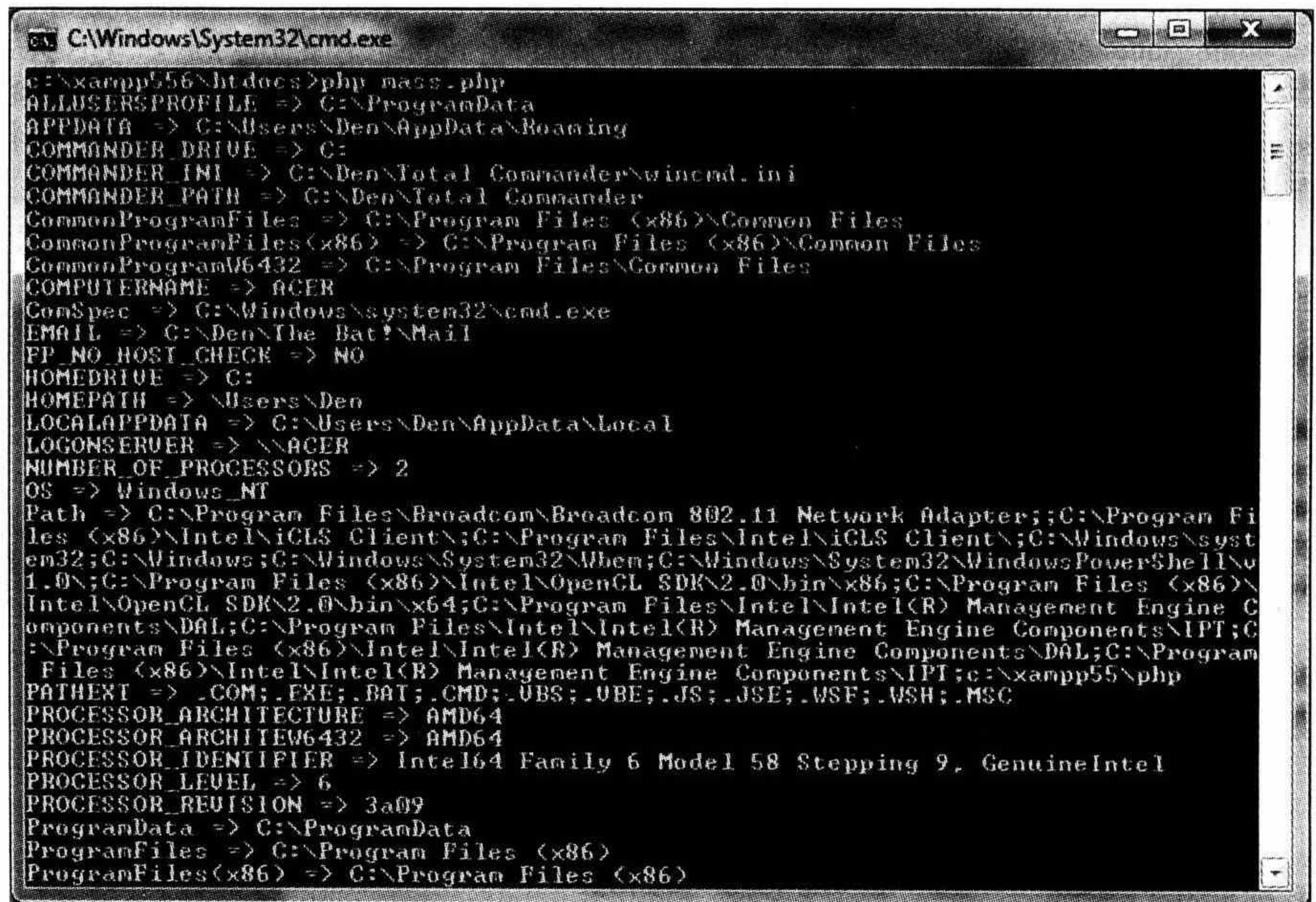


Рис. 9.1. Вывод массива `$_SERVER`

9.2. Функции *list()* и *array()*

Вернемся к нашему массиву `$os` и рассмотрим функцию `list()`:

```
list($os0, $os1, $os2, $os3, $os4) = $os;
```


Переменным `$os0`–`$os5` будут присвоены значения элементов массива `$os[0]`–`$os[5]` соответственно.

Функция `array()` служит для создания массивов — как списков, так и ассоциативных:

```
// пример создания списка
```

```
$os = array("UNIX", "Linux", "Windows XP", "Windows 7", "Mac OS");
```

```
// ассоциативный массив
```

```
$DB = array("host" => "localhost", "user" => "root",  
           "password" => "", "db_name" => "test");
```

Чуть ранее мы создавали массивы `$os` и `$DB` другим способом. Представленные сейчас способы аналогичны рассмотренным ранее. Вам нужно выбрать тот способ, который кажется вам наиболее удобным.

Элементы массива тоже могут быть массивами, т. е. вы можете создавать многомерные массивы. Вот пример массива, содержащего параметры для двух MySQL-соединений:

```
$DB = array (  
    "0" => array("host" => "localhost", "user" => "root",  
                "password" => "", "db_name" => "test" ),  
    "1" => array("host" => "server.ru", "user" => "root",  
                "password" => "", "db_name" => "test" )  
);
```

Получить доступ к параметрам первого соединения можно так:

```
echo $DB["0"]["host"];  
echo $DB["0"]["user"];
```

В PHP 5.5 появилась возможность использовать функцию `list()` внутри цикла `foreach` — например:

```
$arr1 = [  
    [1, 2],  
    [3, 4],  
    [5, 6]  
];  
  
foreach ($arr1 as list($x, $y)) {  
    echo "X = $x; Y = $y <BR>";  
}
```

Сценарий выведет следующее:

```
X = 1; Y = 2  
X = 3; Y = 4  
X = 5; Y = 6
```


9.3. Удаление массива

Вам вряд ли понадобится удалять массив, разве что он содержит очень много информации, которую вы уже быстренько (помните об ограничении на время выполнения сценария?) обработали и хотите продолжить работу сценария, но так, чтобы не превысить ограничение на размер используемой памяти.

Удалить массив можно так же, как и обычную переменную, — с помощью функции `unset()`:

```
unset($DB);
```

9.4. Слияние массивов

Слияние (объединение) массивов подразумевает создание одного массива, состоящего из элементов объединяемых массивов. Слияние осуществляется с помощью оператора `+`.

Предположим, что у нас есть два массива:

```
$SERVER = array("host" => "localhost", "user" => "root", "password" => "");  
$DB = array("dbname" => "test", "table" => "example");
```

Выполним их слияние:

```
$DBINFO = $SERVER + $DB;
```

Получим следующий массив:

```
"host" => "localhost", "user" => "root", "password" => "",  
"dbname" => "test", "table" => "example"
```

Напомню, что вывести массив можно так:

```
foreach($DBINFO as $key => $value)  
    echo "$key => $value\n";
```

Этот вывод изображен на рис. 9.2.

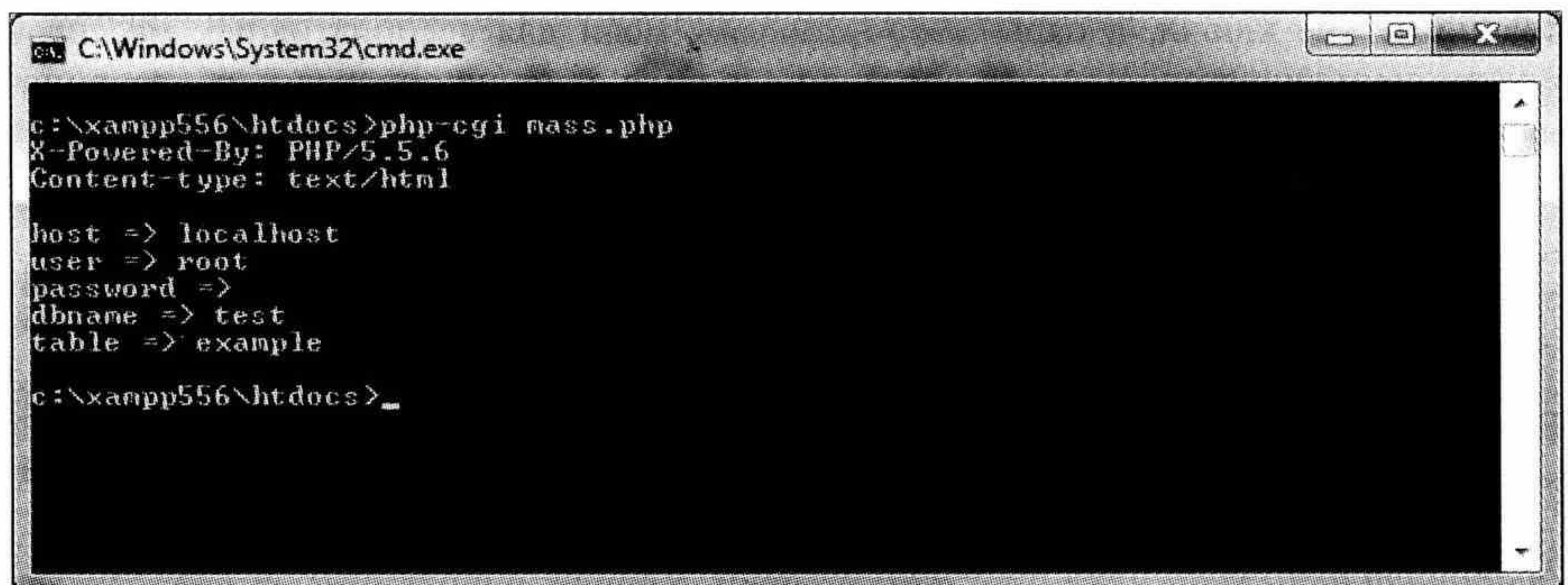


Рис. 9.2. Вывод массива `$DBINFO`

А теперь попробуем объединить два списка, например:

```
$A1 = array(1, 2, 3);
$A2 = array(4, 5, 6);
$A3 = $A1 + $A2;
```

При отладке, когда не нужен "красивый" вывод, гораздо проще использовать функцию `var_dump()`, чем цикл `foreach`, и перебирать массив с ее помощью. `var_dump()` сделает за нас всю работу и предоставит более информативный вывод (рис. 9.3):

```
var_dump($A3);
```

Однако результат вывода нас удивит — полученный массив `$A3` вместо значений 1, 2, 3, 4, 5, 6 будет содержать только значения 1, 2, 3.

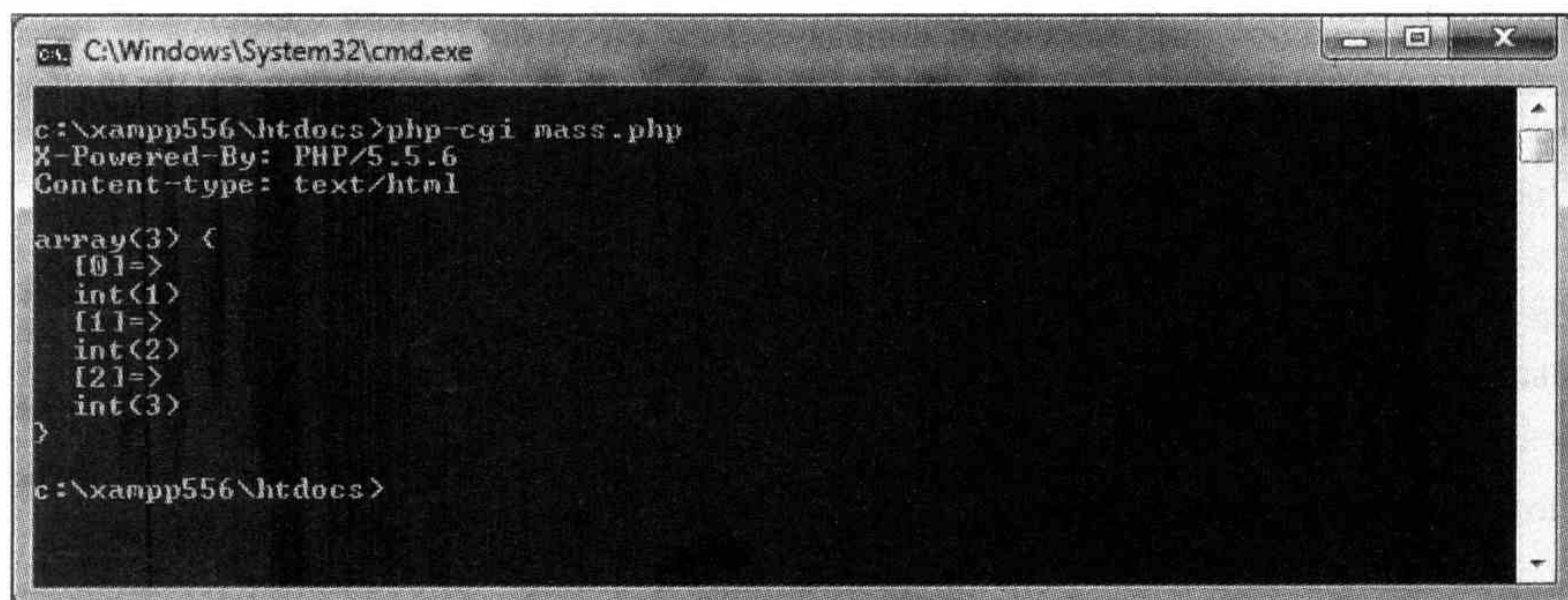


Рис. 9.3. Вывод массива `$A3`

Почему? Это особенность оператора `+`. Если при слиянии массивов будут найдены элементы с одинаковыми индексами, то оператор `+` возьмет их из первого массива. Вот так-то...

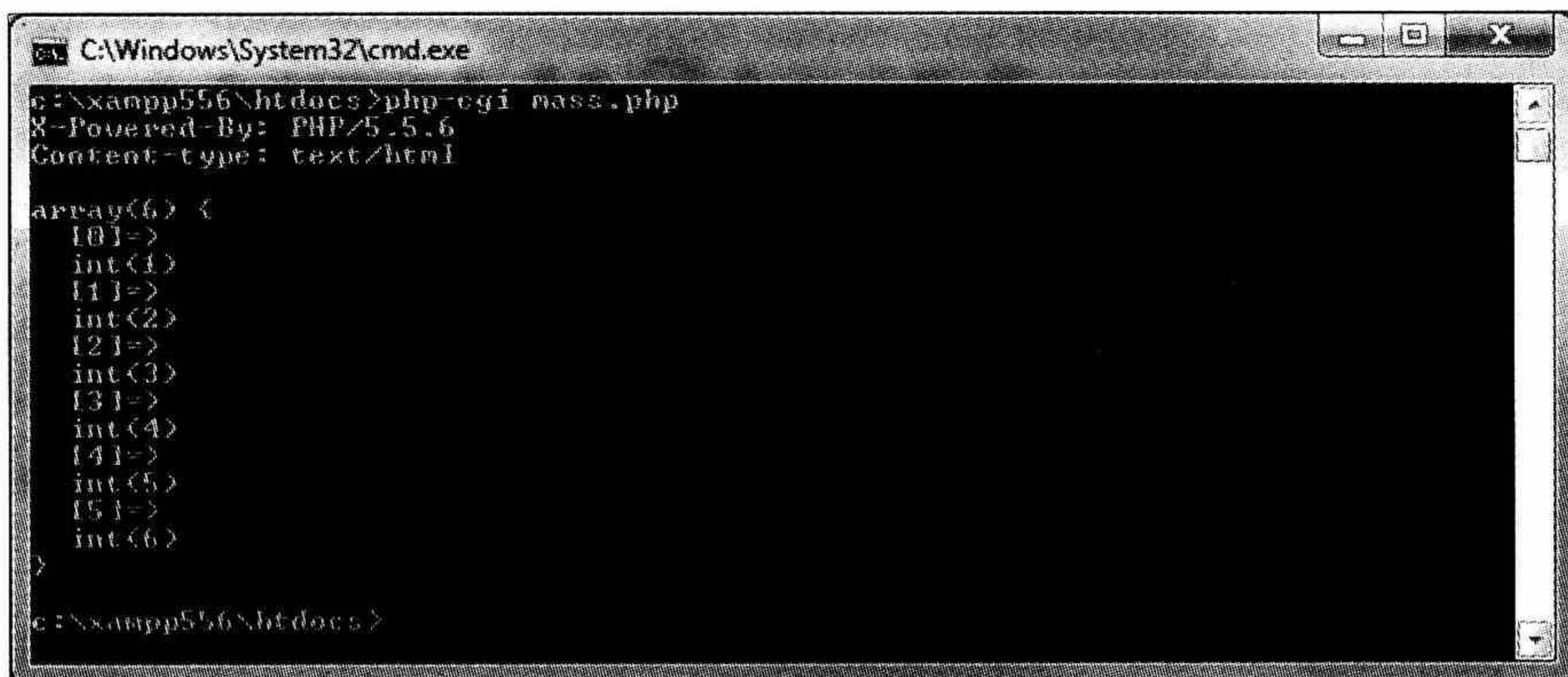
Устранить этот недостаток можно с помощью функции `array_merge()`, которая сравнивает не ключи, а пары `ключ=>значение`. Если для ассоциативных массивов с успехом можно использовать оператор `+`, то для слияния списков удобно применять `array_merge()`:

```
$A1 = array(1, 2, 3);
$A2 = array(4, 5, 6);
$A3 = array_merge($A1, $A2);
var_dump($A3); // см. рис. 9.4
```

В этом случае мы получим то, что нам нужно, а именно — массив `$A3`, содержащий все значения массивов `$A1` и `$A2` (рис. 9.4).

Для слияния многомерных массивов служит функция `array_merge_recursive()`:

```
array array_merge_recursive ( array $array1 [, array $... ] )
```


Рис. 9.4. Результат слияния массивов функцией `array_merge()`

9.5. Функция `print_r()`

Иногда нужно вывести массив в отладочных целях — просто чтобы посмотреть его элементы. Мы уже познакомились с функцией `var_dump()`, а теперь рассмотрим функцию `print_r()`, вывод которой еще лучше подходит для отображения содержимого массивов. Какой из этих двух функций воспользоваться, решать вам: функция `var_dump()` выводит больше полезной информации — например, тип значения, а вывод функции `print_r()` более понятен для человека и не содержит ничего лишнего.

Для глубокого анализа лучше, конечно, взять `var_dump()`, а для того, чтобы просто "заглянуть" внутрь массива, удобнее применить `print_r()`. Помните, что обе эти функции используются исключительно из соображений отладки, и их не следует применять в производственной версии вашего кода.

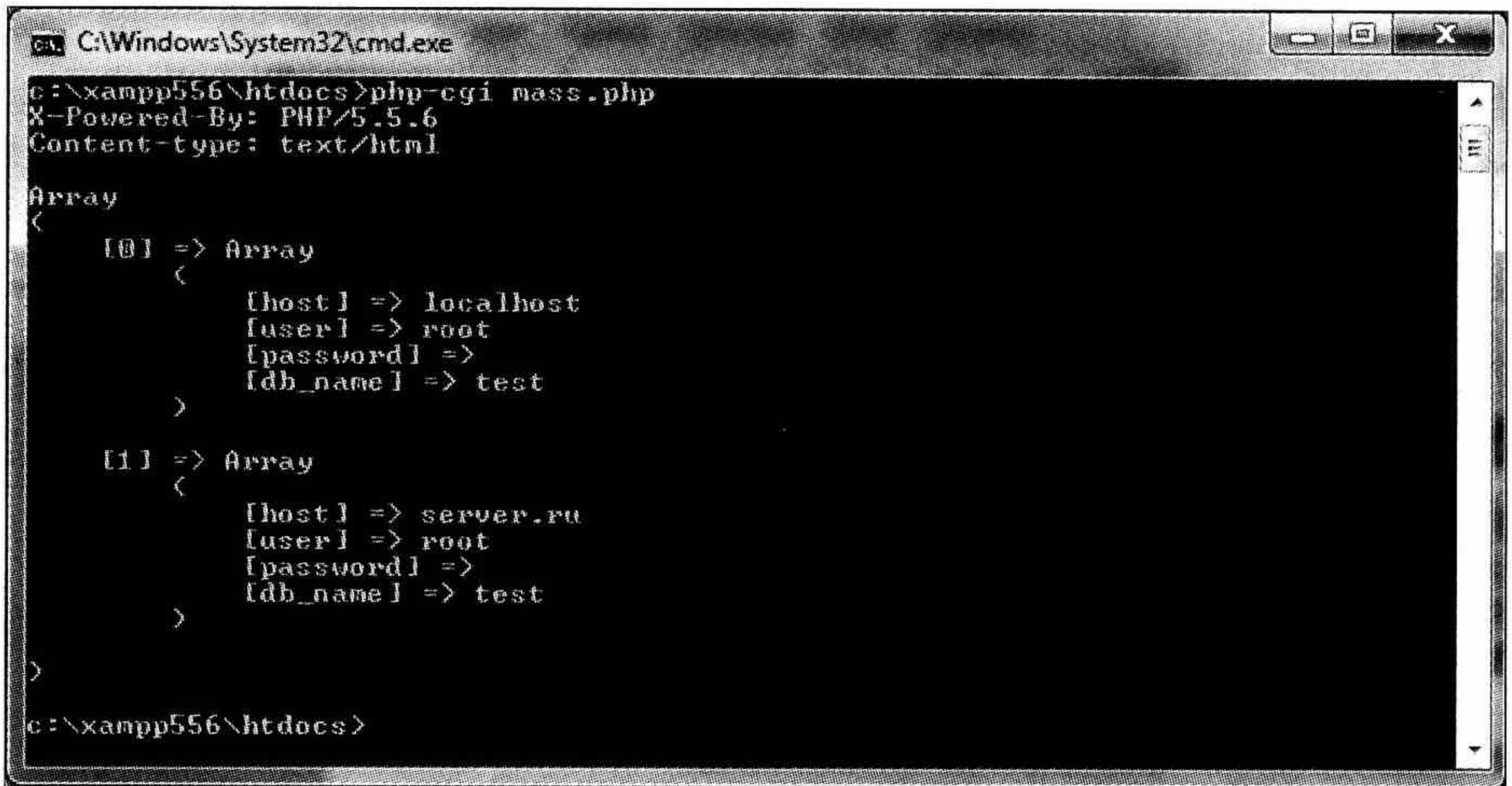
Итак, чтобы не возиться с `foreach()` и `echo`, намного проще вызвать функцию `print_r()`:

```
$DB = array (
    "0" => array( "host" => "localhost", "user" => "root",
        "password" => "", "db_name" => "test" ),
    "1" => array( "host" => "server.ru", "user" => "root",
        "password" => "", "db_name" => "test" )
);

print_r($DB);
```

Вывод функции `print_r()` изображен на рис. 9.5.

Вообще функция `print_r()` довольно разносторонняя и позволяет "распечатать" информацию о переменной любого типа: от обычной текстовой переменной до MySQL-результата или объекта.



```
C:\Windows\System32\cmd.exe
c:\xampp556\htdocs>php-cgi mass.php
X-Powered-By: PHP/5.5.6
Content-type: text/html

Array
(
    [0] => Array
        (
            [host] => localhost
            [user] => root
            [password] =>
            [db_name] => test
        )
    [1] => Array
        (
            [host] => server.ru
            [user] => root
            [password] =>
            [db_name] => test
        )
)

c:\xampp556\htdocs>
```

Рис. 9.5. Вывод массива с помощью функции `print_r()`

9.6. Разыменовывание массива

Возможность разыменовывания массива (array dereferencing) появилась еще в версии PHP 5.4. Подробнее об этой функции вы можете прочитать по адресу <http://ua2.php.net/manual/en/language.types.array.php> (пример 7), а мы подойдем, как обычно, с практической точки зрения.

Представим, что у нас есть функция, возвращающая массив:

```
function a() {
    return array(0, 1, 1, 0, 1, 0, 0, 1);
}
```

Как можно видеть, функция возвращает массив из восьми элементов. Раньше нужно было получить результат этой функции (массив), а потом уже обращаться к элементам полученного массива:

```
$arr1 = a();
echo $arr1[5];
```

Начиная с версии 5.4, вы можете сделать это так:

```
echo a()[5];
```

Другими словами, есть возможность обращаться к элементам массива, возвращаемого функцией, без предварительного присваивания массива временной переменной.

Я бы не сказал, что это очень важная функция PHP, но о ней нельзя не упомянуть, иначе вы, увидев подобный код, не будете знать, что он означает. Лично я предпочитаю не прибегать к разыменовыванию массива. Хотя бы потому, что такой код

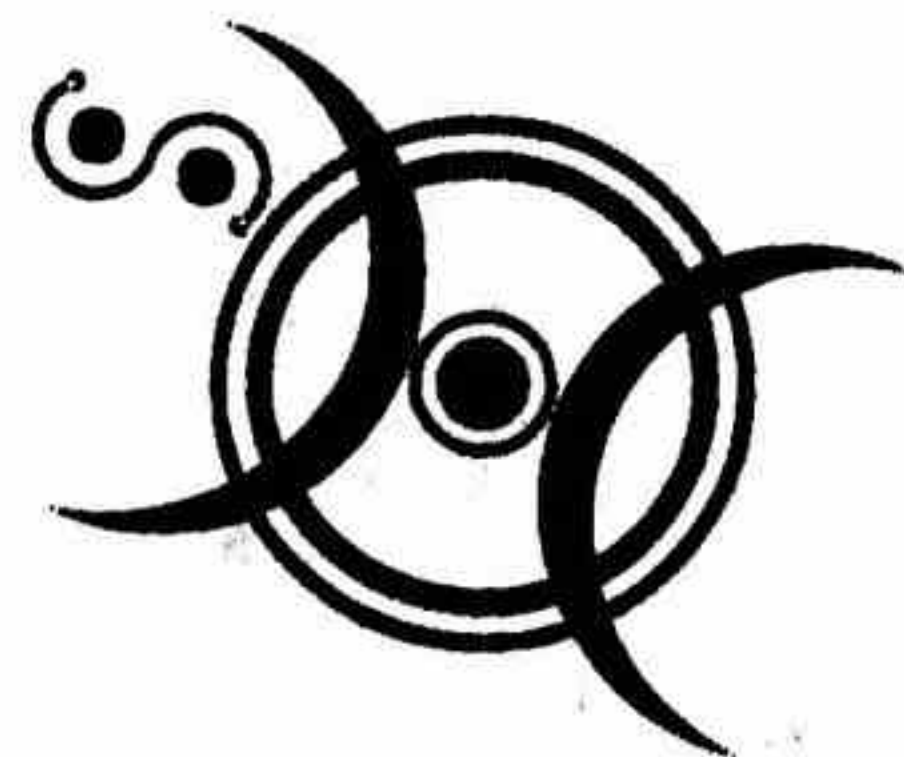
окажется не совместим с более ранними версиями PHP, — например, с 5.3, которая все еще актуальна даже в 2014 году. Да и в компактности кода выигрыш здесь не огромный — вы экономите всего лишь одну строчку кода.

То же и в части эффективности использования памяти. В любом случае функция возвращает массив, а не один лишь элемент, поэтому память все равно выделяется. Если вы работаете с очень большими массивами информации и заботитесь об использовании памяти, просто не забывайте очищать временные переменные функцией `unset()`. Да, код, может, окажется и не таким компактным, зато будет совместимым с предыдущими версиями:

```
$arr1 = a();  
echo $arr1[5];  
unset($arr1);
```

Впрочем, в PHP еще не все так плохо с обратной совместимостью, как в том же Python, где в этом плане имеются серьезные проблемы. Вышла версия 3, которая полностью не совместима с предыдущими версиями, и многие программисты теперь думают, что делать: или переписывать код заново, или остаться на старой версии. В PHP таких проблем нет, и большая часть кода будет совместима со старыми версиями, поскольку новшества вводятся постепенно. Сегодня мы не задумываясь наслаждаемся всеми прелестями объектно-ориентированного программирования (см. главу 24), а ведь в 4-й версии PHP ничего подобного не было! Однако переход от четвертой версии к пятой производился постепенно и не так резко, как в случае с Python...

ГЛАВА 10



Функции сортировки массивов

10.1. Сортировка массивов

Сортировка — это одна из самых востребованных операций над массивом. Получив в массив данные (например, из файла), вам нужно их отсортировать и вывести в браузер пользователю.

В PHP есть много функций сортировки массива. В этой главе мы изучим лишь самые востребованные, а также рассмотрим создание собственной функции сортировки.

10.2. Функция `sort()` — сортировка списка

Функция `sort()` служит для сортировки обычных массивов, т. е. списков. Прототип этой функции выглядит так:

```
bool sort ( array &array [, int sort_flags] )
```

Рассмотрим пример использования этой функции:

```
$A = array (3, 1, 9, 5, 0, 4, 5); // исходный список
sort($A);                       // сортируем
print_r($A);                     // выводим
```

Результат сортировки представлен на рис. 10.1. У функции `sort()` есть аналог — функция `rsort()`, выполняющая сортировку в обратном порядке. Набор параметров у нее такой же, как у функции `sort()`.

Вот несколько фактов, которые вам необходимо знать об этой функции:

- ❑ функция возвращает `true`, если сортировка успешно завершена. Обычно так оно и происходит, поэтому нет необходимости проверять результат функции;
- ❑ функция назначает новые ключи для элементов массива, поэтому ее нельзя использовать для сортировки ассоциативных массивов, иначе вы рискуете потерять данные;
- ❑ функция выполняет сортировку по возрастанию (или в алфавитном порядке, если элементы массива — строки);

- ❑ вы можете указать следующие флаги сортировки (второй параметр функции):
- `SORT_REGULAR` — обычное сравнение элементов без изменения их типов;
 - `SORT_NUMERIC` — сравнивать элементы как числа;
 - `SORT_STRING` — сравнивать элементы как строки;
 - `SORT_LOCALE_STRING` — сравнивать элементы как строки, учитывая текущую "локаль" (языковые параметры системы).



```
C:\WINDOWS.0\System32\cmd.exe
D:\PHP6>php cgi mass.php
X Powered By: PHP/4.0.0 dev
Content-type: text/html; charset: UTF-8

Array
(
    [101] => 0
    [111] => 1
    [121] => 3
    [131] => 4
    [141] => 5
    [151] => 5
    [161] => 9
)
D:\PHP6>
```

Рис. 10.1. Результат сортировки массива

10.3. Функция *asort()* — сортировка массива по значениям

В PHP вам чаще придется работать с ассоциативными массивами. Для их сортировки предназначена функция `asort()`:

```
bool asort ( array &array [, int sort_flags] )
```

Функция `asort()` выполняет сортировку по значениям. Представим, что у нас есть массив `$fruits`. Обратите внимание, что ни ключи, ни значения этого массива не расположены в алфавитном порядке:

```
$fruits = array("b" => "banana", "a" => "orange", "c" => "apple");
```

Выполним его сортировку:

```
asort($fruits);
```

А теперь выведем содержимое массива функцией `print_r()`:

```
Array
(
    [c] => apple
```



```
[b] => banana
[a] => orange
)
```

Как видите, функция `asort()` выполнила сортировку по значениям массива — они расположены в алфавитном порядке.

Кроме функции `asort()` вы можете использовать функцию `arsort()`, выполняющую сортировку по значениям в обратном порядке. Результат применения этой функции к нашему массиву будет таким:

```
Array
(
    [a] => orange
    [b] => banana
    [c] => apple
)
```

Флаги сортировки у функций `asort()` и `arsort()` такие же, как у функции `sort()`.

10.4. Функция `ksort()` — сортировка по ключам

Функция `ksort()` работает подобно функции `asort()`, но выполняет сортировку не по значениям, а по ключам. Сортировку по ключам в обратном порядке обеспечивает функция `krsort()`.

Сортировку по ключам иллюстрирует следующий фрагмент кода:

```
// исходный массив
$fruits = array("b" => "banana", "a" => "orange", "c" => "apple");
// сортировка по ключам
ksort($fruits);
print_r($fruits);
// сортировка по ключам в обратном порядке
krsort($fruits);
print_r($fruits);
// вывод будет следующим:
Array
(
    [a] => orange
    [b] => banana
    [c] => apple
)
Array
(
    [c] => apple
    [b] => banana
    [a] => orange
)
```


10.5. Функции `array_reverse()` и `shuffle()`

Функция `array_reverse()` возвращает массив с элементами в обратном порядке, например:

```
$fruits = array("b" => "banana", "a" => "orange", "c" => "apple");
$reverse_fruits = array_reverse($fruits);
```

Массив `$reverse_fruits` будет таким:

```
Array
(
    [c] => apple
    [a] => orange
    [b] => banana
)
```

Функция `shuffle()` перемешивает элементы массива в произвольном порядке. Каким окажется вывод `shuffle()`, сказать сложно — в этом и прелесть случайной сортировки. Однако помните, что данная функция полностью изменяет массив, поэтому ее нужно использовать только для списков.

10.6. Собственная функция сортировки

До этого мы сортировали массивы, состоящие из строк и чисел. Но в массив вы можете поместить все что угодно — например, результаты MySQL-запросов. Как в этом случае выполнить сортировку?

Здесь вам помогут "пользовательские" версии уже рассмотренных функций:

- ❑ `usort()` — сортировка списка;
- ❑ `uasort()` — сортировка массива по значениям;
- ❑ `uksort()` — сортировка массива по ключам.

Этим функциям нужно передать два параметра: массив, который нужно отсортировать, и *пользовательскую функцию сортировки*.

Функция сортировки должна принять два аргумента (тип аргументов такой же, как у элементов массива) и "сказать", какой из двух аргументов больше. Точнее, функция должна передать одно из следующих значений:

- ❑ `-1`, если первый элемент меньше второго;
- ❑ `0`, если элементы равны;
- ❑ `1`, если первый элемент больше второго.

Рассмотрим небольшой пример:

```
<?php
function cmp($a, $b)
{
    if ($a == $b) {
```



```

        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$a = array(4, 2, 5, 7, 1);

usort($a, "cmp");

print_r($a);

?>

```

Наша функция сравнения — `cmp()`. Она не делает ничего сверхъестественного — просто сравнивает числа: аргументы `$a` и `$b`. Если `$a` равно `$b`, то функция возвращает 0, если `$a` меньше `$b`, то возвращается `-1` или `1` — в противном случае.

Обратите внимание на конструкцию:

```
return ($a < $b) ? -1 : 1;
```

Работает она так. Вычисляется истинность выражения в скобках. Если выражение истинно (в данном случае `$a < $b`), то возвращается первое значение после вопросительного знака (`-1`), в противном случае возвращается второе значение (`1`).

10.7. Натуральная сортировка

Представим, что у нас есть массив, содержащий имена файлов:

```
$files= array("ch13.doc", "ch10.doc", "ch3.doc", "ch1.doc");
```

Попробуем отсортировать его функцией `asort()`:

```
asort($files);
print_r($files);
```

Результат будет таким:

```

Array
(
    [3] => ch1.doc
    [1] => ch10.doc
    [0] => ch13.doc
    [2] => ch3.doc
)

```

С точки зрения компьютера ошибок нет — все элементы отсортированы в лексикографическом порядке. Но нам бы хотелось, чтобы файл главы 3 следовал сразу после файла главы 1. Чтобы реализовать задуманное, нам нужно использовать алгоритм натуральной сортировки:

```
bool natsort ( array &$array )
```


Если отсортировать наш массив функцией `natsort()`, результат будет таким:

```
Array
(
    [3] => ch1.doc
    [2] => ch3.doc
    [1] => ch10.doc
    [0] => ch13.doc
)
```

Функция `natsort()` чувствительная к регистру символов, но в PHP есть аналогичная функция, которая игнорирует регистр символов:

```
bool natcasesort ( array &$array )
```

Сравним результаты `natsort()` и `natcasesort()`:

```
$files = $files2 = array("ch13.doc", "Ch10.doc", "ch3.doc", "ch1.doc");
natsort($files);
print_r($files);
```

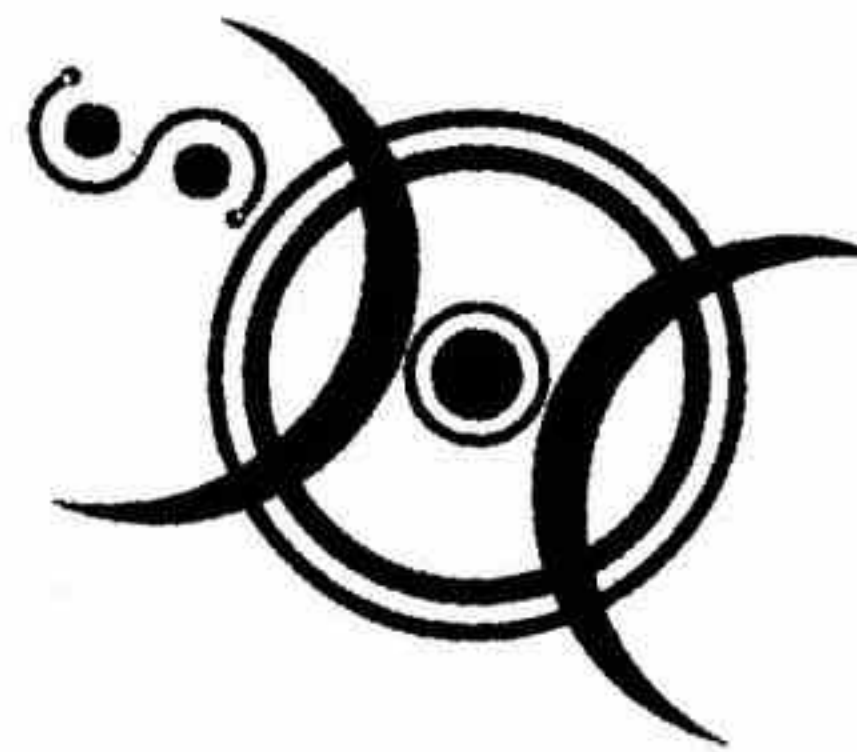
```
natcasesort($files2);
print_r($files2);
```

А теперь посмотрим на вывод:

```
Array
(
    [1] => Ch10.doc
    [3] => ch1.doc
    [2] => ch3.doc
    [0] => ch13.doc
)
```

```
Array
(
    [3] => ch1.doc
    [2] => ch3.doc
    [1] => Ch10.doc
    [0] => ch13.doc
)
```

Второй результат — сортировка функцией `natcasesort()` — более соответствует представлениям о "человеческой" сортировке, не говоря уже о сортировке с помощью `asort()`.



ГЛАВА 11

Особые операции над массивами

11.1. Добавление и удаление элементов массива

Добавить элемент в конец массива можно с помощью оператора []:

```
$массив[] = новый_элемент;
```

Кроме оператора [] можно использовать функцию `array_push()`:

```
int array_push ( array &array, mixed var [, mixed ...] )
```

После добавления элемента в конец массива функция возвращает новое количество элементов массива. Заметьте, что с помощью `array_push()` можно добавить не один элемент, а несколько, — например:

```
array_push(массив, элемент1, элемент2, элемент3 ...)
```

Для удаления последнего элемента массива служит функция `array_pop()`:

```
mixed array_pop ( array &array )
```

Допустим, у нас есть массив фруктов:

```
$fruits = array("orange", "banana", "apple", "linux");
```

Удалим "лишний" элемент, который не имеет никакого отношения к фруктам:

```
$element = array_pop($fruits);
```

Теперь посмотрим, что после этого будут содержать массив `$fruits` и переменная `$element`:

```
print_r($fruits);  
print_r($element);
```

Вот, что выведут функции `print_r()`:

```
Array  
(  
    [0] => orange
```



```
[1] => banana
[2] => apple
)
linux
```

Как видите, из массива `$fruits` удален последний элемент, а переменная `$element` как раз и содержит этот элемент.

Теперь попробуем добавить элементы в начало списка. Для этого нужно использовать функцию `array_unshift()`:

```
int array_unshift ( array &array, mixed var [, mixed ...] )
```

Данная функция, как и `array_push()`, может добавить в массив сразу несколько элементов, например:

```
$fruits = array("orange", "banana", "apple");
array_unshift($fruits, "lemon", "tangerine");
```

Теперь наш массив будет выглядеть так:

```
Array
(
    [0] => lemon
    [1] => tangerine
    [2] => orange
    [3] => banana
    [4] => apple
)
```

Удалить первый элемент массива можно с помощью функции `array_shift()`:

```
array_shift($fruits);
```

После этого массив будет выглядеть так:

```
Array
(
    [0] => tangerine
    [1] => orange
    [2] => banana
    [3] => apple
)
```

11.2. Упаковка переменных в массив и их извлечение

Предположим, что у нас есть три переменные:

```
$host = "localhost";
$user = "root";
$password = "secret";
```


Упакуем их в массив `$DB` с помощью функции `compact()`:

```
$DB = compact("host", "user", "password");
```

ПРИМЕЧАНИЕ

Заметьте, что функции `compact` нужно передать имена переменных, а не их значения! То есть нужно вызвать функцию как `compact("host", "user", "password")`, а не `compact($host, $user, $password)`.

Функция `compact()` создаст следующий массив:

```
Array
(
    [host] => localhost
    [user] => root
    [password] => secret
)
```

Для распаковки ключей массива в переменные служит функция `extract()`:

```
int extract ( array var_array [, int extract_type [, string prefix]] )
```

Например:

```
extract($DB);
```

При распаковке массива будут созданы переменные с именами, соответствующими названиям ключей массива, т. е. `$host`, `$user` и `$password`. Значения этих переменных будут такими же, как у соответствующих им ключей массива.

Все вроде просто, если не принимать во внимание один факт. Что делать, если переменные `$host`, `$user` и `$password` уже существуют?

Поведение функции регулируется вторым параметром:

- ☐ `EXTR_OVERWRITE` — если распаковываемая переменная уже существует, она будет перезаписана;
- ☐ `EXTR_SKIP` — если переменная с таким именем существует, будет сохранено ее прежнее значение;
- ☐ `EXTR_PREFIX_SAME` — если переменная существует, к ее имени будет добавлен префикс, заданный третьим параметром функции `extract()`;
- ☐ `EXTR_PREFIX_ALL` — префикс, заданный третьим параметром, будет добавлен ко всем извлекаемым переменным.

В главе 7 мы познакомились с функцией `serialize()`, которая может сериализировать массив, т. е. свести его к строке. После чего его можно будет сохранить в Cookies или в базе данных. Предположим, что нам нужно сохранить в Cookies несколько переменных. Чтобы не сохранять их все по отдельности, можно запаковать их в массив, затем его сериализовать и сохранить полученную строку как одну переменную.

Как только эти переменные нам понадобятся, мы получим из Cookies сохраненную строку, затем превратим ее в массив функцией `unserialize()` и распакуем массив в переменные.

11.3. Получение части массива

Функция `array_slice()` возвращает часть массива `array` длиной `length` начиная с индекса `offset`:

```
array array_slice ( array array, int offset [, int length [, bool
preserve_keys]] )
```

Если параметр `length` не задан, то возвращаются все элементы массива начиная с `offset`:

```
$input = array(1, 2, 3, 4, 5);
```

```
$output = array_slice($input, 0, 3); // новый массив = 1, 2, 3
```

```
$output = array_slice($input, 2);    // новый массив = 3, 4, 5
```

```
$output = array_slice($input, -2, 1); // новый массив = 4
```

Вы можете задать отрицательное смещение (`offset`) — отсчет будет начинаться с конца массива, а не с начала.

11.4. Функции автоматического заполнения массива

Функция `array_fill()` позволяет произвести заливку массива указанным количеством значений:

```
array array_fill ( int $start_index , int $num , mixed $value )
```

Первый параметр — это начальный индекс массива, второй параметр — количество значений, третий параметр — само значение. Заполним массив `$A` (начиная с индекса 0) значениями 5, всего 10 значений:

```
$A = array_fill(0, 10, 5);
```

Функция `array_fill_keys()` заполняет не только значениями, но и ключами. Ей нужно передать два параметра: массив с ключами, элементам которых будет присвоено значение — второй параметр функции:

```
array array_fill_keys ( array $keys , mixed $value )
```

Рассмотрим небольшой пример:

```
// определяем ключи:
```

```
$keys = array("a", "b", "c");
```

```
// заполняем массив, указываем ключи и значение 777:
```

```
$a = array_fill_keys($keys, 777);
```


Результат будет следующим:

```
Array
(
    [a] => 777
    [b] => 777
    [c] => 777
)
```

Функция `range()` позволяет создать массив, состоящий из диапазона чисел от `$low` до `$high`, шаг ряда чисел задается необязательным третьим параметром, который по умолчанию равен 1:

```
array range ( mixed $low , mixed $high [, number $step = 1 ] )
```

Пример:

```
$R = range(0, 100);
```

11.5. Сравнение массивов

Функция `array_diff()` сравнивает несколько массивов:

```
array array_diff ( array $array1 , array $array2 [, array $ ... ] )
```

Результат работы функции — разница между массивами. Рассмотрим небольшой пример:

```
$a1 = array("a" => "green", "red", "blue", "red");
$a2 = array("b" => "green", "yellow", "red");
$diff = array_diff($a1, $a2);

print_r($diff);
```

Функция `array_diff()` возвращает массив, состоящий из значений массива, заданного первым параметром, которые отсутствуют в любом другом массиве, перечисленном в последующих аргументах.

Функция `array_diff()` не учитывает ключи (индексы), поэтому ее целесообразно использовать для сравнения списков и обычных, не ассоциативных, массивов. В частности, результат предыдущего сравнения будет такой:

```
Array
(
    [1] => blue
)
```

Для сравнения ассоциативных массивов нужно использовать следующую функцию:

```
array array_diff_assoc ( array $array1 , array $array2 [, array $... ] )
```

Функция `array_diff_assoc()` производит дополнительную проверку ключей. Если сравнить массивы `$a1` и `$a2` из предыдущего примера, то результат будет таким:


```
Array
(
    [b] => brown
    [c] => blue
    [0] => red
)
```

Как видно по результату работы функции, она возвращает массив, состоящий из значений массива, заданного первым параметром, которые отсутствуют в любом другом массиве, который перечислен в последующих аргументах.

Для сравнения только ключей массивов используется функция `array_diff_key()`:

```
array array_diff_key ( array array1, array array2 [, array ...] )
```

Функция возвращает массив, содержащий все значения из массива, указанного в первом параметре, имеющие ключи, отсутствующие в последующих параметрах.

11.6. Полезные операции над массивом

11.6.1. Вычисление суммы и произведения всех элементов массива

Функция `array_sum()` возвращает сумму всех элементов массива. Понятно, что все элементы массива должны быть числами:

```
number array_sum ( array $array )
```

Пример:

```
$a = range(1,100);
echo array_sum($a);
```

Для вычисления произведения всех элементов массива служит функция `array_product()`:

```
echo array_product($a);
```

11.6.2. Проверка существования элемента в массиве

Функция `in_array()` возвращает `true`, если элемент `$needle` присутствует в массиве `$haystack`:

```
bool in_array ( mixed $needle , array $haystack [, bool $strict ] )
```

Пример:

```
$a = array("Linux", "Mac OS", "Windows");
if (in_array("BSD", $a)) echo "Элемент BSD находится в массиве";
```

В результате выполнения PHP-кода ничего не будет выведено, поскольку элемента `BSD` в массиве `$a` нет.

Функция `in_array()` существенно сокращает ваш PHP-код — вам больше не нужно возиться с операторами `foreach()` и `for()` для перебора всех элементов массива и сравнения их с эталоном.

Функция `in_array()` производит поиск по значениям, а функция `array_key_exists()` — по ключам, что позволяет проверить существование элемента с указанным ключом:

```
$a= array('name' => 'denis', 'password' => '123');  
if (array_key_exists('name', $a)) {  
    echo "Элемент 'name' существует в массиве";  
}
```

11.6.3. Получение случайного элемента из массива

Функция `array_rand()` получает случайный элемент из массива `$input`:

```
mixed array_rand ( array $input [, int $num_req = 1 ] )
```

Второй параметр задает количество случайных элементов. Функция `array_rand()` возвращает ключ случайного элемента. Если вы указали `$num_req` больше 0, то будет возвращено несколько ключей. При попытке указать больше ключей, чем есть вообще элементов в массиве, вы получите ошибку уровня `E_WARNING`. Небольшой пример:

```
// генерируем массив  
$a = range(0,100);  
// получаем два случайных элемента  
$rkeys = array_rand($a, 2);  
// выводим эти элементы  
echo $input[$rkeys[0]] . "<br>";  
echo $input[$rkeys[1]] . "<br>";
```

Ранее перед вызовом функции `array_rand()` нужно было инициализировать генератор случайных чисел, но, начиная с версии PHP 4.2, это больше делать не нужно.

11.6.4. Удаление дубликатов из массива

Функция `array_unique()` удаляет повторяющиеся элементы из массива:

```
array array_unique(array $array [, int $sort_flags = SORT_STRING ] )
```

Функция не обращает внимание на ключи массива и не изменяет переданный ей массив, а работает с его копией:

```
$a = array(1, 2, 3, 3, 2, 1);  
$unique_a = array_unique($a);
```

В массиве `$unique_a`, как несложно было догадаться, окажутся значения 1, 2, 3.

11.6.5. Получение значений и ключей массива

Функция `array_values()` возвращает все значения массива. Эту функцию удобно использовать для преобразования ассоциативного массива в обычный список. Рассмотрим пример:

```
$array = array("name" => "guest", "password" => "empty");  
print_r(array_values($array));
```

Результат будет следующим:

```
Array  
(  
    [0] => guest  
    [1] => empty  
)
```

Как видите, все ключи ассоциативного массива были удалены, вместо них значениям массива были присвоены числовые индексы.

Кроме функции `array_values()` есть похожая функция — `array_keys()`. Как вы уже догадались, эта функция возвращает все ключи массива в виде списка. Вот результат ее работы:

```
Array  
(  
    [0] => name  
    [1] => password  
)
```

11.6.6. Замена местами значений и ключей

Функция `array_flip()` позволяет поменять местами значения и ключи. Проще всего рассмотреть ее работу на примере:

```
$a = array("a"=>1, "b"=>2);  
print_r(array_flip($a));
```

Результат будет таким:

```
Array  
(  
    [1] => a  
    [2] => b  
)
```

11.6.7. Подсчет значений в массиве

Представим, что у нас есть массив, значения которого могут повторяться. Функция `array_count_values()` позволяет подсчитать количество повторов значений в массиве:


```
$array = array("denis", "hello", 1, 1, "hello");
print_r(array_count_values($array));
```

Результат будет следующим:

```
Array
(
    [denis] => 1
    [hello] => 2
    [1] => 2
)
```

11.6.8. Замена в массиве

Функция `array_replace()` заменяет значения элементов первого массива значениями из последующих массивов с такими же ключами:

```
array array_replace ( array &$amp;array , array &$amp;array1 [, array &$amp;array2 [,
array &$amp;... ] ] )
```

Рассмотрим пример работы этой функции:

```
$a = array("audi", "bmw", "mercedes");
$r = array(0 => "zaz");
```

```
$a = array_replace($a, $r);
print_r($a);
```

Нумерация элементов списка, как мы уже знаем, начинается с 0, — следовательно, элементу "audi" был присвоен индекс 0. Второй массив, `$r`, состоит всего из одного элемента с ключом 0 и значением "zaz". При вызове функции `array_replace` в массиве `$a` был найден только один элемент с ключом 0. Его значение было заменено значением единственного элемента массива `$r`:

```
Array
(
    [0] => zaz
    [1] => bmw
    [2] => mercedes
)
```

11.6.9. Поиск в массиве

Функция `array_search()` производит поиск указанного значения в массиве и в случае успеха возвращает его индекс (ключ):

```
mixed array_search ( mixed $needle , array $haystack [, bool $strict ] )
```

Если указанное значение не найдено, функция возвращает `false`. Учитывая, что булево значение `false` соответствует небулевому значению 0, результат работы функции лучше анализировать оператором `===`.

Пример использования функции:

```
$a = array("audi", "bmw", "mercedes");
$search_result = array_search("zaz", $a);

if ($search_result === false)
echo "Запрашиваемый элемент в массиве не найден";
else echo "Ключ запрашиваемого элемента: $search_result";
```

11.6.10. Прогулка по массиву

Функция `array_walk()` позволяет применить пользовательскую функцию к каждому элементу массива. Использование этой функции удобнее, чем вызов вашей функции в цикле `for` или `foreach`. Прототип функции:

```
bool array_walk ( array &$amp;array , callback $funcname [, mixed $userdata ] )
```

Нашей функции будет передано два параметра: значение элемента и ключ элемента. Вы можете указать третий параметр, `$userdata`, который будет передан функции в качестве третьего параметра, если это вам нужно.

Напишем функцию, выводящую переданный ей элемент:

```
function my_print($item, $key) {
echo "$key = $item <br>";
}
```

"Прогуляемся" по массиву:

```
$a = array("audi", "bmw", "mercedes");
array_walk($a, 'my_print');
```

Вывод будет таким:

```
0 = audi
1 = bmw
2 = mercedes
```

Из главы 9 мы узнали, что PHP поддерживает многомерные массивы, т. е. такие, в которых значением элемента массива также является массив. Для "прогулки" по многомерному массиву используется рекурсивная версия функции `array_walk()`:

```
bool array_walk_recursive ( array &$amp;input , callback $funcname [, mixed $userdata ] )
```

Результатом работы функции является либо `true`, либо `false`.

Иногда полезно собирать результаты работы пользовательской функции. Например, мы напишем функцию вычисления квадрата числа, "скормим" ей массив с числами, функция пройдет по каждому элементу и вернет его квадрат. Теперь нам нужно "собрать" результаты для каждого вызова функции. Я догадываюсь, что вы знаете, как реализовать это средствами PHP, но есть способ проще — функция `array_map()`. Лучше продемонстрировать ее работу на примере.

Первым делом напишем функцию `asqr()`:

```
function asqr($n)
{
    return($n * $n);
}
```

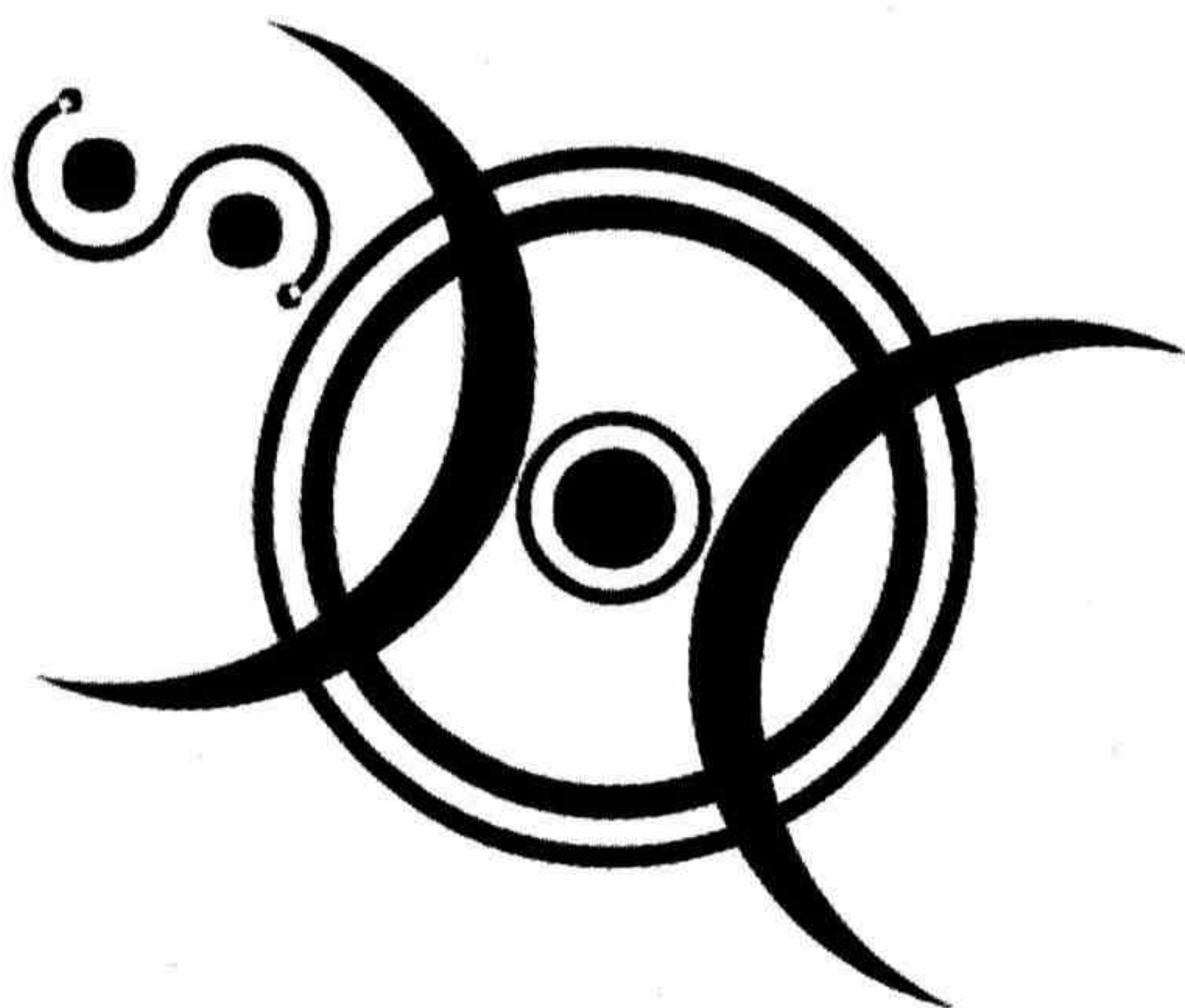
В ней нет ничего сложного. Далее вызовем функцию `array_map()` для массива `$a`, результат запишем в массив `$result`:

```
$a = array(1, 2, 3);
$result = array_map("asqr", $a);
print_r($result);
```

Массив `$result` будет следующим:

```
Array
(
    [0] => 1
    [1] => 4
    [2] => 9
)
```

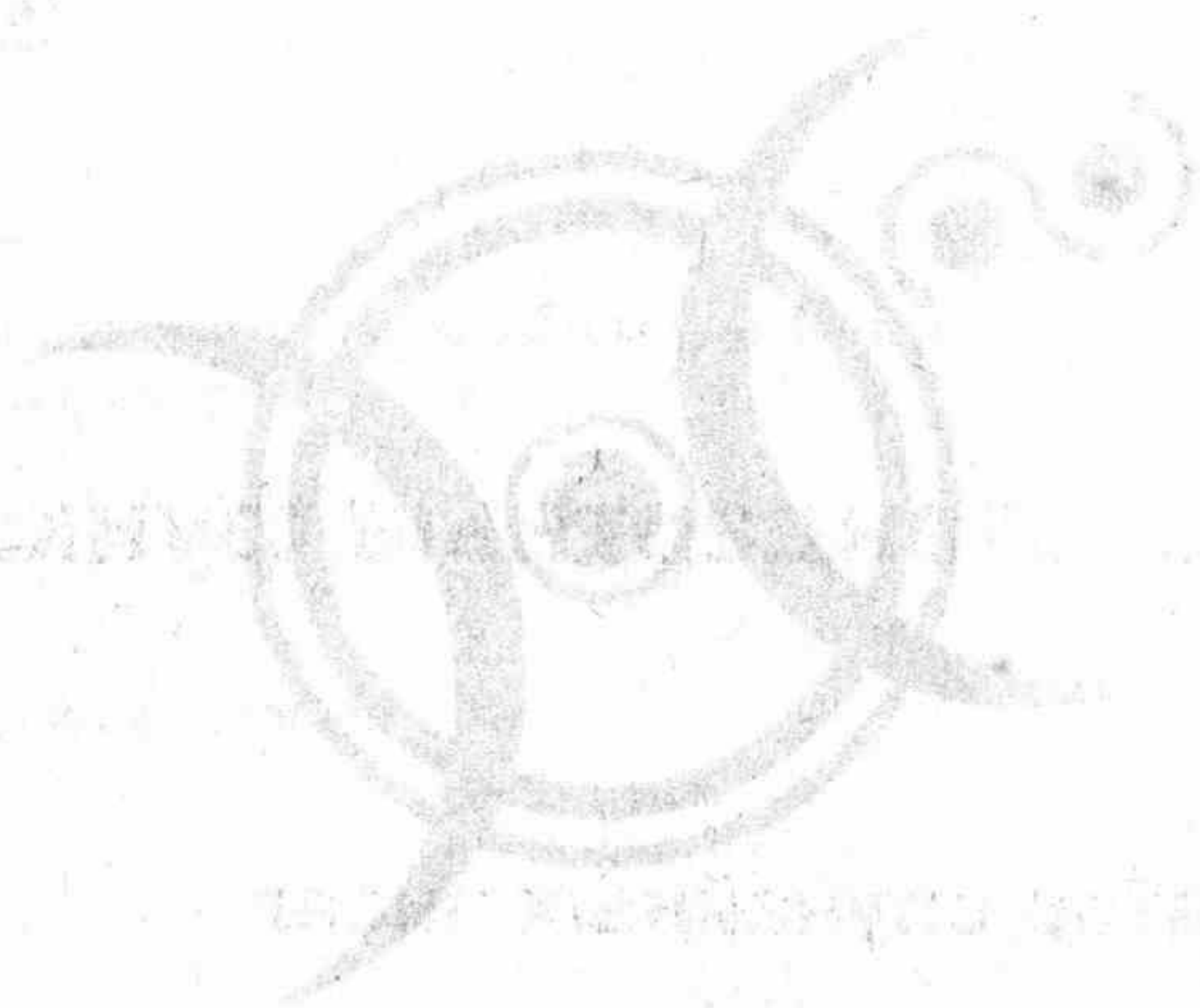
Каждый элемент нового массива является квадратом соответствующего ему элемента старого массива.



I. РАЗДЕЛ 4

Функции в PHP

Глава 12.	Полезные стандартные функции
Глава 13.	Функции для работы со строками
Глава 14.	Работаем с файлами и каталогами
Глава 15.	Вывод графических изображений средствами PHP
Глава 16.	Работа с сетевыми сокетами в PHP. Сетевые функции
Глава 17.	Собственные функции

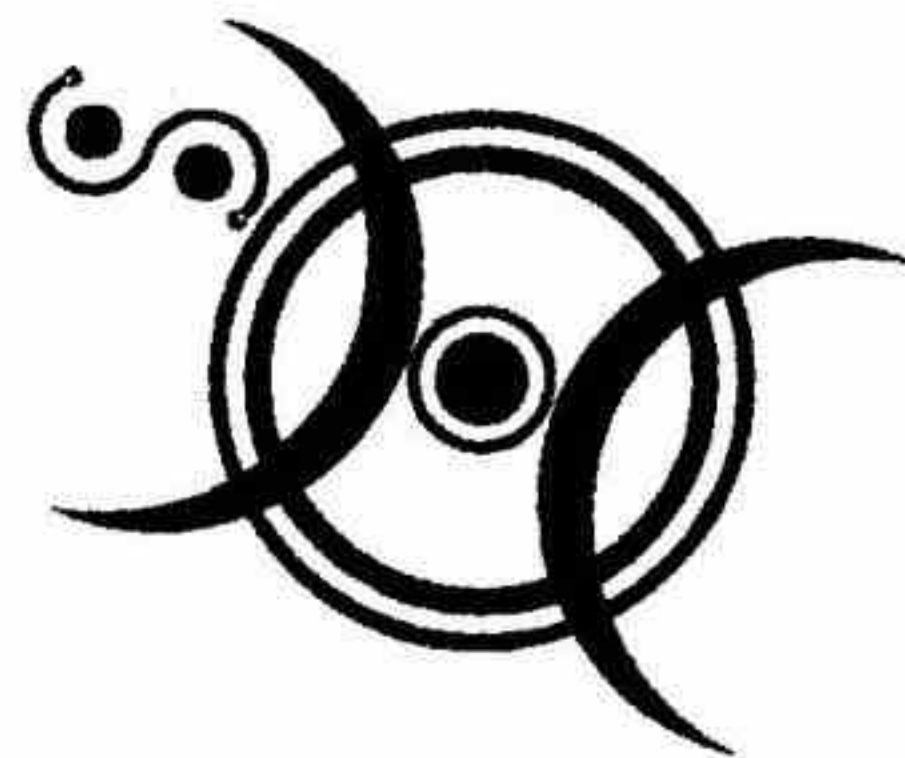


И. ПАВЛОВА

ФУНКЦИИ В РНР

1. Введение	2. Основные понятия
3. Методы исследования	4. Результаты
5. Заключение	6. Литература
7. Приложение	8. Справочные материалы
9. Библиография	10. Заключение
11. Литература	12. Приложение
13. Справочные материалы	14. Заключение
15. Библиография	16. Заключение
17. Литература	18. Приложение
19. Справочные материалы	20. Заключение

ГЛАВА 12



Полезные стандартные функции

12.1. Генератор случайных чисел

В веб-программировании довольно часто используется генератор случайных чисел. Хотите вывести случайную картинку из галереи или фразу дня? А может вам нужно выбрать случайный баннер из рекламной кампании? Все это можно сделать, получив случайный номер, соответствующий отображаемому объекту, а затем отобразив сам объект. Получить случайный номер можно с помощью функции `mt_rand()`:

```
int mt_rand ([int min, int max])
```

Значения `min` и `max` задают диапазон чисел, из которого будет случайно выбрано число. Следующий код генерирует пять случайных чисел:

```
for ($i=0; $i < 5; $i++)  
    echo mt_rand(0, 1000) . " ";
```

Напишем сценарий, выводящий фразу дня (message of the day, сокращенно motd). Создайте два файла, `motd.php` и `motd.txt`. Первый файл — это наш сценарий, код которого представлен в листинге 12.1, а второй — обычный текстовый файл, в котором записано несколько фраз, по одной в каждой строке.

Листинг 12.1. Сценарий `motd.php`

```
<?php  
  
// читаем файл motd.txt в массив  
$motd = file('motd.txt');  
// количество элементов массива  
$max = count($motd);  
  
// получаем случайное число  
$i = mt_rand(0, $max);
```



```
// выводим случайную фразу
echo $motd[$i];

?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *12-1.php*), представленный в каталоге *Glava_12* сопровождающего книгу электронного архива (см. приложение 4).

12.2. Дата и время

12.2.1. Кратко о timestamp

Время в UNIX (следовательно, и в PHP) представляется в формате *timestamp*. *Timestamp* — это количество секунд, прошедших с полуночи 1 января 1970 года. Такой формат не очень понятен пользователям, зато очень удобен для компьютера и самого программиста. Вы можете очень легко сравнивать даты, "переводить" дату на определенное число дней/месяцев/лет — вам нужно всего лишь добавить (или вычесть) определенное количество секунд.

Чтобы вам в дальнейшем было проще оперировать с датами, можно использовать данные табл. 12.1.

Таблица 12.1. Некоторые единицы времени в секундах

Время	Секунды
одна минута	60
один час	3600
один день	86 400
одна неделя	604 800

Получить текущее время (точнее, текущий *timestamp*) можно функцией `time()`:

```
echo time();
```

Если ваш сценарий выполняется больше 1 секунды, желательно сохранить значение *timestamp* в переменной, иначе при следующем вызове `time()` (через 1 секунду) это значение уже изменится:

```
$current_time = time();
```

12.2.2. Функции *strtotime()* и *checkdate()*

Если вы хотите получить *timestamp* даты из прошлого или будущего (а не текущей даты), вам нужно использовать функцию `strtotime()`. В качестве значения этой функции нужно передать строку, описывающую дату. В табл. 12.2 представлены примеры этой функции.

Функция `strtotime()` довольно удобна — вам не надо высчитывать timestamp вручную, а нужно только указать дату или ее смещение. Конечно, если вы не знаете английского, то вам будет полезно запомнить названия дней недели:

- ☐ Monday — понедельник;
- ☐ Tuesday — вторник;
- ☐ Wednesday — среда;
- ☐ Thursday — четверг;
- ☐ Friday — пятница;
- ☐ Saturday — суббота;
- ☐ Sunday — воскресенье.

Если дата указана неправильно, функция вернет `-1`. Первое, что приходит в голову, — использовать эту функцию для проверки даты, введенной пользователем. И если функция вернет `-1`, значит, дата неправильная. Но это очень плохая идея — если пользователь введет `yesterday` или `now`, то для функции это будет корректная дата. Так что для проверки даты лучше воспользоваться функцией `checkdate()`:

```
bool checkdate (int month, int day, int year)
```

Дата считается корректной (функция возвращает `true`), если год находится в диапазоне от 1901 до 32 767, месяц — от 1 до 12, день — в зависимости от выбранного месяца (функция также учитывает високосные годы).

Таблица 12.2. Примеры функции `strtotime()`

Функция <code>strtotime()</code>	Значение timestamp
<code>strtotime("December 31, 2008")</code>	31 декабря 2008 г. (полночь, поскольку время не указано)
<code>strtotime("2008-12-31")</code>	31 декабря 2008 г. (полночь, поскольку время не указано)
<code>strtotime("Friday 12:12")</code>	Пятница, 12:12
<code>strtotime("+1 day")</code>	Следующий день (к текущему timestamp будет добавлено 86 400 секунд)
<code>strtotime("+1 week")</code>	Неделя спустя с текущего момента (к текущему timestamp будет добавлено 604 800 секунд)
<code>strtotime("-3 months")</code>	Три месяца назад (с текущего момента)
<code>strtotime("next Monday")</code>	Следующий понедельник
<code>strtotime("last Monday")</code>	Прошлый понедельник
<code>now</code>	Текущая дата

12.2.3. Вывод даты

Теперь рассмотрим функцию `date()`, выводящую дату, отформатированную в соответствии с заданной строкой формата:

```
string date (string format [, int timestamp])
```


Первый параметр — это формат даты, второй — timestamp. Если второй параметр не задан, выводится текущее время. Например:

```
$d = 1216790861;
echo date("d.m.Y", $d);           // выведет 23.07.2008
```

В строке формата вы можете использовать любые символы, кроме символов модификаторов формата (в данном случае: d, m, Y). Если вам нужно вывести символ, который является модификатором формата, его надо экранировать с помощью слэша — например: \d\a\y. Примеры:

```
echo date("Сегодня d.m.Y");
echo date("\d\a\y d ");
```

Модификаторы формата функции date() представлены в табл. 12.3.

Таблица 12.3. Модификаторы формата функции date()

Модификатор	Описание	Диапазоны/примеры возвращаемых значений
g	Час, 12-часовой формат, без лидирующего нуля	1–12
G	Час, 24-часовой формат, без лидирующего нуля	0–23
h	Час, 12-часовой формат, с лидирующим нулем	01–12
H	Час, 24-часовой формат, с лидирующим нулем	00–23
a	До/после полудня (нижний регистр)	am или pm
A	До/после полудня (верхний регистр)	AM или PM
i	Минуты с лидирующим нулем	00–59
s	Секунды с лидирующим нулем	00–59
d	День месяца, две цифры с лидирующим нулем	01
D	Текстовое представление дня (трехбуквенная аббревиатура)	Mon
j	День месяца (1 или 2 цифры)	1
l	Название дня недели (полное)	Monday
w	Номер дня недели	0 (Вс) до 6 (Сб)
z	Номер дня года	0–365
W	Номер недели в году	42
F	Полное название месяца	December
M	Короткое название месяца (трехбуквенное)	Dec
m	Номер месяца с лидирующим нулем	01
n	Номер месяца без лидирующего нуля	1
t	Количество дней в текущем (или заданном, если указан timestamp) месяце	от 28 до 31
L	Високосный ли год	1, если год високосный, 0 — в противном случае

Таблица 12.3 (окончание)

Модификатор	Описание	Диапазоны/примеры возвращаемых значений
Y	Год, четыре разряда	1999 или 2008
y	Год, два разряда	99 или 08

12.3. Математические функции

Стихия PHP — обработка строк и работа с базами данных, но никак не математические вычисления. Я не утверждаю, что PHP не пригоден для математики, наоборот, у него есть все необходимые (стандартные) математические функции (табл. 12.4), чтобы вам не пришлось изобретать велосипед заново. Однако будете ли вы ими пользоваться? Решать вам!

Таблица 12.4. Некоторые математические функции PHP

Функция	Описание
<code>acos(float \$x)</code>	Арккосинус x
<code>asin(float \$x)</code>	Возвращает арксинус x
<code>atan(float \$x)</code>	Арктангенс аргумента x
<code>cos(float \$x)</code>	Возвращает косинус x
<code>exp(float \$x)</code>	Возвращает экспоненту (2,71828182845) в степени x
<code>log(float \$x)</code>	Возвращает натуральный логарифм x
<code>pi()</code>	Возвращает число π
<code>pow(float \$x, \$a)</code>	Возвращает x в степени a
<code>sin(float \$x)</code>	Возвращает синус аргумента x
<code>sqrt(float \$x)</code>	Возвращает квадратный корень x . Напомню, что x должен быть положительным
<code>tan(float \$x)</code>	Тангенс x

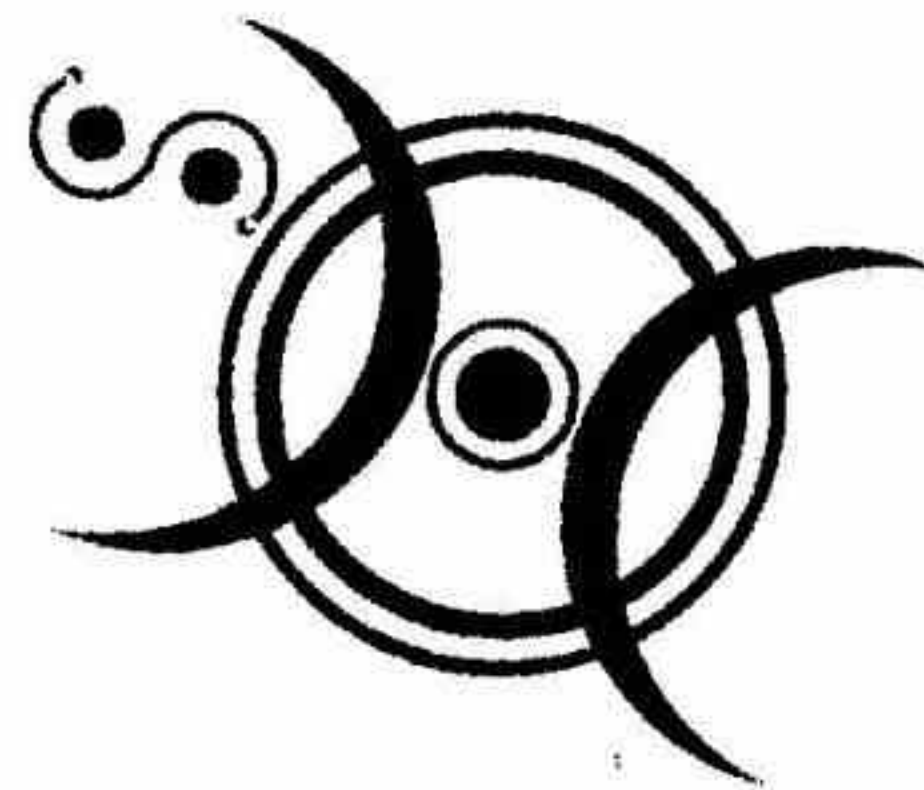
Все функции, представленные в табл. 12.4, относятся к расширению Math. Просмотреть список всех функций этого расширения можно по ссылке: <http://www.php.ru/manual/ref.math.html> (страница руководства на русском языке).

Но это не единственное математическое расширение PHP. Есть еще статистические функции: <http://www.php.ru/manual/ref.stats.html>.

Также есть расширение GMP, позволяющее производить вычисления с повышенной точностью: <http://www.php.ru/manual/ref.gmp.html>.

Для вычислений с произвольной точностью используются функции расширения BC Math: <http://www.php.ru/manual/ref.bc.html>.

ГЛАВА 13



Функции для работы со строками

13.1. Основные строковые функции

В *главе 3* мы рассмотрели основные операции над строками. Сейчас мы поговорим о функциях обработки строк.

К базовым строковым функциям относят следующие функции:

- ❑ `strlen()` — возвращает длину строки;
- ❑ `strpos()` — позволяет найти подстроку в строке;
- ❑ `strstr()` — возвращает первое вхождение подстроки в строку;
- ❑ `strchr()` — является псевдонимом для функции `strstr()`;
- ❑ `substr()` — возвращает подстроку;
- ❑ `strcmp()` — сравнивает две строки;
- ❑ `strncmp()` — сравнивает первые *n* символов строки;
- ❑ `strncasecmp()` — сравнивает первые *n* символов без учета регистра символов;
- ❑ `strcasecmp()` — сравнивает две строки без учета регистра символов;
- ❑ `strnatcmp()` — сравнивает две строки, используя алгоритм натуральной сортировки;
- ❑ `strnatcasecmp()` — то же, что и `strnatcmp()`, но без учета регистра символов;
- ❑ `str_repeat()` — используется для повтора строки;
- ❑ `str_replace()` — производит замену в строке;
- ❑ `strspn()` — определяет наличие начальных символов в строке;
- ❑ `strcspn()` — определяет отсутствие начальных символов в строке.

Рассмотрим все эти функции по порядку.

Первой функции нужно передать всего лишь один параметр — строку:

```
$str = "string";  
echo strlen($str);    // выведет 6
```


Функции `strpos()` нужно передать три параметра: основную строку, искомую подстроку и смещение — номер символа основной строки, с которого следует начинать поиск подстроки:

```
int strpos (string $haystack, string $needle [, int offset])
```

Если подстрока не найдена, функция возвращает `false`, а если найдена, то первую позицию вхождения подстроки в строку. Пример использования функции:

```
$pos = strpos($my_string, $find);

if ($pos === false) {
    echo "Строка '$find' не найдена в строке '$my_string'";
} else {
    echo "Строка '$find' найдена в строке '$my_string' в позиции $pos";
}
```

Функция `strpos()` чувствительна к регистру символов. Если регистр символов для вас не важен, лучше использовать функцию `stripos()`, которая является полным аналогом `strpos()` за исключением того, что она игнорирует регистр символов.

Функция `strrpos()` находит позицию последнего вхождения подстроки `$needle` в строку `$haystack`. Функция `strripos()` делает то же самое, но без учета регистра:

```
int strrpos(string $haystack, string $needle [, int $offset = 0 ])
int strripos(string $haystack, string $needle [, int $offset = 0 ])
```

Функция `strstr()` предназначена для поиска первого вхождения подстроки `needle` в строку `$haystack`:

```
string strstr (string $haystack, mixed $needle [, bool $before_needle = false ])
```

Если третий параметр равен `true`, то функция возвращает часть строки `$haystack`, находящуюся до строки `$needle`. Третий параметр у функции `strstr()` появился в PHP версии 5.3.0. Рассмотрим небольшой пример:

```
$email = 'den@dkws.org.ua';
$domain = strstr($email, '@');
echo $domain; // выведет @dkws.org.ua
```

```
// только в PHP 5.3.0 или выше
$user = strstr($email, '@', true);
echo $user; // выведет den
```

Функция `strstr()` является регистрозависимой. Если вам нужно игнорировать регистр символов, тогда используйте функцию `stristr()`, параметры у нее такие же, как у функции `strstr()`. Обе функции возвращают `false`, если подстрока `$needle` не найдена.

Функции `strstr()` и `stristr()` удобно использовать для определения наличия в строке какого-нибудь образца, например:

```
// если в строке $v есть подстрока "UNION",
// прекратить работу сценария:
if (stristr($v, "UNION") !== false) die();
```


Функция `substr()` возвращает подстроку указанной строки `string` длиной `length` начиная с позиции `start`:

```
string substr (string $string, int $start [, int $length])
```

Если последний параметр (длина) не указан, то возвращается подстрока, начиная с позиции `start` и до конца строки. Пример:

```
$sub = substr("123456", 2); // возвращает "3456"
```

Для сравнения строк служит функция `strcmp()`:

```
int strcmp (string $str1, string $str2)
```

Функции нужно передать две сравниваемые строки. Функция возвращает следующие значения:

- ❑ 0 — строки равны;
- ❑ 1 — первая строка "больше" (лексикографически), чем вторая;
- ❑ -1 — первая строка "меньше", чем вторая.

Функция `strcmp()` учитывает регистр символов. Если вам нужно сравнивать строки без учета регистра, используйте функцию `strcasecmp()`. Ее параметры и возвращаемые значения такие же, как у `strcmp()`.

Функция `strncmp()` сравнивает первые *n* символов строки (учитывается регистр символов):

```
int strncmp (string $str1 , string $str2 , int $len)
```

Первые два параметра — как у обычной `strcmp()`, второй — те самые *n* символов. Возвращаемые значения аналогичны функции `strcmp()`.

Аналог функции `strncmp()`, но без учета регистра символов — `strncasecmp()`.

В *главе 10*, рассматривая функции для сортировки массивов, мы познакомились с алгоритмом натуральной сортировки. Функции `strnatcmp()` и `strnatcasecmp()` сравнивают строки, используя этот алгоритм:

```
int strnatcmp (string $str1 , string $str2)
int strnatcasecmp (string $str1 , string $str2)
```

Указанные функции возвращают те же значения, что и функция `strcmp()`. Функция `strnatcasecmp()` игнорирует регистр символов.

Функция `str_repeat()` возвращает *n* повторов указанной строки:

```
string str_repeat (string $input, int $n)
```

Например:

```
echo str_repeat("1", 3); // выведет 111
```

Осуществить замену в строке можно с помощью функции `str_replace()`:

```
mixed str_replace (mixed $search, mixed $replace, mixed $subject [, int &count])
```

Эта функция производит замену подстроки `search` строкой `replace` в строке `subject`. Последний параметр задает количество замен, которые должна произвести

функция. Обратите внимание, что данный параметр доступен, начиная с PHP версии 5, поэтому если у вас до сих пор PHP 4, то самое время обновиться!

Заметьте, что функция `str_repeat()` не изменяет исходную строку `subject`, а работает с ее копией, а потом возвращает результат.

Функция `str_ireplace()` аналогична функции `str_replace()`, но игнорирует регистр символов.

Теперь рассмотрим функции `strspn()` и `strcspn()`:

```
int strspn(string str1, string str2)
int strcspn(string str1, string str2)
```

Первая функция определяет наличие начальных символов в строке. Возвращает длину начального фрагмента строки `str1`, который полностью состоит из строки `str2`. Например:

```
$str = "function";
$substr_count = strspn($str, "func");
echo ($substr_count);           // выведет 4
```

Функция `strcspn()` возвращает длину начального фрагмента строки `str1`, который состоит полностью не из символов, которые есть в `str2`.

13.2. Специальные функции замены

Следующие функции удобно использовать при обработке данных, полученных от пользователя:

- ❑ `nl2br()` — заменяет все символы новой строки (`\n`) HTML-тегом `<br>`;
- ❑ `htmlspecialchars()` — заменяет специальные символы (вроде "больше", "меньше") их HTML-эквивалентами.

Функциям нужно передать только строку, в которой необходимо произвести замену, например:

```
$string = "Первая строка\nВторая строка";
echo nl2br($string);
```

```
$string2 = "5 < 10 > 2";
$output = htmlspecialchars($string2);
```

Функции не изменяют исходную строку, а работают с ее копией. Для обратного преобразования строки, обработанной функцией `htmlspecialchars()`, служит функция `htmlspecialchars_decode()`:

```
string htmlspecialchars_decode (string $string [, int $quote_style = ENT_COMPAT ])
```

В PHP 5.4 функцию `htmlspecialchars()` существенно улучшили. Старая версия была немного "неинтуитивной". Например, если вы передавали функции неправильно закодированную строку, то функция возвращала просто пустую строку. Самое интересное, она выдавала ошибку, но только в том случае, если отображение ошибок было отключено. Да, именно отключено. Вроде бы логично — несмотря на то, что

отображение ошибок отключено, мы все равно выводим сообщение об ошибке. Но на практике это означало, что на компьютере разработчика вы не видели ошибку (т. к. на нем включается, как правило, отображение всех ошибок), но зато ошибки "вылезали" при переносе проекта на реальный сервер.

В PHP 5.4 неправильное поведение функции исправлено. К тому же добавлено два аргумента, позволяющих выбрать альтернативу возвращаемой строке (для большей информативности):

- ❑ `ENT_IGNORE` — отбрасывает всю некорректную последовательность. Однако в этом случае вы не увидите недопустимых случаев, кроме того, это создает угрозу безопасности;
- ❑ `ENT_SUBSTITUTE` — вместо удаления недопустимых символов они будут заменены на символ Юникода `◆` (`U + FFFD`).

Пример:

```
htmlspecialchars($str, ENT_SUBSTITUTE);
```

Функция `htmlentities()` служит для преобразования всех символов в HTML-эквиваленты:

```
string htmlentities (string $string [, int $flags = ENT_COMPAT [, string $charset [, bool $double_encode = true ]]])
```

Первый параметр функции — это сама строка, второй параметр может принимать следующие значения:

- ❑ `ENT_COMPAT` — конвертирует двойные кавычки и игнорирует одинарные;
- ❑ `ENT_QUOTES` — конвертирует и двойные, и одинарные кавычки;
- ❑ `ENT_NOQUOTES` — игнорирует кавычки;
- ❑ `ENT_IGNORE` — опция предназначена для обеспечения обратной совместимости со старыми версиями PHP — применять ее не следует, поскольку она негативно влияет на безопасность вашего приложения.

Третий параметр — кодировка символов. Используйте следующие значения:

- ❑ `"UTF-8"` — для кодировки UTF-8;
- ❑ `"cp1251"`, `"windows-1251"` или просто `"1251"` — для кодировки Windows-1251;
- ❑ `"KOI8-R"` — для кодировки KOI8-R;
- ❑ `"cp866"` — для кириллической DOS-кодировки.

Когда четвертый параметр выключен (значение `false`), PHP не будет кодировать существующие в строке HTML-эквиваленты. По умолчанию разрешается двойное преобразование.

Функция `html_entity_decode()` служит для обратного преобразования строки, закодированной с помощью функции `htmlentities()`, в первоначальный вид:

```
string html_entity_decode (string $string [, int $quote_style = ENT_COMPAT [, string $charset ]])
```

Назначение параметров у этой функции такое же, как и у `htmlentities()`.

13.3. Преобразование строки

К этой группе относятся следующие функции:

- ❑ `strip_tags()` — удаляет HTML-теги из строки;
- ❑ `stripslashes()` — удаляет слэши из строки;
- ❑ `stripslashes()` — удаляет из строки слэши, добавленные функцией `addslashes()`;
- ❑ `strtolower()` — переводит символы строки в нижний регистр;
- ❑ `strtoupper()` — переводит символы строки в верхний регистр;
- ❑ `strrev()` — переворачивает строку;
- ❑ `trim()` — удаляет пробельные символы до и после строки;
- ❑ `ltrim()` — удаляет пробельные символы в начале строки;
- ❑ `lcfirst()` — делает первый символ строки строчным (переводит в нижний регистр);
- ❑ `rtrim()` — удаляет пробельные символы в конце строки;
- ❑ `chop()` — псевдоним для `rtrim()`;
- ❑ `str_shuffle()` — перемешивает символы строки в случайном порядке;
- ❑ `str_rot13()` — выполняет преобразование ROT13 над строкой;
- ❑ `urlencode()` — кодирует строку для ее передачи в URL;
- ❑ `urldecode()` — раскодирует строку, закодированную функцией `urlencode()`;
- ❑ `ucfirst()` — делает прописным (переводит в верхний регистр) первый символ заданной строки;
- ❑ `ucwords()` — делает заглавным первый символ каждого слова в строке;
- ❑ `wordwrap()` — разбивает текст согласно указанным параметрам.

Конечно, это далеко не все строковые функции, которые есть в арсенале PHP, но зато самые полезные, которые можно использовать в самых разнообразных ситуациях.

Функция `strip_tags()` удаляет из текста HTML-теги, оставляя обычный текст. Если передать функции HTML-страницу, то она вернет обычный текст без форматирования. К тому же эта функция с успехом защищает ваш сайт от вредоносного кода, который может передаваться в HTML-тегах (например, в тегах `OBJECT`, `SCRIPT`). Злоумышленник запросто может передать такой код, например, в качестве текста сообщения в гостевой книге. Функция `strip_tags()` позволяет избавиться от всех тегов. Хотя, если вы хотите оставить некоторые безобидные теги, например `BR`, то можете указать их в качестве второго параметра функции:

```
$html = "<h1>Заголовок<h1><p>Текст<br>Следующая строка";
```

```
// $txt1 = "Заголовок<p>Текст<br>Следующая строка"
```

```
$txt1 = strip_tags($html, "<p><br>");
```



```
// $txt2 = "ЗаголовокТекстСледующая строка"
$txt2 = strip_tags($html);
```

Функция `stripslashes()` проста в использовании — ей нужно передать всего один параметр — строку, из которой требуется удалить слэши:

```
string stripslashes (string $str)
```

Есть функции и обратные этой: `addslashes()` и `addcslashes()`. Вторая экранирует строку слэшами, а первая — слэшами в стиле C:

```
string addslashes (string $str, string $charlist)
string addslashes (string $str)
```

Первый параметр — это экранируемая строка. Параметр `charlist` у `addcslashes` задает символы, которые нужно экранировать, — будьте осторожны при задании этого диапазона.

Функции `strtolower()` и `strtoupper()` подробно рассматривать не будем — они слишком просты. Все что здесь нужно — это передать функции строку, регистр которой надо изменить. Как и остальные строковые функции PHP, эти функции не изменяют исходную строку. Если вам нужно изменить исходную строку, можно вызвать функцию так:

```
$str = strtolower($str);
```

Функция `strrev()` служит для "переворачивания" строки:

```
string strrev (string $string)
```

Пример:

```
echo strrev("123");           // выведет 321
```

Довольно часто приходится удалять пробелы в данных, оставленных пользователем в форме. Функции `trim()`, `ltrim()` и `chop()` удаляют из строки пробелы и пробельные символы (вроде табуляции). Однако эти функции не панацея. Они помогут избавиться от пробелов, которые находятся в начале и/или в конце строки. Но они не в силах удалить лишние пробелы, находящиеся в середине строки, например:

```
"Иван  Иван"
```

Функция `ucfirst()` делает прописным первый символ переданной ей строки, а функция `ucwords()` делает прописным первый символ каждого слова в строке:

```
string ucfirst (string $str)
string ucwords (string $str)
```

Пример использования:

```
echo ucfirst("hello world");           // выведет Hello world
echo ucwords("hello world");           // выведет Hello World
echo lcfirst("Hello");                 // выведет hello
```

Если вам нужно сгенерировать URL, содержащий специальные символы (или хотя бы просто пробелы) или символы русского языка, вам лучше закодировать строку

функцией `urlencode()`. В этом случае все символы заменяются последовательностью `%код`, а пробелы — символом `+`.

Функция `urldecode()` выполняет обратное преобразование строки.

```
$s = "Изучаем PHP";  
echo urlencode($s);
```

Вывод будет следующим:

```
%C8%E7%F3%F7%E0%E5%EC+PHP
```

Предположим, что у нас есть довольно длинный текст (переменная `$txt`), который для удобства вывода/чтения можно разбить на части с помощью функции `wordwrap()`:

```
// разбиваем текст по 30 символов  
$txt = wordwrap($txt, 30, "<br>");
```

Функции `str_shuffle()` и `str_rot13()` очень просты — им нужно передать всего один параметр — строку, которую требуется преобразовать. Результат — преобразованная строка, исходная строка не изменяется.

Заметьте, что функция `wordwrap()` распознает слова и не разбивает их. Если же вы хотите вставлять указанный разделитель через заданное количество символов, тогда установите последний необязательный параметр в 1:

```
$txt = wordwrap($txt, 30, "<br>", 1);
```

13.4. Функции преобразования кодировок

Для преобразования кодировок русского языка служит функция `convert_cyr_string()`:

```
string convert_cyr_string (string $str, string $from, string $to)  
// преобразование текста из кодировки KOI8-R в Windows-1251:  
echo convert_cyr_string ($string, "k", "w");
```

Первый параметр — это строка, кодировку которой нужно преобразовать. Вторым параметром — исходная кодировка, третий — кодировка назначения. Поддерживаются следующие кодировки:

- `k` — `koi8-r`;
- `w` — `windows-1251`;
- `i` — `iso8859-5`;
- `a` — `x-cp866`;
- `d` — `x-cp866`;
- `m` — `x-mac-cyrillic`.

Функция `convert_uencode()` кодирует строку в формате `uencode`. Функция `convert_udecode()` декодирует строку из формата `uencode` в обычный вид:


```
string convert_uencode (string $data)
string convert_udecode (string $data)
```

13.5. Функции для работы с отдельными символами строки. Разыменовывание строки

Вы можете получить доступ к отдельным символам строки с помощью квадратных скобок, например:

```
$nums = "654321";
echo $nums[1]; // выведет 5
```

Заметьте, что нумерация символов в строке начинается с 0, поэтому в приведенном примере и будет выведена цифра 5.

Функции `chr()` и `ord()` возвращают символ по заданному коду и код по заданному символу соответственно. Например:

```
echo chr(65); // выведет A
echo ord('A'); // выведет 65
```

В PHP 5.5 появилось еще одно сомнительное новшество, позволяющее сделать код компактнее, — разыменовывание строки. В целом разыменовывание строки выглядит так:

```
echo 'Привет'[0]; // Выведет "П"
```

Согласитесь, проще было бы просто вывести "П". Хотя немного это новшество себя оправдывает в функциях. Представим, что есть функция, возвращающая строку:

```
function getSt() {
    return "Привет";
}
```

Нам нужно обратиться к первому символу этой строки (с индексом 0). Если раньше приходилось сначала присвоить результат какой-нибудь строке, а уже потом обращаться к ее символам, то теперь мы можем обратиться в результату напрямую:

```
// Раньше
$temp = getSt();
echo $temp[0];
// В PHP 5.5
echo getSt()[0];
```

Ситуация с разыменовыванием строки такая же, как и с разыменовыванием массива. Из главы 8 вы уже узнали, что у данного способа есть как свои преимущества (хотя и сомнительные), так и недостатки.

13.6. Функция *md5()* и другие функции шифрования/хэширования. API хэширования паролей

Функция `md5()` кодирует заданную строку с помощью алгоритма MD5. Особенность этого алгоритма заключается в том, что закодированную строку нельзя обратно раскодировать. Алгоритм MD5 часто используется для кодирования паролей.

Как же потом проверить, правильный ли пароль ввел пользователь, если зашифрованную строку нельзя расшифровать? Все очень просто. Вам нужно закодировать с помощью `md5()` принятый от пользователя пароль, а затем сравнить две закодированные строки, например:

```
$pass = "5ebe2294ecd0e0f08eab7690d2a6ee69";

// получаем пароль от пользователя и кодируем его:
$password = $_POST['password'];
$password = md5($password);

if ($pass === $password) echo "Пароль правильный";
    else echo "Пароль неправильный";
```

Функция `md5_file()` возвращает MD5-хэш указанного файла:

```
string md5_file (string $filename [, bool $raw_output = false ])
```

Вам не нужно читать файл в строку для вычисления его MD5-хэша.

Функции `sha1()` и `sha1_file()` аналогичны функциям `md5()` и `md5_file()`, только вместо алгоритма MD5 используют алгоритм SHA1 (US Secure Hash Algorithm 1).

Функция `crc32()` вычисляет контрольную сумму CRC32 для указанной строки `$str`:

```
int crc32 (string $str)
```

Функция `crypt()` возвращает хэш указанной строки. Функция более универсальна, чем `md5()`, поскольку позволяет использовать различные алгоритмы шифрования: DES, Blowfish, MD5, SHA1:

```
string crypt (string $str [, string $salt ])
```

Подробное описание функции `crypt()` вы найдете по адресу: <http://ua2.php.net/manual/en/function.crypt.php>.

В PHP 5.5 появился интерфейс API хэширования паролей (Password Hashing API). Предполагается, что применение этого API облегчит и стандартизирует хэширование паролей, и вместо множества разных функций программист станет использовать теперь один API. На мой взгляд, функционала у этого API пока маловато — по крайней мере, на данный момент. Надеюсь, скоро все изменится к лучшему, а пока рассмотрим то, что есть. Дополнительную информацию об этом API можно получить по адресу: <http://php.net/password>.

Основная функция API — `password_verify()`. Ей нужно передать пароль и хэш, а она проверит, соответствует ли пароль заданному хэшу. Определение функции следующее:

```
boolean password_verify ( string $password, string $hash )
```

Хэш можно создать функцией `password_hash()` из этого API или же функцией `crypt()`. Функция `password_verify()` возвращает `true`, если пароль и хэш соответствуют друг другу, иначе — `false`.

Пример использования функции `password_verify()`:

```
// Хэш пароля.
$hash = ' $2y$10$hgWxMcltA0SYpCe4zm.H0ew3otg8HCezeTTlCIvuoblVXl2JKFcce';

if (password_verify('1234567', $hash)) {
    echo 'Правильный пароль!';
} else {
    echo 'Пароль неправильный!';
}
```

А теперь рассмотрим функцию `password_hash()`:

```
string password_hash ( string $password, integer $algo [, array $options ] )
```

Первый параметр — это пароль, хэш которого нужно сгенерировать, второй параметр — алгоритм шифрования. Третий параметр необязательный — он задает массив с опциями. В настоящее время поддерживаются две опции:

- ❑ `salt` — "соль", служащая при хэшировании пароля;
- ❑ `cost` — алгоритмическая стоимость вычисления пароля.

Пока поддерживается только алгоритм Blowfish, второй параметр может принимать одно из значений:

- ❑ `PASSWORD_BCRYPT (integer) = 1`
- ❑ `PASSWORD_DEFAULT (integer) = PASSWORD_BCRYPT`

Первая константа соответствует алгоритму Blowfish, а вторая задает алгоритм по умолчанию, который в данный момент является тоже алгоритмом Blowfish. В следующих версиях PHP эта константа будет переопределена, и появятся другие алгоритмы шифрования.

Как видите, возможностей у нового API пока мало, но, полагаю, уже в ближайшее время мы увидим его развитие.

13.7. Функция `explode()`: выделение подстрок

При работе с файлами вы часто будете сталкиваться со строками с разделителями. Например, формат CSV для разделения полей строки использует двоеточие:

Поле1:Поле2:Поле3

Функция `explode()` читает строку с разделителями, разбивает ее на подстроки и помещает их в массив, каждый элемент массива — это подстрока. Для примера рассмотрим следующий сценарий:

```
$str = "user:password:/home/user";
```

```
$info = explode(":", $str);
```

```
print_r($info);
```

Вывод будет следующим:

```
Array
(
    [0] => user
    [1] => password
    [2] => /home/user
)
```

У функции `explode()` есть еще и третий параметр, ограничивающий количество полей в просматриваемой строке. В этом случае функция прочитает только указанное нами количество полей.

Функция `implode()` служит для объединения элементов массивов в строку:

```
string implode (string $glue , array $pieces)
```

Первый параметр — это разделитель, которым будут разделены элементы массива при их помещении в строку. В строку помещаются только значения элементов массива, но не ключи. Функцию `implode()` удобно использовать для чтения всего файла в одну строку:

```
$txt = implode('', file('article.txt'));
```

Функция `join()` является псевдонимом для функции `implode()`.

13.8. Статистические функции

Функция `count_chars()` возвращает количество символов в строке:

```
mixed count_chars (string $string [, int $mode = 0 ])
```

Возвращаемое значение зависит от второго параметра (`$mode`):

- ❑ 0 — массив элементов, ключами которого являются коды символов, значением каждого элемента является количество данного символа в строке;
- ❑ 1 — то же, что и 0, но в массив включаются символы с частотой больше 0;
- ❑ 2 — то же, что и 0, но в массиве представлены символы с частотой 1;
- ❑ 3 — строка, содержащая все уникальные символы;
- ❑ 4 — строка, содержащая все неиспользуемые символы.

Рассмотрим небольшой пример:

```
$str= "Hello!";

foreach (count_chars($str, 1) as $i => $val) {
    echo "Символ " . chr($i) . " встречается в строке $val раз(a) <br>";
}
```

Вывод сценария будет следующим:

```
Символ ! встречается в строке 1 раз(a)
Символ H встречается в строке 1 раз(a)
Символ e встречается в строке 1 раз(a)
Символ l встречается в строке 2 раз(a)
Символ o встречается в строке 1 раз(a)
```

Функция `str_word_count()` возвращает количество слов в строке:

```
$str = "Hello, world!";
echo str_word_count($str);
```

Функция `substr_count()` подсчитывает, сколько раз подстрока `$needle` встречается в подстроке `$haystack`:

```
int substr_count (string $haystack , string $needle [, int $offset = 0 [, int $length ]])
```

13.9. Функции вывода текста

Для вывода текста мы привыкли применять конструкцию `echo`. Если вам не нравится `echo` в том виде, как используем его мы, вы можете вызывать `echo` как функцию — со скобками, внутри которых будут параметры, которые и отобразит функция `echo()`:

```
void echo (string $arg1 [, string $... ])
```

Программистам, знакомым с C, понравится функция `print()`, которая хоть как-то напоминает их любимый язык программирования:

```
int print (string $arg)
```

По большому счету, `print` тоже не является функцией в прямом смысле слова — это конструкция языка программирования (оператор), как и `echo`.

Рассмотрим несколько примеров вывода строки "Hello":

```
echo "Hello";
echo("Hello");
print "Hello";
print("Hello");
```

Все эти варианты равноправны и выведут нашу строку без каких-либо нюансов.

Совсем другое дело — функция `sprintf()`, которая возвращает аргументы согласно переданной ей строке формата. Другими словами, `sprintf()` возвращает отформатированную строку. Эта функция похожа на аналогичную функцию в языке C:

```
string sprintf (string $format [, mixed $args [, mixed $...]])
```

ПРИМЕЧАНИЕ

Обратите внимание — функция `sprintf()` именно возвращает строку, а не выводит ее в браузер. Вы должны получить отформатированную строку (результат работы функции `sprintf()`) и вывести ее в браузер самостоятельно функцией `echo()` или `print()`.

Строка формата (первый аргумент, `$format`) задает формат для каждого аргумента, переданного после аргумента `$format`. Строка формата состоит из 0 или более модификаторов формата. Каждый модификатор формата предваряется символом `%`. В табл. 13.1 приводятся модификаторы формата для функции `sprintf()`.

Таблица 13.1. Модификаторы формата для `sprintf()`

Модификатор	Описание
%	Используется для вывода символа '%'
b	Аргумент считается целым числом, выводится в двоичном виде
c	Аргумент считается целым числом, выводится в виде символа с соответствующим ASCII-кодом
d	Аргумент считается числом, выводится в виде десятичного числа со знаком
e	Аргумент считается числом с плавающей запятой (float) и выводится в научной нотации (например, 1.3e+3)
u	Целое беззначное число в десятичном виде
f	Аргумент считается числом с плавающей запятой (float) в десятичном виде с плавающей запятой
o	Целое число, выводится в восьмеричной системе
x	Число в шестнадцатеричном виде, все символы строчные
X	Число в шестнадцатеричном виде, все символы прописные

Пример:

```
$n = 100;
// выводим число 100 в десятичном, шестнадцатеричном и двоичном виде
echo sprintf("Dec: %d Hex: %X Bin: %b", $n, $n, $n);
```

Функцию `sprintf()` удобно использовать, если требуется присвоить результат ее работы переменной для последующей обработки. Если же нужно просто вывести отформатированную строку, следует воспользоваться функцией `printf()`:

```
int printf (string $format [, mixed $args [, mixed $...]])
```


Параметры у функции `printf()` — такие же, как у `sprintf()`, разница заключается в том, что `printf()` сразу выводит строку в браузер. Дополнительную информацию о модификаторах форматов функций `printf()` и `sprintf()` можно получить по адресу: <http://ua2.php.net/manual/en/function.sprintf.php>.

Функция `fprintf()` выводит отформатированную строку в поток (`stream`), заданный дескриптором `$handle`:

```
int fprintf (resource $handle, string $format [, mixed $args [, mixed $...]])
```

Пример:

```
if (!($f = fopen('numbers.txt', 'w')))  
die();  
$n = 555;  
fprintf($f, "%X %b", $n, $n);
```

Кроме функции `printf()` в PHP есть подобная ей функция — `vprintf()`, которая действует аналогично функции `printf()`, но вместо списка аргументов принимает массив аргументов:

```
int vprintf (string $format, array $args)
```

Массиво-ориентированный аналог есть и для функции `sprintf()` — такая функция называется `vsprintf()`:

```
string vsprintf (string $format, array $args)
```

Функция `vfprintf()` выводит отформатированную строку в поток (`stream`), заданный дескриптором `$handle`:

```
int vfprintf (resource $handle, string $format, array $args)
```

13.10. Установка локали

Многие строковые функции для своей правильной работы требуют точной установки локали. Для установки локали служит функция `setlocale()`:

```
string setlocale (int $category, string $locale [, string $...])
```

Первый параметр — это категория объектов, к которым должны быть применены настройки локали:

- ☐ `LC_ALL` — для всего нижеперечисленного;
- ☐ `LC_COLLATE` — для сравнения строк;
- ☐ `LC_CTYPE` — для классификации и преобразования строк — например, для функций вроде `strtoupper()`;
- ☐ `LC_MONETARY` — для `localeconv()`;
- ☐ `LC_NUMERIC` — для десятичного разделителя;
- ☐ `LC_TIME` — для форматирования даты и времени функцией `strftime()`;

□ `LC_MESSAGES` — для ответов системы (доступно, если PHP откомпилирован с опцией `libintl`).

Второй параметр — это сама локаль. Для русского языка применяется локаль `ru_RU`.

В наши дни кодировка UTF-8 практически стала стандартом, поэтому желательно настроить локаль на использование этой кодировки. Делается это так:

```
setlocale(LC_ALL, "ru_RU.UTF-8");
```

К сожалению, хотя этот вызов будет прекрасно работать на UNIX (чего в большинстве случаев будет достаточно — ведь все серьезные проекты помещаются на UNIX-хостинг), но в Windows такой трюк не сработает.

13.11. Форматирование чисел и денежных величин

Для форматирования денежных величин служит функция `money_format()`. Для ее правильной работы необходима точная установка локали. Получить подробную информацию об этой функции можно по адресу:

<http://ua2.php.net/manual/en/function.money-format.php>.

Функция `number_format()` служит для форматирования чисел:

```
string number_format (float $number [, int $decimals = 0 ])
```

Первый параметр — это число, второй — количество знаков после запятой для вещественных чисел:

```
$n = 1000000;  
echo number_format($n, 0);           // выведет 1,000,000
```

Полный прототип функции выглядит так:

```
string number_format (float $number, int $decimals = 0, string $dec_point =  
'.', string $thousands_sep = ',')
```

Третий параметр — это символ, разделяющий целую и дробную часть числа (по умолчанию — точка), четвертый параметр — это разделитель тысяч (по умолчанию — запятая).

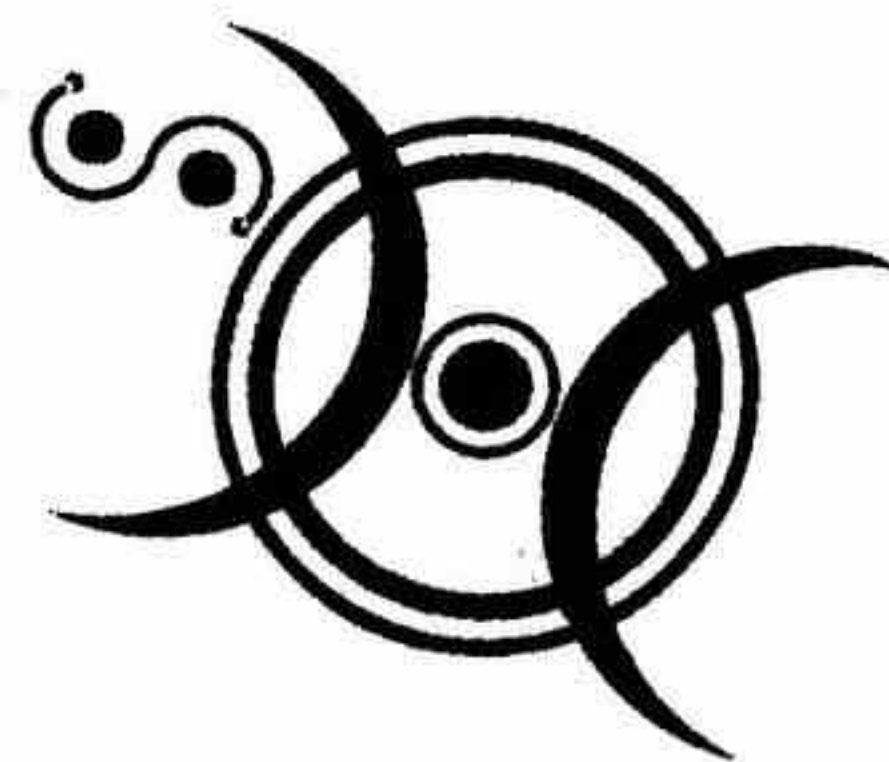
13.12. Преобразование систем счисления

Функция `bin2hex()` выполняет преобразование из двоичной системы в шестнадцатеричную, а функция `hex2bin()` выполняет обратное преобразование — из шестнадцатеричной в двоичную.

Прототипы функций следующие:

```
string bin2hex (string $str)  
string hex2bin (string $data)
```


ГЛАВА 14



Работаем с файлами и каталогами

14.1. Права доступа в UNIX

Прежде чем приступать к изучению функций для работы с файлами и каталогами, мы должны рассмотреть права доступа к файлам и каталогам. Если вы работаете в данный момент в Windows, то можете пропустить этот раздел, но вы обязательно вернетесь к нему, когда станете переносить свои сценарии на сервер хостинг-провайдера (который обычно работает под управлением одного из вариантов UNIX) и столкнетесь с проблемой доступа к файлам.

В UNIX у каждого файла (или каталога, он по сути тоже является файлом) есть свой владелец — пользователь, создавший файл. Для каждого каталога и файла вы можете задать права доступа. Точнее, права доступа автоматически задаются при создании каталога (файла), а вы, при необходимости, можете их изменить. Чуть позже мы поговорим об этой необходимости с точки зрения PHP.

Существуют три права доступа: чтение (r), запись (w), выполнение (x). Для каталога право на выполнение означает право на просмотр содержимого каталога.

Вы можете установить разные права доступа для владельца (т. е. для себя), для группы владельца (т. е. для всех пользователей, входящих в одну с владельцем группу) и для прочих пользователей. Пользователь root может получить доступ к любому файлу (каталогу) вне зависимости от прав, которые вы установили.

При выводе содержимого каталога на UNIX-сервере права доступа к файлу выводятся примерно так:

```
-r--r----- 1 den group 300 Apr 11 11:11 video.txt
```

В этой записи последовательность символов `-r--r-----` обозначает права доступа. Первый дефис означает, что перед нами обычный файл. В случае каталога на его месте стояла бы буква `d`. Следующие три символа (`r--`) определяют права доступа владельца. Первый символ — это чтение, второй — запись, третий — выполнение. Как можно видеть, в данном случае владельцу разрешено только чтение этого файла, запись и выполнение запрещены, поскольку в правах доступа режимы `w` и `x` не определены.

Следующие три символа (r--) задают права доступа для членов группы владельца. Права здесь такие же, как и у владельца, — можно читать файл, но его нельзя изменять или запускать.

Последние три символа (---) задают права доступа для прочих пользователей. Прочие пользователи не имеют права ни читать, ни изменять, ни выполнять этот файл. При попытке получить доступ к файлу они увидят сообщение: **Access denied**.

Права доступа задаются командой `chmod`. Существуют два способа указания прав доступа: символьный (когда указываются символы, задающие право доступа: r, w, x) и абсолютный. Так уж заведено, что в мире UNIX чаще пользуются абсолютным методом. Разберемся, в чем он заключается. Для этого рассмотрим следующий набор прав доступа:

```
rw-r-----
```

Он предоставляет владельцу право чтения и модификации файла (rw-), запускать файл владелец не может. Члены группы владельца могут только просматривать файл (r--), а все остальные пользователи не имеют вообще никакого доступа к файлу.

Возьмем отдельный набор прав, например, для владельца:

```
rw-
```

Чтение разрешено, значит, мысленно записываем 1. Запись разрешена, значит, запоминаем еще 1, а вот выполнение запрещено, поэтому запоминаем 0. Получается число 110. Если из двоичной системы число 110 перевести в восьмеричную, получится число 6. Для перевода можно воспользоваться табл. 14.1.

Таблица 14.1. Преобразование чисел из двоичной системы в восьмеричную

Двоичная система	Восьмеричная система
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Аналогично произведем разбор прав для членов группы владельца. Получится 100, т. е. 4 в восьмеричной системе. С третьим набором (---) все вообще просто: это 000, т. е. восьмеричный 0.

Записываем полученные числа в восьмеричной системе в порядке владелец — группа — остальные. Получится число 640, это и есть права доступа в абсолютной записи. Для того чтобы установить эти права для файла, выполните команду:

```
chmod 640 <имя_файла>
```


Наиболее популярные права доступа:

- ❑ 644 — владельцу можно читать и изменять файл, остальным пользователям — только читать;
- ❑ 666 — читать и изменять файл можно всем пользователям;
- ❑ 777 — всем можно читать, изменять и выполнять файл. Напомню, что для каталога право выполнения — это право просмотра его оглавления.

Если вы хотите "подарить" кому-то файл, т. е. сделать какого-то пользователя владельцем файла, то вам нужно выполнить команду `chown`:

```
chown пользователь файл
```

Учтите, что, возможно, после изменения владельца файла вы сами не сможете получить к нему доступ, ведь его владельцем будете уже не вы.

А теперь поговорим о нюансах доступа к файлам из PHP-сценариев. Предположим, что у вас есть сценарий гостевой книги `gb.php` и файл `gb.txt`, в котором хранятся ее записи. Вы загрузили оба файла на сервер по протоколу FTP. Следовательно, вы — владелец этих двух файлов. По умолчанию (в большинстве случаев) им будут назначены права доступа 644.

Теперь пользователь заходит на страницу гостевой книги и пытается добавить новую запись. PHP пытается открыть файл `gb.txt` в режиме записи, но не может этого сделать. Почему? Потому что интерпретатор PHP выполняется от имени пользователя `apache` (от имени этого же пользователя работает и сам веб-сервер), а для всех остальных пользователей право доступа равно 4 (только чтение).

Что делать в этом случае? Вы можете присвоить файлу `gb.txt` (не `gb.php`!) право доступа 666, но это позволит получить к нему полный доступ не только пользователю `apache`, но и любому другому пользователю. Или же можно "подарить" этот файл пользователю `apache`:

```
chown apache gb.txt
```

Перед таким "подарком" уточните у администратора сервера, от имени какого пользователя выполняется PHP, чтобы случайно не подарить файл другому пользователю. Недостатком этого метода является то, что после этого вы ничего не сможете сделать с файлом `gb.txt` (ни отредактировать, ни удалить), поскольку он принадлежит другому пользователю.

Чтобы не было проблем с доступом к хранилищу данных, рекомендуется применять для хранения данных таблицы MySQL, но иногда (например, когда MySQL недоступен) приходится использовать файлы.

14.2. Чтение файла

Существуют три способа чтения файла:

- ❑ открываем файл функцией `fopen()` и читаем файл функцией `fread`;
- ❑ строчное чтение файла с помощью функции `file()`;
- ❑ чтение всего файла сразу функцией `file_get_contents()`.

14.2.1. Функции *fopen()* и *fread()*

Функции открытия файла `fopen()` нужно передать всего два параметра: имя файла и режим доступа к файлу:

```
fopen (string filename, string mode [, bool use_include_path [, resource
zcontext]])
```

Остальные два параметра настолько редко используются, что мы даже не будем их рассматривать.

Имя файла можно указать разными способами (предположим, что мы хотим получить доступ к файлу `gb.txt`):

- ❑ `/home/den/htdocs/gb.txt` — полный путь к файлу (мы считаем, что файл находится в каталоге `htdocs` домашнего каталога пользователя `den`);
- ❑ `gb.txt` — относительный путь к файлу, файл находится в одном каталоге с PHP-сценарием;
- ❑ `../gb.txt` — файл `gb.txt` находится в родительском каталоге относительно каталога, в котором находится PHP-сценарий;
- ❑ `data/gb.txt` — файл находится в подкаталоге `data` каталога, в котором находится PHP-сценарий.

Второй параметр `fopen()` — это не права доступа, как вы могли подумать, а символы, задающие режим доступа к файлу (табл. 14.2).

Таблица 14.2. Режимы доступа к файлу

Режим	Описание
<code>r</code>	Только чтение. Если файл не существует, функция вернет <code>false</code>
<code>r+</code>	Режим чтения/записи. Указатель файла будет помещен на его начало, т. е. при записи в файл новые данные будут записаны поверх уже имеющихся. Например, если в файле была строка <code>Мама мыла раму</code> , то после записи в файл строки <code>Папа</code> получится строка <code>Папа мыла раму</code> . Чтобы не получилось непонятных ситуаций (что "папа" "мыла"), помните об особенности этого режима — новые данные не вставляются в файл, а записываются поверх уже имеющихся (как в режиме вставки на клавиатуре). Если файл не существует, функция вернет <code>false</code>
<code>w</code>	Функция создаст новый пустой файл. Если файл с таким именем существует, он будет перезаписан. Файл открывается в режиме записи
<code>w+</code>	То же самое, что и <code>r+</code> , но перезаписывает файл, если он существует, или создает файл, если такого файла не существует. Файл открывается в режиме чтения/записи
<code>a</code>	Открывает файл в режиме записи. Запись производится в конец файла. Если файл не существует, функция вернет <code>false</code>
<code>a+</code>	Открывает файл в режиме чтения/записи. Указатель записи файла устанавливается на конец файла. Это наиболее универсальный режим: если файл существует, его содержимое не удаляется, если файл не существует, создается пустой файл
<code>b</code>	Открывает файл в двоичном (бинарном) режиме. Полезен для чтения нетекстовых файлов (когда вы знаете, что делаете)

Большинство ошибок при открытии файлов связано с нарушением прав доступа. Предположим, что вы пытаетесь создать в своем домашнем каталоге `/home/den` файл `gb.txt` (открыть в режиме `w` или `a+`). Но интерпретатор PHP, выполняющийся от имени пользователя `apache`, не имеет доступа к вашему домашнему каталогу в режиме записи, поэтому вы не сможете создать файл. Вот пути решения данной проблемы:

- ❑ назначить домашнему каталогу права доступа `777` — тогда вы сможете без ограничений создавать/модифицировать/удалять файл, но это же смогут делать любые другие пользователи системы. Потом не удивляйтесь, что ваш сайт взломали. Одним словом, этот способ отпадает сразу;
- ❑ создать каталог `data` (в домашнем каталоге) и назначить ему права `777` — в этом каталоге вы сможете создавать и удалять файлы. Если ваши сценарии используют много файлов для хранения данных, это неплохой вариант;
- ❑ создать файл `gb.txt` вручную и присвоить ему права `666` — неплохой вариант, если файл всего один. Если файлов 100, лучше использовать предыдущий вариант.

Для открытия файла следует не просто вызвать функцию `fopen()`, а получить дескриптор файла `$handle`, который потом будет использован в других функциях (чтобы та же функция `fread()` знала, из какого файла нужно произвести чтение, если вы открыли несколько файлов). Вот примеры открытия файлов функцией `fopen()`:

```
$handle = fopen("/home/den/file.txt", "r");  
$handle = fopen("/home/den/file.gif", "wb");  
$handle = fopen("http://www.example.com/", "r");  
$handle = fopen("ftp://user:password@example.com/somefile.txt", "w");
```

После открытия файла нужно проанализировать, открыл ли PHP файл. Если `$handle` установлен в `false`, то файл не открыт. Намного проще это сделать с помощью конструкции `or die`:

```
$handle = fopen("/home/den/file.txt", "w") or die("Не могу создать файл");
```

Как работает эта конструкция, думаю, понятно — если функция возвращает `false`, выводится сообщение об ошибке и завершается работа сценария.

Итак, файл мы открыли. Теперь нужно что-то из него прочитать. Для этого служит функция `fread()`, которой нужно передать два параметра: дескриптор файла и количество байтов, которое должно быть из файла прочитано:

```
// читаем из файла 100 байтов и помещаем в строку $f  
$s = fread($handle, 100);
```

Если в файле будет всего 50 байтов, то функция `fread()` прочитает только эти 50 байтов. При чтении файла нужно контролировать, достигнут ли конец файла. Функция `feof()` возвращает `true`, если достигнут конец файла.

Рассмотрим пример посимвольного чтения файла в строку `$s`:

```
$handle = fopen("file.txt", "r");
```

```
$s = "";
```



```
while (!feof($handle))  
{  
$s = $s . fread($handle, 1);  
}
```

```
echo $s;
```

ПРИМЕЧАНИЕ

Вместо `fread($handle, 1)` можно использовать функцию `fgets($handle)` для чтения одного символа.

По окончании обработки файл нужно закрыть с помощью функции `fclose()`:

```
fclose($handle);
```

Вообще-то, PHP при завершении сценария автоматически закрывает все открытые файлы, но самостоятельное закрытие файла считается хорошим тоном.

14.2.2. Функция *file()*: построчное чтение файла

Функция `fread()` возвращает указанное пользователем количество байтов. А что делать, если нам нужно получить строку файла? Ведь мы не знаем, сколько символов в строке. В одной строке может быть один символ, в другой — 50.

Для чтения строки из файла служат следующие функции:

- ❑ `fgets()` — читает строку из файла;
- ❑ `fgetss()` — читает строку из файла и удаляет HTML-теги.

Вот пример построчного чтения и вывода файла:

```
$handle = fopen("file.txt", "r");
```

```
while (!feof($handle))  
{  
echo fgets($handle);  
}
```

```
fclose($handle);
```

Обратите внимание, что для организации построчного чтения нам потребовались четыре функции. Намного проще использовать функцию `file()`, которая читает файл в массив. Каждый элемент массива — это прочитанная строка из файла. Вот как можно организовать построчное чтение файла:

```
$f = file('file.txt');
```

Все, что осталось, — это вывести файл. Вот пример построчного чтения и вывода HTML-страницы:


```
$lines = file('http://www.example.com/');

foreach ($lines as $line_num => $line) {
    echo "Строка #<b>{$line_num}</b> : " . htmlspecialchars($line) . "<br />\n";
}
```

Если вы будете использовать функции `fgets()` и `fgetss()`, вам нужно знать их прототипы:

```
string fgets (resource handle [, int length])
string fgetss (resource handle [, int length [, string allowable_tags]])
```

Первый параметр — это дескриптор файла, второй — максимальная длина строки в байтах, третий (для функции `fgetss()`) — разрешенные HTML-теги.

14.2.3. Чтение всего файла: функция `file_get_contents()`

Предположим, что мы прочитали файл функцией `file()`. Затем мы его можем обработать построчно — например, вывести. Но использовать цикл для вывода строки не совсем удобно. Несколько удобнее объединить массив в строку и вывести уже ее, например:

```
$s = join('', file('file.txt'));
echo $s;
```

Но и эта конструкция слишком громоздкая. Намного проще использовать функцию `file_get_contents()`, которая возвращает сразу все содержимое файла:

```
echo file_get_contents('file.txt');
```

Эта функция появилась в PHP 5, поэтому если у вас "древняя" версия PHP, то придется использовать функции `join()` и `file()`.

14.3. Запись файла

Запись информации в файл можно осуществить с помощью следующих функций:

- ❑ `file_put_contents()` — записывает строку в файл;
- ❑ `fwrite()` (или `fputs()`) — служит как для записи строки, так и для записи бинарных данных.

Вот прототипы этих функций:

```
int file_put_contents (string filename, mixed data [, int flags [, resource context]])
int fwrite (resource handle, string string [, int length])
```

Функцию `file_put_contents()` использовать весьма просто — вам не потребуется получать дескриптор файла, проверять этот дескриптор, вызывать функцию

`fwrite()`. Достаточно только указать имя файла, данные и флаги. Самый полезный флаг `FILE_APPEND` — его наличие обеспечивает запись данных в конец файла.

Функции `fwrite()` нужно передать дескриптор файла, данные и необязательный параметр `length` — максимальную длину записываемых данных (если строка `string` длиннее, чем `length`, то будет записано `length` байтов).

Если вы хотите убедиться, что функцией `fwrite()` данные записаны в файл, а не остались где-то в буфере записи, используйте функцию `fflush()`:

```
fflush($handle);
```

14.4. Создание временных файлов

Иногда для хранения промежуточных данных нам нужен временный файл. Создавать его с помощью `fopen()` не слишком удобно. Во-первых, нам тогда придется контролировать имя файла — чтобы ненароком не перезаписать существующий файл. Во-вторых, временный файл после обработки нужно будет удалить. Намного проще использовать функцию `tmpfile()`, создающую временный файл в каталоге `/tmp`.

Вот пример работы функции `tmpfile()`:

```
$temp = tmpfile();  
fwrite($temp, "данные");  
fseek($temp, 0);           // переходим на начало файла  
echo fread($temp, 1024);  
fclose($temp);             // при закрытии временный файл удаляется
```

Заметьте, что временный файл открывается в режиме чтения/записи.

14.5. Работа с CSV-файлами

Предположим, что у нас есть небольшая электронная таблица вроде изображенной на рис. 14.1. Сохранив эту таблицу в формате CSV, мы получим файл следующего содержания:

```
Январь;19000  
Февраль;25000  
Март;64400  
Апрель;56000  
Май;29000  
Июнь;35006  
Июль;19800  
Август;17000  
Сентябрь;45000  
Октябрь;59800  
Ноябрь;48000  
Декабрь;75000
```


The screenshot shows a Microsoft Excel window titled 'Книга1'. The menu bar includes 'Файл', 'Правка', 'Вид', 'Вставка', 'Формат', 'Сервис', 'Данные', 'Окно', 'Справка', and 'Adobe PDF'. The toolbar shows various icons for file operations, editing, and formatting. The status bar at the bottom indicates 'Готово' and 'NUM'. The active sheet is 'Лист1'.

	1	2	3	4	5	6	7	8	9	10	11
1	Январь	19000									
2	Февраль	25000									
3	Март	64400									
4	Апрель	56000									
5	Май	29000									
6	Июнь	35006									
7	Июль	19800									
8	Август	17000									
9	Сентябрь	45000									
10	Октябрь	59800									
11	Ноябрь	48000									
12	Декабрь	75000									
13											
14											
15											
16											
17											
18											
19											
20											
21											
22											
23											
24											
25											
26											
27											
28											
29											
30											
31											

Рис. 14.1. Пример электронной таблицы

Несмотря на то что формат называется CSV (Comma-Separated Values, разделенные запятой значения), Excel почему-то разделяет поля точкой с запятой, а не просто запятой. PHP-функция `fgetcsv()` по умолчанию использует в качестве разделителя запятую, поэтому, если мы работаем с CSV-файлами, полученными из Excel, нужно задать разделитель:

```
fgetcsv($handle, 1000, ";")
```

Первый параметр — это дескриптор файла, полученный с помощью `fopen()`, второй — максимальная длина строки в байтах, третий — разделитель. Если разделитель не указан, то в качестве разделителя используется запятая. Если не указана длина (или длина равна 0), то функция будет читать данные до конца строки.

Рассмотрим пример обработки CSV-файла:

```
$row = 1; // номер строки
$handle = fopen("book.csv", "r");
```



```
while (($data = fgetcsv($handle, 1000, ";")) !== FALSE) {  
    // количество полей в строке  
    $num = count($data);  
  
    echo "<p> $num полей в строке $row: <br /></p>\n";  
  
    $row++;  
  
    // выводим поля  
    for ($c=0; $c < $num; $c++) {  
        echo $data[$c] . "<br />\n";  
    }  
}  
fclose($handle);
```

Для записи строки в CSV-файл служит функция `fputcsv()`:

```
int fputcsv (resource handle [, array fields [, string delimiter [, string  
enclosure]])
```

Первый параметр — дескриптор файла, второй — массив полей, которые нужно поместить в файл, третий параметр — разделитель. Последний параметр задает ограничитель полей — по умолчанию это двойная кавычка (").

Каждый элемент массива, передаваемого функции, — это строка, содержащая поля, разделенные разделителем, например:

```
$list = array (  
    'январь;10700',  
    'февраль;1789',  
    'март; 10012'  
);
```

Вот пример записи массива `$list` в файл `file.csv`:

```
$fp = fopen('file.csv', 'w');  
  
foreach ($list as $line) {  
    fputcsv($fp, split(',', $line));  
}  
  
fclose($fp);
```

14.6. Специальные функции для работы с файлами

PHP умеет не только читать и записывать файлы, но и производить операции с файлами на уровне файловой системы: вы можете копировать, переименовывать и удалять файлы. Также в составе PHP есть функции, позволяющие изменять права доступа к файлам и каталогам. Далее мы рассмотрим PHP-функции для работы с файлами, разбитые на группы.

14.6.1. Функции для работы с именами файлов

К данной группе относятся следующие три функции:

- ❑ `basename()` — возвращает имя файла из указанного пути;
- ❑ `realpath()` — возвращает абсолютный путь к файлу;
- ❑ `file_exists()` — позволяет проверить, существует ли файл.

Функции в работе довольно простые, поэтому мы ограничимся примерами их использования:

```
$path = "/home/den/htdocs/index.php";  
$file = basename($path);           // $file содержит "index.php"  
$file = basename($path, ".php");    // $file содержит "index"  
  
// выведет /home/den/htdocs/index.php  
echo realpath("index.php");  
  
if (file_exists($path)) {  
    echo "Файл существует";  
} else {  
    echo "Файл не существует";  
}
```

14.6.2. Работа с правами доступа

Функции работы с правами доступа позволяют установить права доступа, определить права доступа, определить, можно ли выполнять определенные операции с файлом (например, запись или выполнение) и другие операции, связанные с доступом к файлу:

- ❑ `chgrp()` — изменяет группу владельцев файла;
- ❑ `chmod()` — изменяет права доступа к файлу или каталогу;
- ❑ `chown()` — изменяет владельца файла;
- ❑ `filegroup()` — возвращает идентификатор группы файла;
- ❑ `fileowner()` — возвращает идентификатор владельца файла;
- ❑ `fileperms()` — предоставляет информацию о правах на файл или каталог;
- ❑ `is_dir()` — определяет, является ли файл каталогом;
- ❑ `is_executable()` — определяет, является ли файл исполняемым;
- ❑ `is_file()` — определяет, является ли файл обычным файлом;
- ❑ `is_link()` — определяет, является ли файл символической ссылкой;
- ❑ `is_readable()` — определяет, доступен ли файл для чтения;
- ❑ `is_uploaded_file()` — определяет, был ли файл загружен с помощью HTTP POST;
- ❑ `is_writable()` — определяет, доступен ли файл для записи.

Функциям `chgrp()`, `chmod()` и `chown()` нужно передать имя файла и, соответственно, идентификатор группы, значение прав доступа, идентификатор пользователя (нового владельца файла), например:

```
chgrp("file.txt", "den");
chgrp("file.txt", 500);
chown("file.txt", "den");
chmod("file.txt", 666);
```

ПРИМЕЧАНИЕ

Идентификаторы пользователя и группы можно задавать как строкой, так и числом. Права доступа для `chmod()` нужно задавать только числом.

Функциям `filegroup()`, `fileowner()` и `fileperms()` нужно передать всего один параметр — имя файла. Функции возвращают число: идентификатор группы, владельца, значение прав доступа соответственно. Пример:

```
echo substr(sprintf('%o', fileperms('/etc/passwd')), -4);
```

Функциям `is_*` нужно тоже передать имя файла/каталога, а в результате получить `true` или `false` в зависимости от того, истинно ли проверяемое функциями условие или нет:

```
if (is_writable('file.txt'))
echo "Файл доступен для записи";
else
echo "Файл не доступен для записи";
```

14.6.3. Копирование, переименование и удаление файлов

Для копирования файла служит функция `copy()`:

```
bool copy (string source, string dest)
```

Первый параметр — это имя исходного файла, второй параметр — имя файла назначения (файл будет перезаписан в случае существования). Если копирование прошло удачно, функция возвращает `true`, в противном случае — `false`.

Небольшой пример копирования файла:

```
$file = 'file.txt';
$newfile = 'file.txt.bak';

if (!copy($file, $newfile)) {
echo "Не удалось скопировать $file...\n";
}
```

Для переименования или перемещения файла служит функция `rename()`:

```
bool rename (string oldname, string newname)
```

Как и функция `copy()`, данная функция возвращает `true` в случае успешного завершения операции. Пример переименования/перемещения файла:


```
// переименовываем /tmp/tmp.txt в /home/den/my_file.txt
```

```
rename("/tmp/tmp.txt", "/home/den/my_file.txt");
```

Для перемещения загруженного файла в другой каталог служит функция `move_uploaded_file()`:

```
bool move_uploaded_file (string filename, string destination)
```

Данная функция похожа на `rename()`, но перед перемещением проверяет, был ли файл загружен по HTTP.

Функция `unlink()` удаляет файл, которому нужно передать имя удаляемого файла:

```
unlink('file.txt');
```

При выполнении операций с файлами помните о правах доступа:

- ☐ чтобы PHP мог скопировать файл в указанный вами каталог, пользователь, от имени которого запущен PHP, должен иметь право записи в этот каталог;
- ☐ чтобы прочитать файл (например, при копировании), у PHP должно быть право на чтение файла.

14.6.4. Время доступа к файлу

Для получения и установки времени последнего доступа к файлу используются следующие функции:

- ☐ `fileatime()` — возвращает время последнего доступа к файлу;
- ☐ `filemtime()` — возвращает время последнего изменения файла;
- ☐ `touch()` — устанавливает время доступа и модификации файла.

Первым двум функциям нужно передать всего один параметр — имя файла. Функции возвращают время в формате UNIX timestamp (число секунд, прошедшее с полуночи 1 января 1970 г.).

Функции `touch()` можно передать три параметра:

```
bool touch (string filename [, int time [, int atime]])
```

Первый параметр — имя файла, второй — время доступа и модификации файла. Если вам нужно задать разное время для операций доступа (открытия) и модификации (изменения), то укажите третий параметр — время доступа. Тогда второй параметр будет использоваться для установки только времени модификации файла. Время нужно задавать в формате timestamp (см. главу 12).

Если второй (понятно, и третий) параметр не задан, то будет использовано текущее время.

14.6.5. Другие полезные функции

Функция `filesize()` позволяет получить размер файла в байтах. Ей нужно передать только имя файла. А с помощью функции `filetype()` вы сможете определить тип файла. Функции нужно передать имя файла.

Вот возможные значения результата:

- ☐ `fifo` — очередь, First In First Out;
- ☐ `char` — символьное устройство UNIX (например, консоль или клавиатура);
- ☐ `dir` — каталог;
- ☐ `block` — блочное устройство UNIX (например, жесткий диск);
- ☐ `link` — ссылка;
- ☐ `file` — обычный файл;
- ☐ `unknown` — файл неизвестного типа.

Например:

```
echo filetype('/etc/passwd');      // выведет file
echo filetype('/etc/');            // выведет dir
```

Нужно заметить, что в Windows будут доступны лишь два типа: `file` и `dir`. Остальные типы доступны только в UNIX.

14.7. Совместный доступ к файлу

Предположим, что мы разрабатываем сценарий гостевой книги, который хранит записи в файле `gb.txt`. Вроде бы все просто: сначала сценарий открывает файл только для чтения и выводит записи гостевой книги, затем, если пользователь пожелает оставить сообщение, файл открывается для записи, и новое сообщение записывается в файл.

Но что делать, если два пользователя одновременно пытаются добавить сообщение в гостевую книгу, или один сценарий открыл файл для записи, а второй пытается открыть для чтения?

Результат в обоих случаях непредсказуем. Поэтому, чтобы быть уверенным, что одновременно отправленные данные записаны в нужном порядке (а не вперемешку), и что сценарий точно сможет прочитать файл, нужно использовать механизм блокировок. Этот механизм доступен только в UNIX, т. е. он будет работать на сервере хостинг-провайдера, но не на вашем домашнем компьютере под управлением Windows.

Для установки блокировки служит функция `flock()`:

```
bool flock (resource handle, int operation [, int &wouldblock])
```

Первый параметр — это дескриптор файла, второй — тип блокировки. Если третий параметр установлен в `true`, то нужно не только запереть файл, но и блокировать выполнение сценария, пока файл не станет доступным.

Рассмотрим значения второго параметра:

- ☐ `LOCK_EX` — исключительная блокировка, ее нужно применять при записи в файл, при этом все остальные сценарии не смогут получить доступ к файлу, пока вы не снимете блокировку (или пока не завершится ваш сценарий);

- ❑ `LOCK_SH` — разделяемая блокировка, ее можно использовать при чтении данных из файла;
- ❑ `LOCK_NB` — если другой процесс заблокировал файл, то вы можете или ждать, пока файл разблокируется, или же получить сообщение об ошибке и завершить выполнение сценария. Значение `LOCK_NB`, добавленное, например, к исключительной блокировке, запрещает ожидание;
- ❑ `LOCK_UN` — разблокирование файла:

```
$handle = fopen("gb.txt", "a+");

// исключительная блокировка
if (flock($handle, LOCK_EX+LOCK_NB)) {
    fwrite($handle, "Данные");
    flock($handle, LOCK_UN);           // отпираем файл
} else {
    echo "Не могу выполнить блокировку файла!";
}

fclose($fp);                        // закрываем файл
```

14.8. Функции для работы с каталогами

Для создания каталога служит функция `mkdir()`:

```
bool mkdir (string pathname [, int mode [, bool recursive [, resource
context]])
```

Первый параметр — это имя создаваемого каталога, а второй — права доступа к каталогу. Понятное дело, что у вас должно быть право на изменение каталога, в котором вы собираетесь создать новый подкаталог. Третий параметр нужно установить в `true`, если вы хотите создать дерево каталогов.

Примеры использования функции `mkdir()`:

```
mkdir("images", 777);
mkdir("gallery/upload", 777, true);
```

Функция `rmdir()` удаляет пустой каталог:

```
rmdir("images");
```

Функция `dirname()` возвращает имя каталога из пути, например:

```
$path = "/etc/passwd";
echo dirname($path);           // выведет /etc
```

Для получения списка файлов каталога служит функция `scandir()`:

```
array scandir (string $directory [, integer $sorting_order])
```

Первый параметр задает имя каталога, второй — порядок сортировки (1 — обратная сортировка). Пример использования этой функции:


```
$dir = '/home/den';  
$files_1 = scandir($dir);  
$files_2 = scandir($dir, 1);  
  
print_r($files_1);  
print_r($files_2);
```

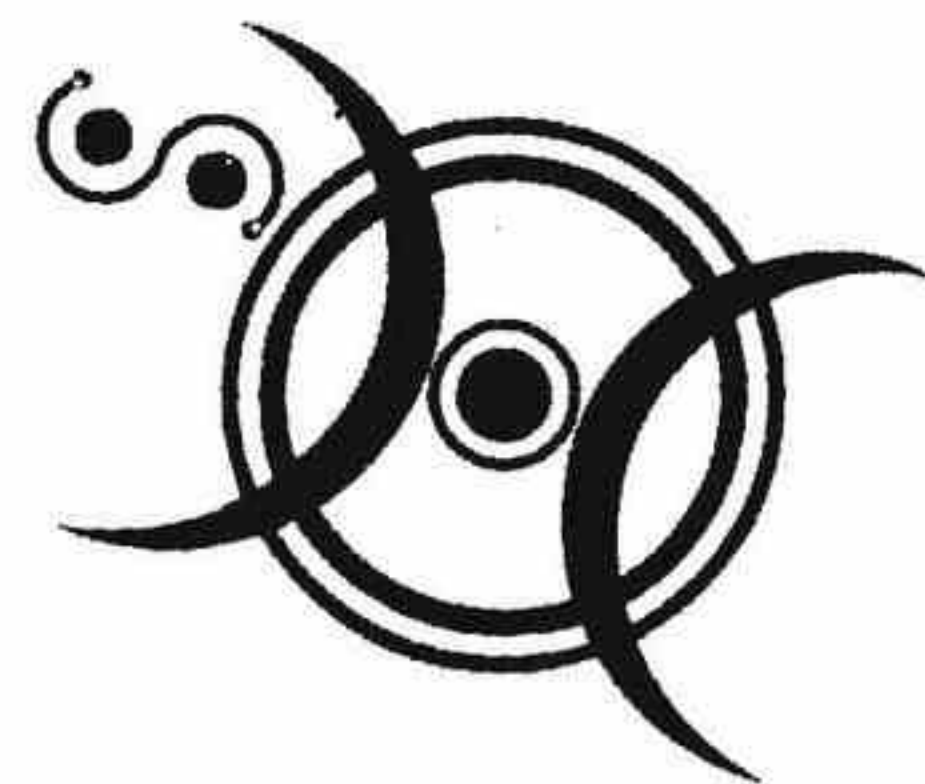
Функция `scandir()` появилась только в PHP 5, в более "древних" версиях PHP для вывода содержимого каталога служили функции `opendir()` и `readdir()`:

```
$dir = "/home/den";  
  
$dh = opendir($dir);  
while (false !== ($filename = readdir($dh))) {  
    $files[] = $filename;  
}  
  
// сортировка массива files в алфавитном порядке  
sort($files);  
print_r($files);  
  
// обратная сортировка массива files  
rsort($files);  
print_r($files);
```

ПРИМЕЧАНИЕ

Много различных примеров использования функции `scandir()` вы найдете по адресу: <http://www.php.net/manual/ru/function.scandir.php>.

ГЛАВА 15



Вывод графических изображений средствами PHP

15.1. Библиотека GD

Библиотека GD предназначена для работы с изображениями и графическими файлами форматов JPEG, GIF, PNG, WBMP, XPM и WebP. Используя эту библиотеку, можно программно создавать изображения, сохранять их в разных форматах или сразу выводить в браузер.

Тем не менее, хотя PHP и имеет в наличии все функции, необходимые для создания графических шедевров, гораздо удобнее это делать в том же Adobe Photoshop. Ведь в графическом редакторе сразу видно, где окажется та или иная линия, а вот при работе с PHP вам придется в буквальном смысле слова представлять, как будет выглядеть картинка попиксельно (в качестве домашнего задания предлагаю вам программным способом нарисовать пингвина размером 100×100 пикселей, не прибегая к помощи Photoshop. Шутка!). Вот поэтому практически никто не создает в PHP изображения с нуля, а модифицирует уже готовые картинки, нарисованные в полноценном графическом редакторе. Предположим, вам требуется счетчик. В графическом редакторе вы создаете его фоновое изображение, а с помощью PHP просто в нужном месте выводите значение.

Сначала мы опишем некоторые очень полезные функции, необходимые нам для реализации графической галереи, а далее будут рассмотрены два практических примера: изменение размера изображения и нанесение на него водяных знаков.

15.1.1. Получение информации об изображении

Для определения формата графического файла служит функция `exif_imagetype()`, которой нужно передать имя графического файла:

```
int exif_imagetype (string filename)
```

Функция возвращает числовое значение, соответствующее формату файла (табл. 15.1).

Если формат файла определить не удалось, то функция возвращает `false`.

Таблица 15.1. Значения, возвращаемые функцией `exif_imagetype()`

Значение	Константа	Значение	Константа
1	IMAGETYPE_GIF	9	IMAGETYPE_JPC
2	IMAGETYPE_JPEG	10	IMAGETYPE_JP2
3	IMAGETYPE_PNG	11	IMAGETYPE_JPX
4	IMAGETYPE_SWF	12	IMAGETYPE_JB2
5	IMAGETYPE_PSD	13	IMAGETYPE_SWC
6	IMAGETYPE_BMP	14	IMAGETYPE_IFF
7	IMAGETYPE_TIFF_II	15	IMAGETYPE_WBMP
8	IMAGETYPE_TIFF_MM	16	IMAGETYPE_XBM

Еще одна очень полезная функция — `exif_read_data()`, позволяющая прочитать заголовки JPEG- и TIFF-файлов и вернуть ассоциативный массив, содержащий полную информацию о графическом файле:

```
array exif_read_data (string filename
                    [, string sections
                    [, bool arrays
                    [, bool thumbnail]]])
```

Первый параметр — это имя файла, второй — список секций (через запятую), которые нужно вернуть. Список разделов и содержащихся в них параметров графического файла приведен в табл. 15.2.

Таблица 15.2. Разделы и параметры графического файла

Раздел	Параметры
FILE	FileName — имя файла; FileSize — размер файла; FileDateTime — дата создания файла; SectionsFound — секции файла
COMPUTED	html — HTML-строка для тега IMG (<width> и <height>) Width — ширина картинки Height — высота
IFDO	Все данные с тегами IFDO
THUMBNAIL	Содержит миниатюру (уменьшенное изображение картинки), если она создана, и информацию о ней
COMMENT	Комментарии для графического файла (поддерживаются форматом JPEG)
EXIF	Детальная информация об изображении. Формируется, как правило, цифровыми фотоаппаратами

Третий параметр, `arrays`, определяет, будет ли каждый раздел оформлен в виде отдельного массива (`arrays = true`) или нет (`arrays = false`). Разделы `FILE`, `COMPUTED` и `THUMBNAIL` всегда формируются как массивы.

Последний параметр, `thumbnail`, определяет, нужно ли загружать саму миниатюру (`true`) или только информацию о ней (`false`).

Рассмотрим пример использования функции `exif_read_data()` (листинг 15.1).

Листинг 15.1. Пример использования функции `exif_read_data()`

```
<?
$info = exif_read_data ('av.jpg','', true, false);

echo "<p>Раздел <b>FILE</b>";
echo "<br>Имя файла " . $info['FILE']['FileName'];
echo "<br>Размер файла " . $info['FILE']['FileSize'];
echo "<br>Дата создания " . $info['FILE']['FileDateTime'];
echo "<br>Секции " . $info['FILE']['SectionsFound'];
echo "<p>Раздел <b>COMPUTED</b>";
echo "<br>HTML-строка " . $info['COMPUTED']['html'];
echo "<br>Ширина " . $info['COMPUTED']['Width'];
echo "<br>Высота " . $info['COMPUTED']['Height'];
echo "<br>";
var_dump($info['COMPUTED']);
echo "<p>Раздел <b>IFD0</b><p>";
var_dump($info['IFD0']);
echo "<p>Раздел <b>THUMBNAIL</b><p>";
var_dump($info['THUMBNAIL']);

echo "<p>Раздел <b>COMMENT</b><p>";
var_dump($info['COMMENT']);

echo "<p>Раздел <b>EXIF</b><p>";
var_dump($info['EXIF']);

?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *15-1.php*), представленный в каталоге *Glava_15* сопровождающего книгу электронного архива (см. приложение 4).

Результат выполнения сценария представлен на рис. 15.1.


```

Раздел FILE
Имя файла av.jpg
Размер файла 12768
Дата создания 1159095403
Секции ANY_TAG, IFD0, THUMBNAIL, EXIF, INTEROP

Раздел COMPUTED
HTML-строка width="100" height="75"
Ширина 100
Высота 75
array(10) ( ["html"]=> string(23) "width="100" height="75" ["Height"]=> int(75) ["Width"]=> int(100) ["IsColor"]=> int(1) ["ByteOrderMotorola"]=> int
(0) ["ApertureFNumber"]=> string(5) "f2.8" ["UserComment"]=> string(1) " " ["UserCommentEncoding"]=> string(9)
"UNDEFINED" ["Thumbnail FileType"]=> int(2) ["Thumbnail MimeType"]=> string(10) "image/jpeg" )

Раздел IFD0
array(11) ( ["ImageDescription"]=> string(31) "OLYMPUS DIGITAL CAMERA " ["Make"]=> string(21) "OLYMPUS IMAGING CORP." ["Model"]
=> string(12) "FE-120,X-700" ["Orientation"]=> int(1) ["XResolution"]=> string(4) "72/1" ["YResolution"]=> string(4) "72/1" ["ResolutionUnit"]=> int(2)
["Software"]=> string(27) "ACD Systems Digital Imaging" ["DateTime"]=> string(19) "2006:09:24 14:01:30" ["YCbCrPositioning"]=> int(1)
["Exif_IFD_Pointer"]=> int(277) )

Раздел THUMBNAIL
array(3) ( ["Compression"]=> int(6) ["JPEGInterchangeFormat"]=> int(2950) ["JPEGInterchangeFormatLength"]=> int(5236) )

Раздел COMMENT
NULL

Раздел EXIF

```

Рис. 15.1. Информация о JPEG-файле (фрагмент вывода)

15.1.2. Конвертирование графических форматов

Функция `jpeg2wbmp()` осуществляет преобразование файла из формата JPEG в формат Windows BMP:

```

int jpeg2wbmp (string jpegname,
               string wbmpname,
               int d_height,
               int d_width,
               int threshold)

```

Первый параметр — это имя файла изображения в формате JPEG, второй — имя результирующего BMP-файла. Параметры `d_height`, `d_width` задают соответственно высоту и ширину изображения BMP-формата. Последний параметр обычно можно установить в ноль.

Преобразование файла формата PNG в формат Windows BMP осуществляет функция `png2wbmp()`. Ее параметры аналогичны параметрам предыдущей функции:

```

int png2wbmp (string pngname,
              string wbmpname,
              int d_height,
              int d_width,
              int threshold)

```

Но если необходимо преобразовать файл в формат, отличный от BMP, придется создавать конвертер вручную. К счастью, это очень просто. Рассмотрим следующий пример (листинг 15.2).

Листинг 15.2. Простейший графический конвертер

```
<?php
$image = ImageCreateFromPng("image.png");
Header("Content-type: image/gif");
ImageGif($image);
ImageDestroy($image);
?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *15-2.php*), представленный в каталоге *Glava_15* сопровождающего книгу электронного архива (см. приложение 4).

Да, именно так выглядит простейший графический конвертер. Обратите внимание — мы с помощью функции `ImageCreateFromPng()` открываем PNG-файл, а затем с помощью функции `ImageGif()` выводим это изображение в браузер как GIF-файл. Функция `Header()` информирует браузер, что мы сейчас будем выводить GIF-изображение, а не какое-то другое. Если наш PHP-файл называется *15-2.php*, то в теге `IMG` вы смело можете написать:

```
<img src=http://localhost/15-2.php>
```

Браузер отобразит GIF-картинку.

Если вам нужно не выводить картинку в браузер, а сохранить ее на диске в другом формате, следует указать еще один параметр для функции `ImageGif()` — имя файла:

```
ImageGif($image, 'image.gif');
```

Понятно, что функция `Header()` уже не потребуется, поскольку мы указали имя файла, и вывод функции будет направлен в этот файл, а не во входной поток браузера. Другие функции, пригодные для конвертирования графических форматов, приведены в табл. 15.3.

Таблица 15.3. Функции загрузки и сохранения графических файлов

Функция	Описание
<code>resource imagecreate (int x_size, int y_size)</code>	Создает пустое изображение указанного размера. Именно эту функцию нужно использовать, если у вас появится желание нарисовать свой "шедевр" с нуля
<code>resource imagecreatefromgd (string filename)</code>	Загружает изображение формата GD из файла или URL
<code>resource imagecreatefromgd2 (string filename)</code>	Загружает изображение формата GD2 из файла или URL
<code>resource imagecreatefromgif (string filename)</code>	Загружает изображение из файла формата GIF или URL
<code>resource imagecreatefromjpeg (string filename)</code>	То же, JPEG

Таблица 15.3 (окончание)

Функция	Описание
<code>resource imagecreatefrompng (string filename)</code>	То же, PNG
<code>resource imagecreatefromwbmp (string filename)</code>	То же, WBMP
<code>resource imagecreatefromxbm (string filename)</code>	То же, XBM
<code>resource imagecreatefromxpm (string filename)</code>	То же, XPM
<code>int imagegd (resource image [, string filename])</code>	Сохраняет изображение на диск в формате GD. Если имя файла не задано, то картинка передается на стандартный вывод, т. е. в браузер. Аналогично работают остальные функции сохранения графических файлов
<code>int imagegd2 (resource image [, string filename [, int chunk_size [, int type]]])</code>	Сохраняет изображение на диск в формате GD2
<code>int imagegif (resource image [, string filename])</code>	Сохраняет изображение на диск в формате GIF
<code>int imagejpeg (resource image [, string filename [, int quality]])</code>	Сохраняет изображение на диск в формате JPEG. Последний параметр, <code>quality</code> , задает качество изображения (от 0 до 100). Чем оно выше, тем больше размер файла. По умолчанию качество равно 75
<code>int imagepng (resource image [, string filename])</code>	Сохраняет изображение на диск в формате PNG
<code>int imagewbmp (resource image [, string filename [, int foreground]])</code>	Сохраняет изображение на диск в формате WBMP. Параметр <code>foreground</code> задает цвет переднего плана (по умолчанию черный, т. е. <code>foreground = 0</code>). Для вывода такого изображения в браузер нужно передать заголовок: <code>header('Content-type: image/vnd.wap.wbmp');</code>
<code>int imagexbm (resource image, string filename [, int foreground])</code>	Сохраняет изображение на диск в формате XBM. Параметр <code>foreground</code> задает цвет переднего плана (по умолчанию 0)

15.1.3. Вывод текста поверх картинки. Задание цвета

Для вывода строки служит функция `imagestring()`:

```
int imagestring (resource image,
                int font,
                int x,
                int y,
                string s,
                int color)
```


Эта функция выводит строку *s*, расположенную по горизонтали, начиная с координат *x* и *y* (цвет букв задается параметром *color*). Параметр *font* определяет шрифт, который будет использоваться для вывода строки. Параметры *x* и *y* задают координаты верхнего левого угла первого символа.

Если требуется вывести строку вертикально, то вызывается функция `imagestringup()`:

```
int imagestringup (resource image,
                  int font,
                  int x,
                  int y,
                  string s,
                  int col)
```

Параметры *x* и *y* задают координаты верхнего левого угла самого верхнего (первого) символа строки.

ПРИМЕЧАНИЕ

Для вывода одного символа поверх картинки можно использовать функции `imagechar()` и `imagecharup()`. Но, как правило, в этих функциях нет необходимости — из соображений универсальности для вывода строки, состоящей из одного символа, лучше применить функции `imagestring()` и `imagestringup()`.

Задать цвет можно с помощью функции `imagecolorallocate()`:

```
int imagecolorallocate (resource image,
                       int red,
                       int green,
                       int blue)
```

Цвет задается в формате RGB (Red, Green, Blue). В табл. 15.4 приведены RGB-значения для основных цветов.

Таблица 15.4. Основные шестнадцать цветов

Название	RGB-значение
Aqua (морская волна)	(000,255,255)
Black (черный)	(000,000,000)
Blue (голубой)	(000,000,255)
Fuchsia (фуксия)	(255,000,255)
Gray (серый)	(128,128,128)
Green (зеленый)	(000,128,000)
Lime (ярко-зеленый)	(000,255,000)
Maroon (темно-бордовый)	(128,000,000)
Navy (темно-синий)	(000,000,128)
Olive (оливковый)	(128,128,000)
Purple (фиолетовый)	(128,000,128)

Таблица 15.4 (окончание)

Название	RGB-значение
Red (красный)	(255,000,000)
Silver (серебряный)	(192,192,192)
Teal (серо-зеленый)	(000,128,128)
White (белый)	(255,255,255)
Yellow (желтый)	(255,255,000)

Получить цвет можно и из самого изображения. Для этого служит функция `imagecolorat()`, которая возвращает значение цвета пиксела с координатами `$x` и `$y` из картинке `$image`:

```
int imagecolorat ( resource $image , int $x , int $y )
```

Эта функция аналогична инструменту "Пипетка" в графических редакторах.

Вот небольшой пример вывода строки красным цветом:

```
$color = imagecolorallocate ($image, 255, 0, 0);  
imagestring ($image, 1, 5, 5, 'Test', $color);
```

Параметр `font` функций `imagestring()` и `imagestringup()` задает шрифт, которым будет выведена строка. По умолчанию вам доступно пять шрифтов с номерами от 1 до 5 (1 — самый мелкий шрифт, 5 — самый большой).

Нужно отметить, что у функций `imagestring()` и `imagestringup()` есть один большой недостаток — они не позволяют вывести русский текст. Обычно это не критично, поскольку выводятся различные числовые значения — например, показания счетчика, но если вам потребуются символы кириллицы, тогда вам не обойтись без функции `imagefttext()`:

```
array imagefttext (resource image,  
    int size,  
    int angle,  
    int x,  
    int y,  
    int color,  
    string fontfile,  
    string text)
```

Функция `imagefttext()` позволяет вывести текст заданным ТТФ-шрифтом (в каталоге `Windows\Fonts` вы найдете ТТФ-шрифты на любой вкус). Файл шрифта задается параметром `fontfile`, размер шрифта (в пикселах) — параметром `size`.

Параметр `angle` задает угол поворота текста против часовой стрелки. Параметры `x`, `y` задают координаты точки, с которой начинается вывод текста.

Параметр `color` — это идентификатор цвета, а `text` — строка, которую нужно вывести.

Вам необходимо помнить, что большинство TTF-шрифтов рассчитаны на использование кодировки Windows-1251, поэтому строка `text` должна быть написана именно в этой кодировке.

Рассмотрим небольшой пример вывода текста на русском языке (листинг 15.3).

Листинг 15.3: Вывод текста на русском языке

```
<?php

// создаем пустое изображение
$image = imagecreate(500, 500);

// фон (белый)
$back = imagecolorallocate($image, 255, 255, 255);

// цвет шрифта (синий)
$color = imagecolorallocate($image, 0, 0, 255);

// выводим строку "Привет"
// размер шрифта — 20 пунктов
// цвет — синий
// гарнитура — Arial (файл arial.ttf находится в одном каталоге // со
сценарием)
// угол — 0
// 10, 10 — координаты верхнего левого угла первой буквы
imagettftext($image, 20, 0, 10, 10, $color, "arial.ttf", "Привет");

// выводим картинку в формате PNG в браузер
Header('Content-type: image/png');

ImagePng($image);

?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл `15-3.php`), представленный в каталоге *Glava_15* сопровождающего книгу электронного архива (см. приложение 4).

15.1.4. Прозрачность

Частенько приходится задавать прозрачный цвет, чтобы он не отображался, а заменялся цветом фона. Самый простой пример: у нас есть изображение на черном фоне, а фон нашего сайта — белый (или синий, зеленый — не важно, но не черный). Поэтому мы можем задать черный цвет прозрачным, и он не будет отображаться. Для этого существует функция `imagecolortransparent()`:

```
int imagecolortransparent (resource image [, int color])
```


Все что нужно — это задать цвет, который необходимо сделать прозрачным. Вот небольшой пример:

```
// делаем черный цвет прозрачным
$black_color = imagecolorallocate($image, 0, 0, 0);
imagecolortransparent ($image, $black_color);
```

Помните, что если другая часть изображения (а не только фон) тоже черная, то она также станет прозрачной.

15.1.5. Рисование графических примитивов

В составе библиотеки GD имеются функции, позволяющие нарисовать графические примитивы: линии, прямоугольники, эллипсы и т. д. Начнем с рисования одного пиксела:

```
bool imagesetpixel (resource $image, int $x, int $y, int $color)
```

Первый параметр — идентификатор картинки, далее координаты пиксела, последний параметр — цвет.

Далее нарисуем линию:

```
bool imageline (resource $image, int $x1, int $y1, int $x2, int $y2, int $color)
```

Первый параметр — идентификатор изображения, затем следуют координаты линии, которая будет нарисована от точки (x1, y1) до (x2, y2). Последний параметр задает цвет линии.

ПРИМЕЧАНИЕ

Для рисования пунктирной линии ранее применялась функция `imagedashedline()`. Теперь она упразднена, и вместо нее нужно использовать комбинацию функций `imagesetstyle()` и `imageline()`.

Перед рисованием линии можно задать ее стиль (функция `imagesetstyle`), толщину (`imagesetthickness`) и кисть (`imagesetbrush`), которой будет нарисована линия. Начнем с функции `imagesetstyle()`:

```
bool imagesetstyle (resource $image, array $style)
```

Первый параметр — изображение, второй — массив пикселей. Рассмотрим пример. Нарисуем пунктирную линию:

```
/* 5 красных и 5 белых пикселей */
$wh = imagecolorallocate($im, 255, 255, 255);
$red = imagecolorallocate($im, 255, 0, 0);
$style = array($red, $red, $red, $red, $red, $wh, $wh, $wh, $wh, $wh);
imagesetstyle($im, $style);
imageline($im, 0, 0, 100, 100, IMG_COLOR_STYLED);
```

Толщина линии устанавливается функцией `imagesetthickness()`:

```
bool imagesetthickness (resource $image, int $thickness)
```


Тут все просто: первый параметр — изображение, а второй — толщина в пикселах. Осталось разобраться, как устанавливается кисть для рисования:

```
bool imagesetbrush ( resource $image , resource $brush )
```

Кисть — это, по сути, другое изображение, которым будет нарисована линия. Небольшой пример:

```
// загружаем кисть
$brush = imagecreatefrompng('./brush.png');
// создаем главное изображение
$im = imagecreatetruecolor(100, 100);
// устанавливаем кисть
imagesetbrush($im, $brush);
// рисуем линию
imageline($im, 50, 50, 50, 60, IMG_COLOR_BRUSHED);
```

Когда мы знаем, как нарисовать точку и линию, можно перейти к более сложным примитивам: прямоугольнику, эллипсу и дуге:

```
bool imagerectangle (resource $image, int $x1, int $y1, int $x2, int $y2, int $color)
bool imageellipse (resource $image, int $cx, int $cy, int $width, int $height, int $color)
bool imagearc (resource $image, int $cx, int $cy, int $width, int $height, int $start, int $end, int $color)
```

Как обычно, первый параметр всех этих функций задает идентификатор рисунка, последний — цвет, которым будет нарисован примитив. В случае с прямоугольником (функция `imagerectangle()`) вам нужно задать координаты верхнего левого (x_1 и y_1) и нижнего правого (x_2 и y_2) углов.

Эллипс (`imageellipse()`) тоже рисуется несложно. Параметры cx и cy — это координаты центра, $width$ и $height$ — ширина и высота эллипса (в пикселах) соответственно.

Функция `imagearc()` рисует дугу (часть эллипса) с центром в cx , cy (у самого верхнего левого пиксела координаты 0, 0), $width$ и $height$ — ширина и высота воображаемого эллипса. Параметры $start$ и end — это начальная и конечная точки, указываются в градусах. Помните, что 0 находится в позиции "три часа", а сама дуга рисуется против часовой стрелки.

У функций `imagerectangle()`, `imageellipse()` и `imagearc()` есть "filled"-аналоги:

```
bool imagefilledrectangle (resource $image, int $x1, int $y1, int $x2, int $y2, int $color)
bool imagefilledellipse (resource $image, int $cx, int $cy, int $width, int $height, int $color)
bool imagefilledarc (resource $image, int $cx, int $cy, int $width, int $height, int $start, int $end, int $color, int $style)
```

Разница в том, что обычные функции рисуют неокрашенные версии фигур, а `filled`-версии окрашивают фигуры внутри цветом $color$.

Самая сложная функция — `imagepolygon()`, рисующая многоугольник. Прототип этой функции следующий:

```
bool imagepolygon (resource $image, array $points, int $num_points, int $color)
```

Первый и последний параметры, надеюсь, понятны. Массив `$points` задает координаты точек в формате:

```
points[0]      = x0
points[1]      = y0
points[2]      = x1
points[3]      = y1
...
```

То есть четный элемент массива — это координата *x* точки, а нечетный элемент — координата *y*. Параметр `$num_points` задает общее количество точек (число вершин многоугольника).

Аналог этой функции — `imagefilledpolygon()`. Рисует окрашенный многоугольник, если, конечно, он замкнутый.

15.1.6. Поворот изображения

Для поворота изображения служит функция `imagerotate()`:

```
resource imagerotate (resource $image, float $angle, int $bgd_color [, int
$ignore_transparent = 0 ])
```

Как обычно, первый параметр — это изображение. Вторым параметром — угол поворота. Центром поворота всегда является центр изображения. Повернутое изображение может отличаться размером от оригинального. Если при повороте остается пустая зона, то она окрашивается цветом `$bgcolor`. Если последний параметр не равен 0, то прозрачность игнорируется.

15.2. Изменение размера изображения

Вспомним, как организована любая графическая галерея. Сначала выводятся миниатюрные изображения. Как правило, размер миниатюры равен 100×80 пикселей. При щелчке на миниатюре выводится увеличенное промежуточное изображение, размер которого зависит от самого изображения и от дизайна галереи — важно, чтобы картинка вписывалась в окно браузера. Промежуточный размер обычно равен 480×280 или 640×480 пикселей. Еще один щелчок — и пользователь может увидеть оригинальное изображение в полном размере. Оригинальное изображение обычно открывается в отдельном окне, поэтому о дизайне никто не думает — важно вывести изображение в таком виде, в каком пользователь поместил его в галерею.

В PHP нет встроенной функции, позволяющей изменять размер изображения, но мы можем написать ее самостоятельно (листинг 15.4).

Листинг 15.4. Функция изменения размера картинки

```
function imageresize($src, $dest, $width, $height, $quality) {  
  
    $im=imagecreatefromjpeg($src);  
    $im1=imagecreatetruecolor($width, $height);  
    imagecopyresampled($im1,$im,0,0,0,0,$width,$height,imagesx($im),imagesy($im));  
  
    imagejpeg($im1,$dest,$quality);  
  
    imagedestroy($im);  
    imagedestroy($im1);  
}
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *15-4.php*), представленный в каталоге *Glava_15* сопровождающего книгу электронного архива (см. приложение 4).

Разберемся с параметрами функции. Первый параметр — это имя исходного (большого) файла. Второй параметр — имя результирующего (маленького) файла. Параметры *width* и *height* задают ширину и высоту результирующего файла. Параметр *quality* задает качество результирующего JPEG-файла.

В этой функции мы задействуем вспомогательные функции, входящие в библиотеку GD:

- ❑ *imagesx()* — возвращает ширину изображения;
- ❑ *imagesy()* — возвращает высоту изображения;
- ❑ *imagecopyresampled()* — копирует прямоугольную часть одного изображения на другое изображение, при этом изменяет размер и следит за качеством изображения — чтобы при уменьшении размера картинки это не отразилось на ее качестве.

Внимательное изучение кода нашей функции (см. листинг 15.4) показывает, что фактически изменяет размер изображения именно функция *imagecopyresampled()*, копирующая прямоугольные области с одного изображения на другое. Наша функция только подготавливает все для ее работы.

Вот пример вызова этой функции:

```
imageresize('big_file.jpg','big_file_thumb.jpg',160,100,75);
```

Созданная нами функция работает только с JPEG-файлами, но при желании вы без проблем можете переделать ее для работы с GIF- и PNG-файлами.

На рис. 15.2 изображен результат работы нашей функции на примере реально работающего проекта.

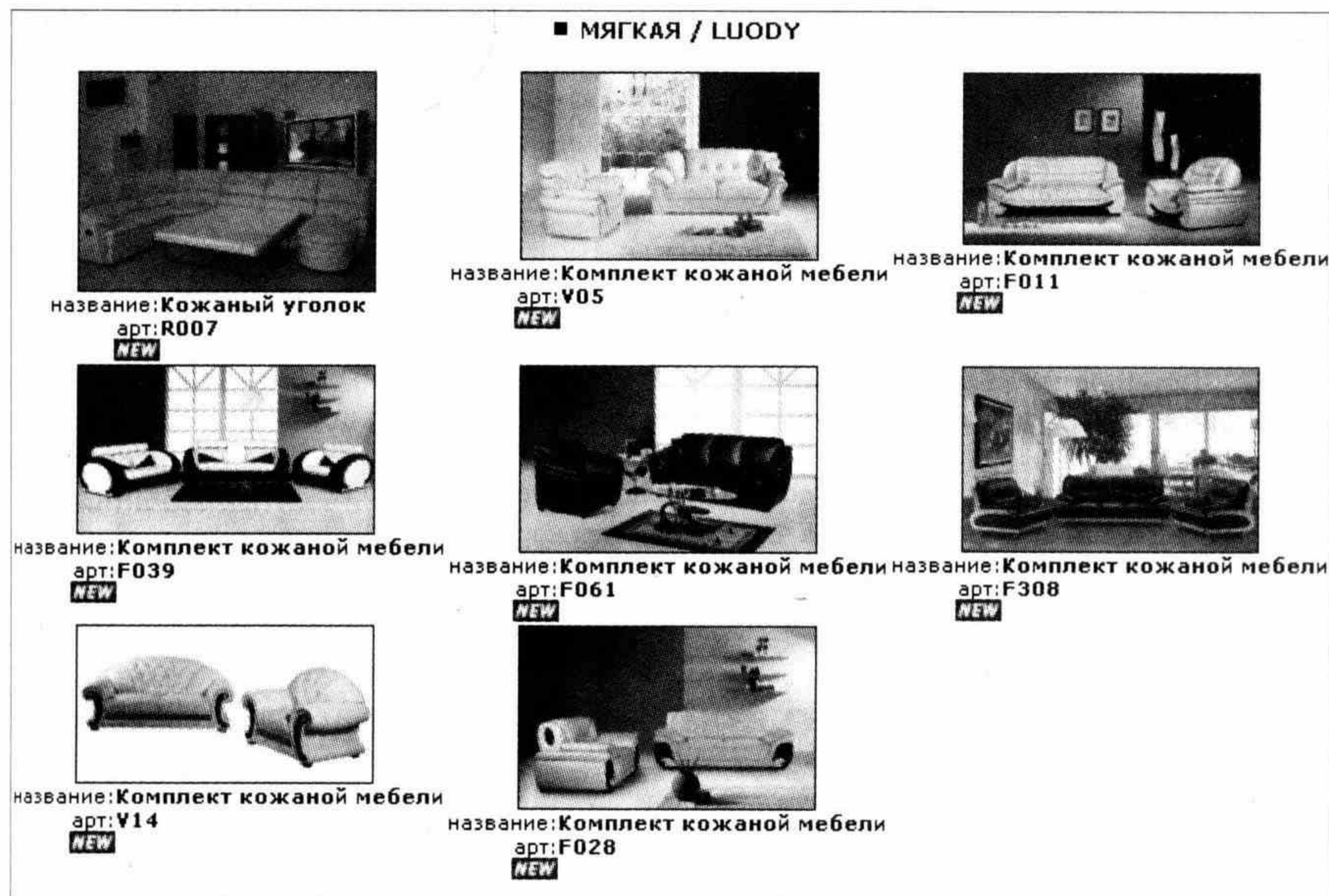


Рис. 15.2. Вывод миниатюр

15.3. Создание водяных знаков

Водяные знаки на банкнотах защищают их от подделки. Водяные знаки на изображениях позволяют защитить изображения от несанкционированного копирования. Однажды с подобной проблемой ко мне обратился один мой хороший знакомый. Он всю ночь сканировал каталог с продукцией, для некоторых товаров делал авторские фотографии, а через пару дней обнаружил свои же изображения на сайте конкурента. Непорядок!

Защитить изображение можно путем нанесения поверх него какой-то надписи. Но если надпись нанести по центру, она будет перекрывать само изображение, что нежелательно. От надписи, нанесенной внизу или сверху (слева или справа — вертикально), очень просто избавиться в любом самом примитивном графическом редакторе.

А вот от водяного знака избавиться будет сложно. Он полупрозрачный, и его можно поместить по центру нашей картинке — таким образом и картинку защитим, и изображенный на ней объект не будет закрыт нашей надписью.

Чтобы не изобретать велосипед заново, в Интернете я нашел класс `watermark`, который и пригодится нам для создания водяных знаков. Рассмотрим листинг 15.5, который использует класс `watermark` для нанесения водяного знака.

Листинг 15.5. Нанесение водяного знака

```

<?php
// подключаем класс
include 'api.watermark.php';

// создаем объект wmark
$wmark = new watermark();
// загружаем основное изображение
$main_img = imagecreatefromjpeg('test.jpg');

// загружаем водяной знак
$watermark_img = imagecreatefrompng('./watermark.png');
// накладываем водяной знак на оригинальную картинку
$res_img = $wmark->create_watermark($main_img, $watermark_img, 66);

// выводим "защищенную" картинку (качество 50%)
header('Content-Type: image/jpeg');
header('Content-Disposition: inline; filename=' . $_GET['src']);
imagejpeg($res_img, '', 50);

?>

```



Рис. 15.3. Изображение с водяным знаком

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *15-5.php*), представленный в каталоге *Glava_15* сопровождающего книгу электронного архива (см. приложение 4). Код класса *watermark* также находится в этом каталоге (файл *api.watermark.php*).

Как видите, все достаточно просто. Сначала создаем объект *\$wmark*, затем загружаем исходную картинку, после этого загружаем картинку, которая будет использоваться в качестве водяного знака. Картинка водяного знака должна быть в формате PNG, без фона (фон — прозрачный), а поверх прозрачного фона наносится любая надпись — водяной знак. Уровень прозрачности водяного знака задается последним параметром метода *create_watermark()*. В нашем случае уровень прозрачности равен 66.

После наложения водяного знака все, что остается нам, — это вывести изображение в браузер. Результат работы сценария показан на рис. 15.3.

15.4. Поддержка графического формата WebP в PHP 5.5

О формате WebP хочется поговорить отдельно. Во-первых, это довольно молодой графический формат, который только набирает обороты. Во-вторых, он разработан компанией Google, поэтому успех ему гарантирован — при поддержке такого гиганта как Google. В-третьих, раз разработчики PHP добавили поддержку этого формата в библиотеку GD, значит, считают его важным и перспективным. Ведь заметьте, что есть много других графических форматов, которые библиотекой GD не поддерживаются — видимо, за ненадобностью.

Формат WebP обеспечивает значительное сжатие графических изображений — как с потерей, так и без потери качества. По сравнению с форматом JPEG, формат WebP обеспечивает то же качество изображения, но позволяет экономить до 25 % дискового пространства. Такая экономия особенно чувствуется при хранении в этом формате фотографий высокого качества (когда размер файла составляет несколько мегабайтов). Подробно описывать здесь особенности формата WebP я не стану, а предоставляю ссылку, пройдя по которой, вы сможете сравнить качество форматов JPEG и WebP, а также занимаемое изображениями в этих форматах дисковое пространство: <http://www.andrewmunsell.com/blog/jpg-vs-webp>.

Полную информацию о формате WebP можно получить на сайте разработчиков: <https://developers.google.com/speed/webp/>.

Конечно, никто не спорит — в настоящее время формат JPEG все еще более популярен и будет таковым оставаться в ближайшие пару лет. Однако формат WebP, учитывая, что он спонсируется Google, нельзя просто так списывать со счета.

Чтобы убедиться, что ваша версия PHP поддерживает формат WebP, вызовите функцию *phpinfo()* и убедитесь, что в секции *gd* прописана поддержка WebP (*WebP support enabled*).

Для вывода изображений в браузер или файл в формате WebP служит следующая функция:

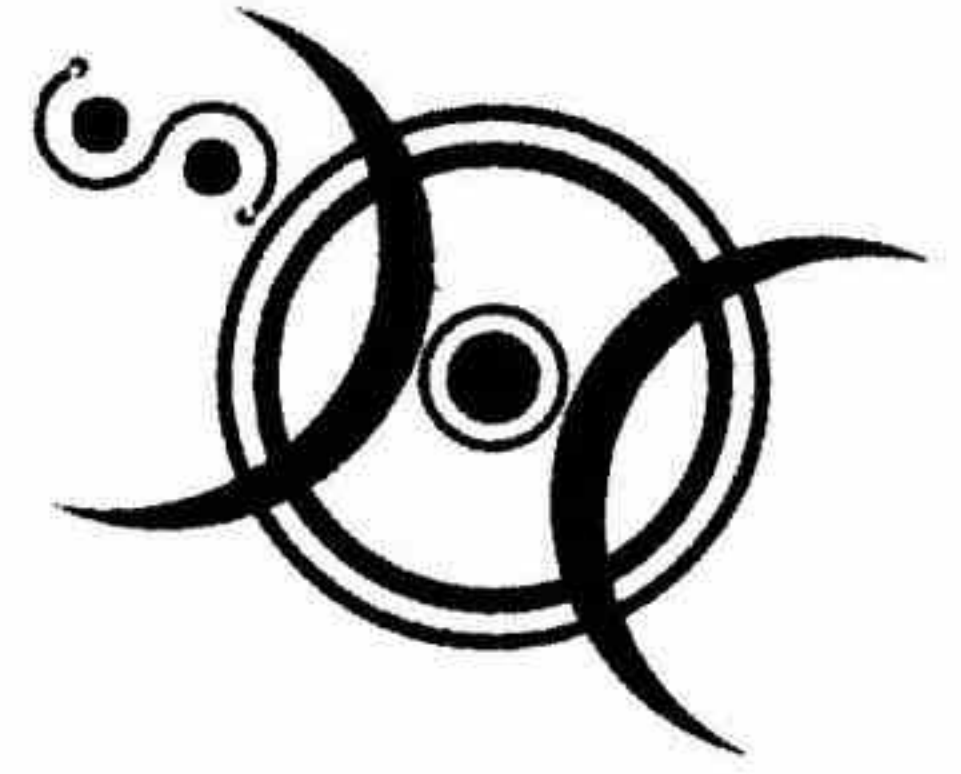
```
bool imagewebp ( resource $image , string $filename )
```

Первый параметр — это уже сформированное средствами PHP изображение. Вторым параметром — имя файла. Если этот параметр не указан или равен `NULL`, будет произведен вывод в браузер.

Загрузить изображение в формате WebP с диска можно функцией `imagecreatefromwebp()`:

```
resource imagecreatefromwebp ( string $filename )
```

Используя эти две функции: `imagewebp()` и `imagecreatefromwebp()` — вы можете загружать и сохранять изображения в формате WebP. А после загрузки изображения применять к нему любые функции из библиотеки GD — например, добавить текст, нарисовать поверх геометрическую фигуру и т. д.



ГЛАВА 16

Работа с сетевыми сокетами в RНР. Сетевые функции

16.1. Еще раз о том, что такое сокет

Все мы знаем, как работать с файлами в RНР. Функцией `fopen()` мы открываем файл. Если файл открыт успешно, то функция возвращает его дескриптор, через который производятся все дальнейшие операции с файлом (чтение, запись).

Сокет очень похож на дескриптор файла, но он является дескриптором соединения, т. е. служит для передачи файлов по сети. Примечательно, что с сетевым соединением в RНР можно работать как с обычным файлом, с помощью стандартных функций для работы с файлами.

Но все равно не следует путать сокеты и файловые дескрипторы (ведь то, что с сокетами осуществляется работа посредством файловых функций, многих вводит в заблуждение). Как мы помним, функция `fopen()` умеет открывать файлы на удаленных серверах, но ведь сокеты используются не для работы с файлами, а для обмена данными с различными серверами. С помощью сокетов можно подключиться к любому ТСР-серверу, например к веб-, FТР-серверу, почтовому и т. д. Конечно, функции RНР и так могут работать с веб-, FТР-серверами, а для работы с почтовыми серверами уже есть готовые классы. Так для чего нужны сокеты? Для реализации нестандартного обмена. Предположим, что вам необходимо обменяться данными с собственным сервером, реализующим свой собственный протокол передачи данных, который знаете только вы. Вот тут вам и пригодятся сокеты — ведь можно со 100%-ной уверенностью сказать, что класса для работы с вашим сервером точно в Сети не появится, если, конечно, его не сделаете вы сами.

16.2. Использование сокетов

Функция `fsockopen()` открывает сокет, т. е. устанавливает соединение с удаленным сервером:

```
fsockopen(string $host, int $port [, int $err] [, string $err_msg])
```

Назначение параметров, думаю, понятно. Первый параметр — это имя или IP-адрес удаленного узла, второй — порт службы. Последние два параметра не обязательны.

Если вы их указали, то в переменную `$err` будет записан код ошибки, а во вторую — текст с сообщением об ошибке.

После того как мы открыли сокет, с ним можно работать с помощью обычных функций для работы с файлами: `fwrite()`, `fread()`, `fgets()`, `fputs()` и др. Для вашего удобства прототипы и описания этих четырех функций сведены в табл. 16.1.

Таблица 16.1. Прототипы функций `fwrite()`, `fread()`, `fgets` и `fputs()`

Функция	Описание
<code>string fgets (resource descriptor [, int length])</code>	Читает строку из файла (сокета), первый параметр — это дескриптор файла (сокета), второй — длина строки в байтах (по умолчанию длина равна 1024 байтам)
<code>int fwrite (resource handle, string string [, int length])</code>	Записывает в файл (сокет), заданный дескриптором, строку, переданную с помощью второго параметра. Если указана длина строки, то запись будет остановлена после того, как будет записано указанное количество байтов (или будет достигнут конец строки). Функция <code>fputs()</code> является псевдонимом функции <code>fwrite()</code>
<code>int fputs (resource handle, string string [, int length])</code>	См. функцию <code>fwrite()</code>
<code>string fread (resource handle, int length)</code>	Читает указанное количество байтов из файла (сокета), заданного дескриптором. Используется преимущественно для чтения бинарных данных

Если вы помните, то при вызове функции `fopen()` нужно было задать режим открытия файла. Сокет же открывается для двухстороннего обмена, т. е. и для чтения, и для записи.

16.3. Пример использования сокетов

Рассмотрим небольшой пример, демонстрирующий общую концепцию работы с сокетами. Мы подключимся к веб-серверу, передадим ему запрос и прочитаем ответ сервера. Как уже было отмечено, сокет открывается как для чтения, так и для записи, а вся работа с ним осуществляется в режиме "запрос-ответ". Вы отправляете запрос серверу и сразу же читаете его ответ, затем ответ необходимо проанализировать (вдруг он содержит сообщение об ошибке) или просто вывести в браузер (листинг 16.1).

Листинг 16.1. Работа через сокет с протоколом HTTP

```
<?
// открываем сокет, устанавливаем соединение с сервером localhost,
// порт 80
$sock = fsockopen("localhost",80);
```



```
// отправляем команду GET — получить index.html
fputs($sock, "GET /index.html HTTP/1.0\n\n");

// читаем ответ сервера (1024 байтов) и сразу выводим его в браузер
while (!feof($sock))
    echo fgets($sock);

// закрываем сокет
fclose($sock);
?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *16-1.php*), представленный в каталоге *Glava_16* сопровождающего книгу электронного архива (см. приложение 4).

Для того чтобы обмениваться информацией с сервером, вам нужно знать протокол обмена. В Интернете можно найти специальные документы — запросы комментариев (RFC, Request For Comments), в которых описаны все стандартные протоколы. В табл. 16.2 приведены номера документов RFC и ссылки на описание протоколов.

Таблица 16.2. Некоторые RFC-документы

Номер	Описание	Ссылка
2068, 2616	Протокол HTTP/1.1. Документ 2616 — более свежий, чем 2068	http://www.faqs.org/rfcs/rfc2616.html
1870	Протокол SMTP	http://www.faqs.org/rfcs/rfc1870.html
1939	Протокол POP3	http://www.faqs.org/rfcs/rfc1939.html

ПРИМЕЧАНИЕ

Архив с документами RFC 2068, 2616, 1870, 1939 можно скачать по ссылке <http://dkws.org.ua/mybooks/rfc.rar>.

Рассмотрим еще один пример использования сокетов. Напишем сценарий, который подключится к серверу **whois.ripe.net**, передаст интересующий нас IP-адрес и выведет информацию об этом IP-адресе, полученную от **whois.ripe.net** (листинг 16.2). Вызывать сценарий нужно так:

`http://<сервер>/soc.php?ip=<IP-адрес>`

Листинг 16.2. Сценарий soc.php

```
<?php

if ($ip!="")
{
    // подключаемся к whois.ripe.net, порт 43
    $sock = fsockopen ("whois.ripe.net", 43, $errno, $errstr);
```



```
descr:      server.ru
descr:      Moscow, Russia
country:    RU
admin-c:    VG659-RIPE
tech-c:     VG659-RIPE
status:     ASSIGNED PA
mnt-by:     AS3216-MNT
source:     RIPE # Filtered

person:     Ivan Ivanov
address:    19, Pushechaya str., Moscow, Russia
e-mail:     Ivanov@mail.ru
remarks:    fax-no:      +7 095 7777777
fax-no:     +7 095 7777777
remarks:    phone:       +7 095 7777777
phone:      +7 095 7777777
nic-hdl:    VG659-RIPE
source:     RIPE # Filtered
remarks:    modified for Russian phone area changes
```

```
% Information related to '192.168.0.0/18AS3216'
```

```
route:      192.168.0.0/18
descr:      SOVAM DELEGATED BLOCK-1
origin:     AS3216
mnt-by:     AS3216-MNT
source:     RIPE # Filtered
```

ПРИМЕЧАНИЕ

Информация в листинге 16.3 (домен, имя владельца, IP-адреса, номера телефонов) вымышленная.

16.4. Блокирующий и неблокирующий режимы сокета

Существуют два режима работы сокета: блокирующий и неблокирующий. В первом режиме функции чтения из сокета (`fgets()` и `fread()`) будут ждать до тех пор, пока не прочитают данные из сокета. Во втором, если данных в соquete нет, то функции чтения завершают свою работу.

Для переключения в блокирующий режим нужно вызвать функцию `socket_set_blocking()`, указав дескриптор соединения и передав `true` в качестве второго параметра:

```
socket_set_blocking($sock, TRUE);
```

Отключить блокирующий режим можно так:

```
socket_set_blocking($sock, FALSE);
```


ПРИМЕЧАНИЕ

Функция `socket_set_blocking()` появилась только в версии PHP 4.3. Если у вас версия PHP младше, то данной функции у вас не будет.

16.5. DNS-функции

Для разрешения IP-адреса в доменное имя и наоборот служат две следующие функции:

```
string gethostbyaddr ( string $ip_address )  
string gethostbyname ( string $hostname )
```

Первая функция возвращает доменное имя узла по заданному IP-адресу, например:

```
<?php  
// получаем IP-адрес удаленного пользователя, преобразуем в имя и выводим  
$hostname = gethostbyaddr($_SERVER['REMOTE_ADDR']);  
  
echo $hostname;  
?>
```

Вторая функция возвращает IP-адрес по заданному доменному имени:

```
<?php  
$ip = gethostbyname('www.server.ru');  
  
echo $ip;  
?>
```

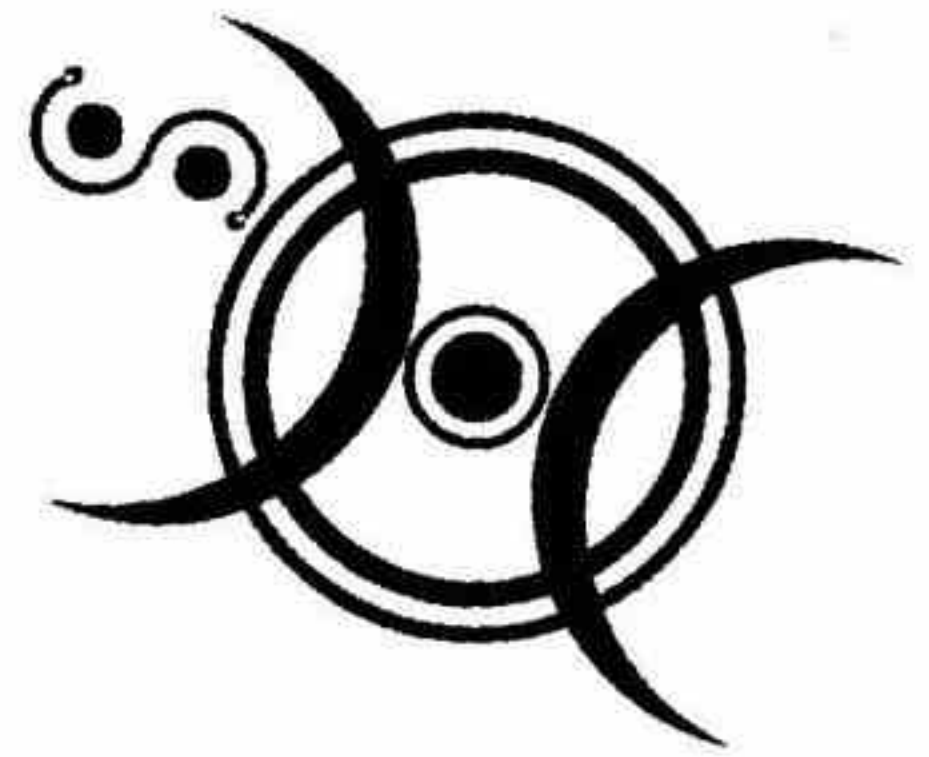
Есть еще и третья DNS-функция:

```
array gethostbyname1 ( string $hostname )
```

Дело в том, что одному и тому же доменному имени могут соответствовать несколько IP-адресов. С помощью функции `gethostbyname1()` мы можем получить все эти IP-адреса. Вот пример использования этой функции:

```
<?php  
  
$hosts = gethostbyname1('www.example.com');  
  
print_r($hosts);  
  
?>
```


ГЛАВА 17



Собственные функции

17.1. Зачем нужны собственные функции?

До сих пор мы рассматривали встроенные функции PHP, а теперь настало время поговорить о создании собственных функций. Спрашивается, зачем нужны собственные функции? Ответ прост — если вы не можете найти нужную вам функцию среди стандартных функций PHP, тогда вам остается только разработать собственную.

Например, в PHP нет функции для проверки правильности адреса электронной почты. Используя функции для работы с регулярными выражениями, вы можете с легкостью создать такую функцию. Вот, например:

```
function is_email($email) {  
if (! preg_match( '/^[A-Za-z0-9!#$%&\'*+-<=?^_`{|}~]+@[A-Za-z0-9-]+(\.[A-Za-z0-9-]+)+[A-Za-z]$/', $email)) {  
return false;  
} else {  
return true;  
}  
}
```

Конечно, если вам нужно проверить правильность адреса только один раз и в одном сценарии, то можно обойтись и без собственной функции, а просто вызвать функцию `preg_match()` и проанализировать ее результат. Но если вам нужно вызывать этот код несколько раз в разных частях сценария и даже в разных файлах, то целесообразно создать свою функцию и вынести ее в отдельный PHP-сценарий, который можно будет подключить к любому сценарию, где требуется функция проверки почты.

17.2. Особенности функций в PHP

PHP — довольно гибкий язык программирования, и эта гибкость проявляется во всем, в том числе и при объявлении функций.

- ☐ Вы можете объявить произвольное число параметров и вызвать функцию с произвольным числом параметров. Отсутствующие аргументы будут заменены аргументами по умолчанию, которые вы укажете при объявлении функции.
- ☐ Вы можете использовать как локальные, так и глобальные переменные. Области видимости переменных будут рассмотрены далее.
- ☐ Функции могут вкладываться друг в друга.
- ☐ Возвращаемое значение задается инструкцией `return`.
- ☐ Функция может возвращать значение любого типа.
- ☐ Функция может изменять значения переменных, которые были переданы в качестве значений параметров.

Все эти особенности будут рассмотрены в данной главе.

17.3. Объявление функции

Пользовательская функция в PHP объявляется так:

```
function имя([параметр1[=значение1], ..., параметрN[=значениеN]])  
{  
    операторы  
}
```

Функцию можно объявить в любой части сценария, но до места первого использования. Имя функции должно соответствовать следующим требованиям:

- ☐ имя функции должно содержать только английские символы, цифры и знак подчеркивания;
- ☐ в имени не должно быть пробелов и других пробельных символов;
- ☐ имя должно начинаться с буквы, т. е. `f1()`, но не `1f()`.

К именам параметров требования такие же, как и к именам переменных. Функция вообще может не содержать параметров.

При объявлении функции вы можете задать параметры по умолчанию, например:

```
function sum($a, $b=0)  
{  
    return $a + $b;  
}
```

Вызвать функцию `sum()` можно так:

```
$num=7;  
echo sum(5);           // результат 5  
echo sum(5, 5);        // результат 10  
echo sum($num);        // результат 7  
echo sum($num, 5);     // результат 12
```


Другими словами, в качестве аргументов функции вы можете передавать как значения, так и переменные — на результат функции это никак не повлияет. Параметры по умолчанию вы можете не указывать, тогда функция будет использовать заданные вами значения по умолчанию.

17.4. Области видимости функции

Для начала уточним понятия локальных и глобальных переменных. Глобальной называется переменная, объявленная вне функции. Локальной называется переменная, объявленная в теле функции.

Предположим, у вас есть глобальная переменная `$i`, служащая в качестве счетчика цикла `for`, и есть локальная переменная с таким же именем и используемая для этих же целей, но внутри функции `f1()`:

```
function f1()  
{  
  for ($i=0; $i < 3; $i++) echo $i;  
}
```

```
for ($i=0; $i < 3; $i++) f1();
```

Приведенный код будет прекрасно работать, поскольку функция `f1()` использует свою собственную переменную `$i`, которая не имеет никакого отношения к глобальной переменной с таким же именем.

Но если у вас есть необходимость обратиться к глобальной переменной из функции, то придется использовать инструкцию `global` в теле функции:

```
global <имя_переменной>;
```

Например:

```
global $i;
```

17.5. Вложенность функций

Вы можете объявить одну функцию внутри другой функции, например:

```
function f1()  
{  
  
  echo "Это функция f1 <br>\n";  
  
  function f2()  
  {  
    echo "Это функция f2 <br>\n";  
  }  
  
}
```


Однако результат будет совсем не таким, как вы ожидали. Можно предположить, что функция `f2()` является локальной функцией функции `f1()` и будет доступна только в пределах функции `f1()`. Но на самом деле PHP все равно, где объявлена функция. Все функции объявляются как глобальные и будут доступны в любой части сценария. Например, за пределами функции `f1()` вы можете вызвать обе функции:

```
f1();  
f2();
```

Вывод будет таким:

```
Это функция f1 <br>  
Это функция f2 <br>
```

Поскольку никакой разницы, где объявлять функцию, нет, лучше не использовать вложенность функций, иначе вы ненароком можете создать непрямую рекурсию. А если вы не предусмотрите условие выхода из такой рекурсии, сценарий заикнется. Хорошо, что интерпретатор прервет его выполнение через 180 секунд!

Уточним, что такое рекурсия. Прямая рекурсия (или просто рекурсия) — это явление, когда функция вызывает саму себя. Рассмотрим простейшую функцию, вычисляющую факториал числа `$n`:

```
function f($n)  
{  
    if ($n==0) return 1;  
    else return $n*f($n-1);  
}
```

Заметьте, что мы предусмотрели условие выхода. Если `$n = 0`, то функция прекращает рекурсию. Поскольку значение `$n` уменьшается при каждом новом вызове функции, то рано или поздно `$n` станет равным 0.

А вот если бы мы забыли уменьшить `$n`, то наш сценарий бы заикнулся:

```
// не запускайте эту функцию!!!  
function f($n)  
{  
    if ($n==0) return 1;  
    else return $n*f($n);  
}
```

Непрямая рекурсия — это когда одна функция запускает вторую функцию, а она, в свою очередь, вызывает первую функцию:

```
function f1()  
{  
  
    echo "Это функция f1 <br>\n";
```



```
f2();  
  
function f2()  
{  
echo "Это функция f2 <br>\n";  
f1();  
  
}
```

Если вызвать любую из этих функций, произойдет заикливание сценария, поскольку мы не предусмотрели условие выхода из непрямой рекурсии.

Рекурсия — мощный инструмент, но в то же время он довольно опасен в неопытных руках. Вообще любой рекурсивный алгоритм можно преобразовать в аналогичный нерекурсивный, т. е. без рекурсии можно спокойно жить. Учитывая всю коварность рекурсии, лучше от нее отказаться.

17.6. Переменное число аргументов

Далеко не всегда мы знаем, сколько параметров понадобится функции. PHP позволяет написать действительно универсальную функцию с переменным числом параметров. Вы можете передать такой функции один параметр, два параметра, сто параметров. Получить доступ ко всем параметрам функции позволяет функция `func_get_args()`. Вот пример ее использования:

```
function f()  
{  
foreach(func_get_args() as $argn => $argv)  
echo "Параметр $argn Значение $argv <br>\n";  
}
```

Вызвать функцию можно так:

```
f("аргумент", 5, 10);
```

Вывод функции будет таким:

```
Параметр 0 Значение аргумент  
Параметр 1 Значение 5  
Параметр 2 Значение 10
```

ВНИМАНИЕ!

Как видите, мы передали функции параметры разных типов: одну строку и два числа. Поэтому перед использованием аргументов функции нужно установить их тип. Получить количество переданных параметров можно с помощью функции `func_num_args()`.

17.7. Передача массивов в качестве параметров

Вы можете передать массив в качестве аргумента функции. Рассмотрим функцию, определяющую индекс минимального элемента списка:

```
function array_min($Arr)
{

    $min = $Arr[0];
    $min_ind = 0;

    foreach($Arr as $k => $v)
        if ($v < $min) { $min_ind = $k; }

    return $min_ind-1;

}
```

Как видите, работать с параметрами-массивами так же просто, как и с обычными параметрами.

17.8. Генераторы

Генератор — это функция, возвращающая не одно значение, а последовательность значений. Обратите внимание: не массив значений, а именно последовательность значений. При каждом новом вызове функции-генератора возвращается следующее значение из последовательности.

Напишем простейшую функцию-генератор:

```
function squares($start, $end, $step = 1) {
    for ($i = $start; $i <= $end; $i += $step) {
        yield $i*$i;
    }
}

foreach (squares(1, 10) as $n) {
    echo $n . " ";
}
```

Генератор `squares` генерирует последовательность квадратов чисел от `$start` до `$end` с шагом `$step`. Каждое значение возвращается не инструкцией `return`, а инструкцией `yield`. Обратите внимание, что генератор не возвращает всю последовательность сразу. При каждом вызове генератора возвращается следующее значение. Наш генератор выведет:

1 4 9 16 25 36 49 64 81 100

17.9. Особенности использования функции *empty()*

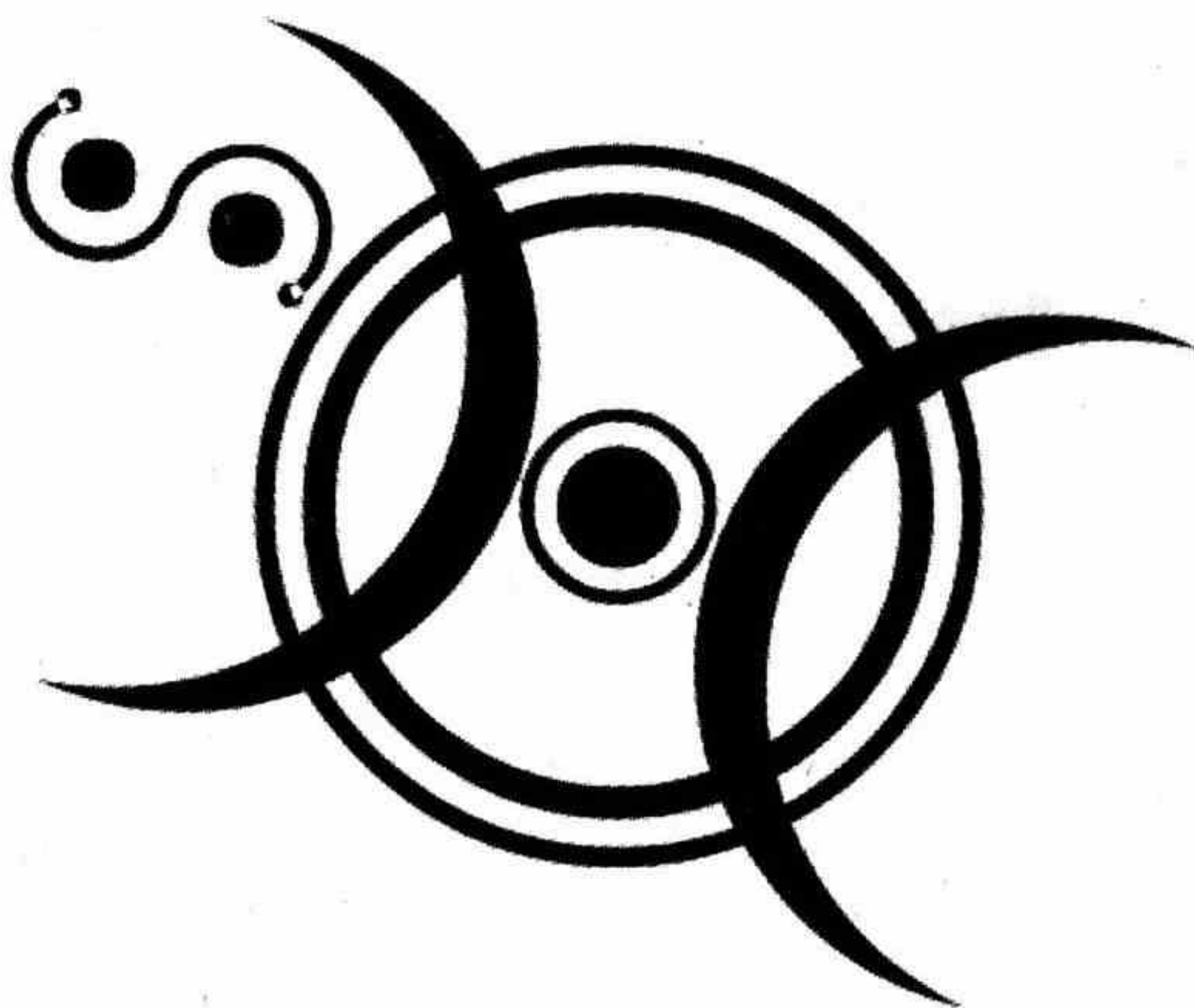
В PHP 5.5 большое внимание уделили "компактизации" кода — другими словами, разработчики всеми силами старались сделать код компактнее. Примерами тому могут служить множество нововведений: и ключевое слово *finally*, и разыменовывание массивов и строк (правда, разыменовывание массивов впервые появилось в версии 5.4). Из соображений "компактизации" была усовершенствована и функция *empty()*. Напомню, она возвращает *true*, если ей передан пустой объект. Представим, что есть некая функция *A()*, которая может вернуть пустой результат (например, она возвращает результат запроса к базе данных). Ранее, чтобы проверить, что результат не пуст, нужно было писать вот такой код:

```
$res = A();  
if (!empty($res)) {  
    // результат не пустой }
```

Сейчас же можно упростить этот код до следующего:

```
if ( !empty(A()) ) {  
    // результат не пустой }
```

Да, код стал компактнее, но не намного. И дело даже не в том, что мы избавились от строки *\$res = A();* — все равно ее придется вызвать! Но уже в теле блока *if*. Судите сами. Нам важно знать, не пуст ли результат. Так? Но еще более важно выполнить какие-то действия над результатом — например, вывести результат в браузер или еще как-то его обработать. Однако не присвоив результат временной переменной (в нашем случае *\$res*), мы не сможем с ним работать! Допустим, мы убедились, что результат не пуст, а что дальше? Нам придется еще раз вызывать эту функцию, чтобы опять получить тот же результат и обработать его. Получается, что вместо оптимизации мы создаем дополнительную нагрузку на сервер баз данных (поскольку функция *A()*, осуществляющая запрос к БД, выполняется дважды). Это еще одно сомнительное нововведение в версии 5.5, но не сказать о нем я не мог.

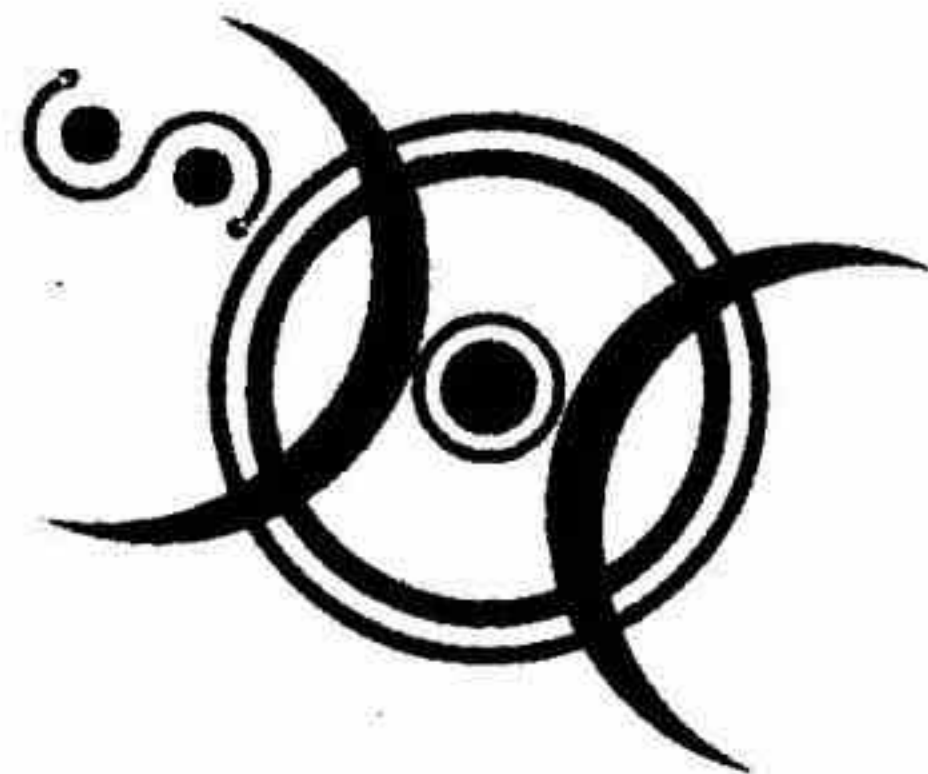


I. РАЗДЕЛ 5

Серверы баз данных MySQL 6 и SQLite

Глава 18.	Основы SQL
Глава 19.	Функции для работы с MySQL
Глава 20.	Альтернативная база данных SQLite
Глава 21.	Полезные приемы при работе с MySQL 6

ГЛАВА 18



Основы SQL

18.1. Немного истории

В далеком 1970-м компания IBM разработала экспериментальную систему управления базами данных System R. В ее основе лежал язык SEQUEL (Structured English Query Language, структурированный английский язык запросов). Чуть позже этот язык по юридическим и политическим соображениям был переименован в просто SQL (Structured Query Language).

Разработкой SQL занимались Чэмбэрлин (Chamberlin) и Рэй Бойс (Ray Boyce). Кроме них над SQL вели работу Пэт Селинджер (Pat Selinger), которая разрабатывала оптимизатор, и Рэймонд Лори (Raymond Lorie), создававший компилятор запросов. Их главной целью было создание языка запросов к базе данных, с которым мог бы разобраться любой пользователь, не имеющий никаких навыков программирования.

В 1986 году Американский национальный институт стандартизации (ANSI, American National Standards Institute) принял первый стандарт языка SQL. Принятие стандарта должно было способствовать дальнейшему развитию языка запросов. В 1987 году стандарт был уточнен (появился так называемый SQL level 1). Еще одно изменение в стандарте произошло в 1989 году (SQL level 2).

Развитие и совершенствование систем управления базами данных послужило причиной принятия в 1992 году следующего стандарта (он назывался ANSI SQL-92 или просто SQL-2). Последующие стандарты были приняты в 1999 (SQL-99) и 2003 годах (SQL-3).

18.2. Преимущества SQL

Как уже было отмечено, основное преимущество SQL — простота его использования. Предположим, что SQL нет, но нам необходима база данных. Что делать? Сначала нужно разработать формат двоичного файла (с двоичным файлом операции поиска выполняются быстрее, чем с текстовым): продумать заголовок файла, разделители полей и индексы. На все это уйдет много времени. Еще больше времени

займет разработка набора функций, которые будут работать с нашим двоичным форматом. Спустя какое-то время этот набор функций мы создадим. Затем напишем приложение с их использованием. Допустим, нам потребуется просмотреть весь файл данных и выбрать из него записи с определенным значением поля — например, чтобы поле "Количество" было больше нуля. Алгоритм обработки предполагается следующим — нужно в цикле пересмотреть все записи и вывести только те, которые не содержат ноль в поле "Количество". Не очень удобно, но все же выполнимо. Но вот незадача — а что делать, если возникнет необходимость радикально изменить формат файла? Ведь при разработке мы могли допустить ошибку, точнее, недоработку, которая "всплыла" несколько месяцев спустя. Тогда нам придется заново переделывать весь наш набор функций. Причем делать все так, чтобы это не затронуло работу уже существующих приложений, иначе нужно будет переписывать и их.

В случае с SQL все гораздо проще. Вам не придется изобретать велосипед заново: забудьте о формате данных и наборе функций — для вас этих проблем просто не существует. Все, что вам необходимо знать, — это то, что есть база данных, которая выполнит любой ваш приказ, сформулированный по правилам языка SQL. Программировать вам тоже ничего не понадобится — зачем циклы, если можно попросить SQL: "Просмотри все записи и выбери те, у которых поле «Количество» равно нулю". Даже если возникнет необходимость перейти на другую систему управления базами данных, SQL-запросы останутся прежними.

Таким образом, преимущества SQL следующие:

- независимость от системы управления базами данных — благодаря тому, что приняты стандарты SQL, его запросы будут одинаково выполняться что в Oracle, что в MySQL. Конечно, есть определенные отличия синтаксиса некоторых СУБД, но они, как правило, незначительны;
- полноценность SQL как языка управления данными — с помощью SQL можно не только производить выборку данных, но и полноценно управлять ими: добавлять новые данные, изменять и удалять уже имеющиеся, создавать базы данных. Выходит, что кроме знания SQL вам для работы с СУБД не понадобятся никакие другие дополнительные сведения.

18.3. Как выглядят запросы?

Поскольку SQL является структурированным языком запросов, все запросы в нем имеют четкую структуру, которой рекомендуется придерживаться, если вы хотите, чтобы ваш код был удобен для чтения. Вот пример запроса SELECT (выборка данных):

```
SELECT Q_NO, NAME, PHONE  
FROM CUSTOMERS  
WHERE CITY = 'KIEV';
```

Как видите, не зря SQL когда-то назывался SEQUEL — уж очень он похож на обычный английский язык. Понять суть этого запроса можно, даже не имея пред-

ставления об SQL. Очевидно, что здесь запрос выбирает информацию обо всех клиентах, проживающих в Киеве.

В конце каждого запроса обязательна точка с запятой. Но когда вы будете передавать запросы в функцию PHP, разрешается точку с запятой не ставить и записывать запросы в одну строку. Поступайте так, как вам будет удобнее:

```
// можно так
$q = 'select * from clients';
```

```
// а можно так
$q = 'select *
from clients';
```

```
$r = mysql_query($q);
```

18.4. Что такое база данных?

Что такое система управления базами данных (СУБД), понятно из самого названия. А вот что такое *база данных*? База данных — это набор связанных (не всегда) между собой таблиц. С технической точки зрения, базой данных может быть обычный каталог на диске, в котором размещены файлы таблиц, или же один большой двоичный файл, содержащий всю информацию, — тут все зависит от реализации формата базы данных. Например, в MySQL базой данных является каталог с файлами таблиц, а вот в InterBase — это один большой файл.

Теперь поговорим о связи таблиц. Отличительным признаком реляционной (от англ. *relations* — отношения) базы данных является как раз "набор взаимосвязанных таблиц". Предположим, что у нас есть база данных интернет-магазина, содержащая информацию о товарах, клиентах и заказах. В первой таблице содержится информация о товарах: номер товара, описание и цена. Во второй — информация о клиентах: номер клиента, фамилия, электронный и почтовый адреса. А вот третья таблица связывает две предыдущих — в ней находится информация о заказах, т. е. номер клиента, номер товара и количество (это минимальный набор полей).

В теории так оно и есть — все таблицы базы данных связаны между собой. Но на практике наблюдается совершенно противоположная ситуация. Количество баз данных часто ограничивается. Нет, не самой СУБД, а хостинг-провайдерами. Например, для минимальных хостинг-планов предоставляется всего одна база данных. А ведь хочется установить и форум, и интернет-магазин, и еще многое другое. Вот и получается, что в одной базе данных окажется несколько наборов таблиц, связанных только внутри себя, но не с другими наборами. Например, мы создали в одной и той же базе данных два набора таблиц: один для форума, а второй — для интернет-магазина. Таблицы форума будут связаны между собой, но они не будут иметь никакого отношения к таблицам интернет-магазина. Поэтому базу данных лучше рассматривать просто как набор таблиц без учета связей между ними.

Теперь поговорим о более "мелких" элементах базы данных — таблицах. Таблица — это набор записей одной структуры. Структура записей, т. е. набор полей, задается при создании таблицы. Запись — это набор полей, содержащих связанную

информацию. Например, `C_NO`, `NAME`, `CITY`, `EMAIL` — данный набор полей содержит информацию о человеке: его персональный номер (`C_NO`), имя (`NAME`), город (`CITY`), электронный адрес (`EMAIL`). Связь заключается в том, что вся информация в одной записи относится к одному и тому же объекту. Например, запись

```
1 Иванов Москва ivanov@i.ru
```

говорит о том, что персональный номер Иванова — 1, Иванов проживает в Москве и его электронный адрес — `ivanov@i.ru`.

Поле — это базовый элемент базы данных, из полей формируются записи. Поле содержит информацию ранее заданного типа (тип поля задается при создании таблицы).

Кроме полей, записей и таблиц в базе данных могут быть индексы (ключи), они не менее важны, чем поля и таблицы. Индексы используются для быстрого поиска нужной записи и для управления порядком отображения записей. Существуют первичный и вторичный индексы.

□ *Первичный индекс* (`primary key`, `primary index`) управляет порядком отображения записей в таблице. Первичный индекс строится по одному полю, значение которого должно быть уникальным. В приведенном примере на звание первичного индекса претендует поле `C_NO`, содержащее уникальный номер человека в системе. По полю `NAME` первичный индекс строить нельзя, поскольку Ивановых на просторах нашей страны очень много, и в таблицу могут попасть несколько Ивановых, — как тогда понять, кто есть кто? А вот номер `C_NO` будет уникальным в нашей системе (об этом может даже позаботиться сама система), поэтому `C_NO` — идеальное поле для первичного индекса. Первичный индекс обычно задается при создании таблицы.

□ *Вторичный индекс* (`secondary index`) не обязательно должен быть уникальным, к тому же он может строиться по нескольким полям сразу. Вторичные индексы, как правило, служат для связывания таблиц. Вторичный ключ еще иногда называют внешним ключом.

Вот теперь можно приступить к рассмотрению основ SQL. Сразу отмечу, что приведенного далее материала вполне хватит для разработки даже сложных проектов, но он никак не претендует на полноту описания SQL.

18.5. Создание таблиц

Для создания таблицы предназначен SQL-оператор `CREATE`:

```
CREATE TABLE Имя_таблицы
```

```
(
```

```
Имя_поля1          Тип          Модификатор,
```

```
...
```

```
Имя_поляN          Тип          Модификатор,
```

```
[Первичный    ключ,]
```

```
[Внешний      ключ]
```

```
)
```


Оператор `CREATE` в SQL используется для создания не только таблиц, но и других объектов базы данных. В этой книге мы будем говорить только о создании таблиц, но для интересующихся в табл. 18.1 описаны альтернативные операторы `CREATE`, позволяющие создавать другие объекты базы данных.

Таблица 18.1. Синтаксис оператора `CREATE`

Синтаксис оператора <code>CREATE</code>	Объект
<code>CREATE DATABASE <имя></code>	Создает базу данных. Создать базу данных не всегда возможно. Во-первых, у вас должны быть права на ее создание. Во-вторых, многие хостинг-провайдеры устанавливают лимит на количество баз данных
<code>CREATE DEFAULT <имя></code> <code>AS <выражение></code>	Создает константу по умолчанию
<code>CREATE FUNCTION <имя></code> <code>RETURNS</code> <code><значение> AS <операторы_SQL></code>	Создает пользовательскую функцию
<code>CREATE INDEX <имя></code> <code>ON <таблица_или_представление></code> <code><индексируемые_поля></code>	Создает индекс (в таблице или представлении)
<code>CREATE PROCEDURE <имя></code> <code>AS <операторы_SQL></code>	Создает пользовательскую процедуру
<code>CREATE RULE <имя></code> <code>AS <выражение></code>	Создает правило
<code>CREATE SCHEMA AUTHORIZATION</code> <code><владелец> <объекты></code>	Таблицы, представления и разрешения будут созданы как один объект
<code>CREATE TABLE <имя> (определение таблицы)</code>	Создает таблицу
<code>CREATE VIEW <имя></code> <code>AS <оператор_выборки></code>	Создает представление

Вернемся к созданию таблицы. Для определения поля нужно указать его имя, тип и модификатор поля. Допустимые типы данных приведены в табл. 18.2.

Таблица 18.2. Часто используемые типы данных

Тип	Описание
<code>TINYINT</code>	Целый тип, может принимать значения из диапазона –128...+127
<code>SMALLINT</code>	Целый тип, может принимать значения из диапазона –32 768...+32 767
<code>MEDIUMINT</code>	Целый тип, может принимать значения из диапазона –8 388 608...+8 388 607

Таблица 18.2 (окончание)

Тип	Описание
INT	Целый тип, может принимать значения из диапазона –2 147 483 648...+2 147 483 647
BIGINT	Целый тип, может принимать значения из диапазона –9 223 372 036 854 775 808...+9 223 372 036 854 775 807
FLOAT (<Д>,<З>) [UNSIGNED]	Вещественный тип, малая точность. Для каждого вещественного типа можно указать его длину (Д), т. е. количество знакомест, в которых будет размещено число, а также количество знаков (З) после десятичной запятой, которое будет учитываться. Модификатор UNSIGNED говорит о том, что знак числа учитываться не будет
DOUBLE (Д,З) [UNSIGNED]	Вещественный тип, наибольшая точность
NUMERIC (Д,З) [UNSIGNED]	Дробное число, хранящееся в виде строки
DECIMAL (Д,З) [UNSIGNED]	Псевдоним для NUMERIC
REAL (Д,З) [UNSIGNED]	Псевдоним для DOUBLE
VARCHAR (N)	Строка, N — число символов (не более 255)
TINYTEXT	Небольшой текст, максимум 255 символов
TEXT	Текст (до 64 Кбайт)
MEDIUMTEXT	Текст, максимум 16 777 215 символов
LONGTEXT	Текст, максимум 4 294 967 295 символов
BLOB	Используется для хранения бинарных данных, максимальный размер которых — 64 Кбайт. Бинарные данные не перекодируются "на лету", если включена перекодировка символов
MEDIUMBLOB	Используется для хранения бинарных данных, максимальный размер — 16 777 215 символов
LOBLOB	Используется для хранения бинарных данных, максимальный размер — 4 294 967 295 символов
DATE	Дата в формате ГГГГ-ММ-ДД
TIME	Время в формате ЧЧ:ММ:СС
TIMESTAMP	Дата и время в формате timestamp, выводится в виде ГГГГММДДЧЧММСС. Формат не очень удобный, поэтому лучше использовать DATETIME
DATETIME	Дата и время в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС

После типа нужно указать модификатор поля (хотя это необязательно):

- ☐ NOT NULL — поле не может содержать пустое значение, т. е. при добавлении записи нужно обязательно указать значение этого поля;

- PRIMARY KEY — поле будет первичным ключом, хотя индексы лучше определять по-другому (об этом поговорим чуть позже);
- AUTO_INCREMENT — при добавлении новой записи значение этого поля будет автоматически увеличено на единицу (данный модификатор допустим только для целых типов);
- DEFAULT — задает значение по умолчанию, если значение поля при добавлении записи не задано, например DEFAULT 0.

Рассмотрим пример создания таблицы (листинг 18.1).

Листинг 18.1. Пример создания таблицы

```
CREATE TABLE friends
(
  id            int            auto_increment,
  name          varchar(50)    not null,
  email         varchar(30)    not null,
  comment text,
  primarykey (id)
);
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *18-1.sql*), представленный в каталоге *Glava_18* сопровождающего книгу электронного архива (см. приложение 4).

Итак, наша таблица `friends` будет содержать четыре поля:

- `id` — может принимать целые значения (это будет номер записи), значения поля увеличиваются автоматически (`auto_increment`). Это сделано для нашего удобства — при добавлении новой записи нам не нужно будет помнить номер последней, т. к. его MySQL укажет за нас;
- `name` — текстовое поле для имени вашего друга, максимальная длина поля 50 символов;
- `email` — тоже текстовое поле, но с длиной строки 30 символов;
- `comment` — поле типа TEXT, может содержать небольшой текст объемом до 64 Кбайт.

Еще раз поговорим о поле `id`. Это поле объявлено как поле-счетчик (`AUTO_INCREMENT`), оно же будет первичным индексом. Если индексом должно быть не целочисленное поле или же не задан модификатор `AUTO_INCREMENT`, то следует обязательно задать модификатор `NOT NULL`, поскольку индекс не может содержать пустого значения.

18.6. Добавление записей в таблицу

Для добавления записей служит оператор INSERT:

```
INSERT INTO Имя_таблицы [ (Список полей) ]  
VALUES (Список констант);
```

Рассмотрим пример добавления записи в нашу таблицу:

```
INSERT INTO friends  
VALUES (0, 'Denis', 'denis@host.ru', 'some comment');
```

Поскольку поле `id` является счетчиком, то вместо значения можно просто указать 0. Поля `NAME` и `EMAIL` обязательны для заполнения. Список полей мы не указывали, чтобы упростить запрос. Но можно указать поля явно (причем порядок полей может быть не такой, какой был указан при создании таблицы):

```
INSERT INTO friends (COMMENT, EMAIL, NAME)  
VALUES ('some comment', 'denis@host.ru', 'Denis');
```

Главное, чтобы указанные значения соответствовали полям.

Для вставки даты удобно использовать встроенную SQL-функцию `TO_DATE`:

```
TO_DATE('01/02/77', 'DD/MM/YY')
```

Например:

```
INSERT INTO BIRTHDAY (ADATE, NAME)  
VALUES (TO_DATE('01/02/77', 'DD/MM/YY'), 'Andrew');
```

18.7. Обновление записей

Обновление записей осуществляется оператором `UPDATE`, синтаксис которого следующий:

```
UPDATE таблица  
SET список_полей_и_значений  
WHERE условие;
```

Список полей и значений задается так:

поле = значение

Например:

```
UPDATE friends  
SET name='Igor', email='777@host.ru'  
...
```

В случае с оператором `UPDATE` мы обязательно должны указать условие, иначе заданные нами значения полей будут установлены для всех записей, а это нежелательно. Чтобы изменить запись, мы должны ее идентифицировать. Поскольку у нас есть ключ, то сделать это достаточно просто — нужно указать значение ключевого поля:


```
UPDATE friends
SET name='Igor', email='777@host.ru'
WHERE id=1;
```

Этот запрос найдет запись, для которой поле `id` равно единице, и изменит значения ее полей `name` и `email`. Поле `comment` останется без изменений, как и поле `id`.

Совсем другая была бы ситуация, если бы поле `id` не было ключевым, т. е. могло содержать повторяющиеся значения. Тогда теоретически в нашей таблице могло существовать две записи с одинаковым значением этого поля. Поскольку нам нужно обновить только одну запись, а не все, которые содержат такое же значение поля `id`, следует указать дополнительное условие — `LIMIT`:

```
UPDATE friends
SET name='Igor', email='777@host.ru'
WHERE id=1 LIMIT 1;
```

Условие `LIMIT` задает количество записей, которые будут затронуты в результате обработки запроса. В этом случае будет изменена одна из записей (обычно первая попавшаяся). Но что делать, если нам требуется изменить не любую запись с заданным условием, а конкретную запись? Думаю, вы уже догадались — мы будем идентифицировать запись по другому полю, например, по `email`:

```
UPDATE friends
SET name='Igor', email='777@host.ru'
WHERE email='user@somehost.ru' LIMIT 1;
```

Понятно, что если запись со значением `'user@somehost.ru'` поля `email` не будет найдена, то и обновления не произойдет. Для большей точности можно дополнительно указать поле `id`:

```
UPDATE friends
SET name='Igor', email='777@host.ru'
WHERE (id=1) and (email='user@somehost.ru')
LIMIT 1;
```

Вот теперь точно будет обновлена нужная запись.

18.8. Выборка записей

Выборка записей из таблицы выполняется с помощью оператора `SELECT`. Синтаксис этого оператора довольно сложный, поэтому рассмотрим его сокращенную версию:

```
SELECT список_полей
FROM таблица
[WHERE условие];
```

Например:

```
SELECT *
FROM friends;
```


В приведенном примере оператор `SELECT` выведет все записи из таблицы `friends`. Вместо списка полей мы указали `*`, поэтому будут выведены все поля. Понятно, что в нашей таблице есть поле `comment`, которое может содержать большой текст и его вывод, мягко говоря, нежелателен, поэтому целесообразнее следующий оператор:

```
SELECT id, name, email
FROM friends;
```

Данный оператор также выведет все записи (пока у нас только одна запись) из таблицы `friends`, но при этом будут выведены все поля, кроме `comment`. Чтобы увидеть поле `comment`, которое соответствует определенной записи, нам нужно задать условие оператора `SELECT`. Предположим, что вам требуется вывести поле `comment` для первой записи (обычно ее `id` будет равен единице):

```
SELECT comment
FROM friends
WHERE id=1;
```

ПРИМЕЧАНИЕ

Полный синтаксис оператора `SELECT` выглядит так:

```
SELECT [DISTINCT|ALL] {*| [поле1 AS псевдоним] [, ..., полеN AS
псевдоним] }
FROM Имя_таблицы1 [, ..., Имя_таблицыN]
[WHERE условие]
[GROUP BY список полей] [HAVING условие]
[ORDER BY список полей]
```

О том, как использовать все его возможности, вы узнаете чуть позже в этой главе.

18.9. Удаление записей

Условия удаления записи устанавливаются аналогично условиям обновления записи.

Удаление записей осуществляется оператором `DELETE`:

```
DELETE
FROM таблица
WHERE условие;
```

Вот небольшой пример:

```
DELETE
FROM friends
WHERE id=1;
```

Этот оператор удаляет первую запись в таблице. Удалить все записи из таблицы можно, указав оператор:

```
DELETE
FROM friends;
```


18.10. Встроенные функции

В SQL есть встроенные функции, например COUNT, MIN, MAX, AVG, SUM (существуют и другие, но эти — самые полезные). Функция COUNT вычисляет число элементов, MIN и MAX — минимальное и максимальное значения, AVG — среднее значение, а SUM — сумму элементов.

Использовать эти функции очень просто. Предположим, что у нас имеется таблица, созданная оператором CREATE (естественно, будем считать, что таблицу сразу после создания заполнили реальными данными):

```
CREATE TABLE books (  
  bid            int            auto_increment,  
  title          varchar(50)    not null,  
  autor          varchar(50)    not null,  
  price          double,  
  primary key (bid)  
);
```

Следующий оператор выведет число записей в таблице:

```
SELECT COUNT(*)  
FROM books;
```

Вообще, в этом операторе особого смысла нет, поскольку существует функция `mysql_num_rows()`, но иногда он тоже может пригодиться.

Функция MAX удобна для вычисления максимального значения указанного поля:

```
SELECT MAX(price)  
FROM books;
```

У MySQL очень много разных функций. Чтобы описать их все, нужно написать еще одну книгу, посвященную самому синтаксису SQL. Отмечу еще две полезные функции:

- ❑ `RAND()` — используется для выборки случайных записей;
- ❑ `LAST_INSERT_ID()` — возвращает ID последней добавленной записи.

Небольшой пример. Допустим, нужно выбрать 10 случайных товаров из категории 1:

```
SELECT * FROM items WHERE catid = 1  
ORDER BY RAND()  
LIMIT 10;
```

Первая часть запроса выбирает товары из категории 1, вторая обеспечивает сортировку в случайном порядке, а третья — ограничивает количество записей.

ПРИМЕЧАНИЕ

Подробное описание всех функций MySQL на русском языке вы найдете по адресу <http://phpclub.ru/mysql/doc/index.html>.

У MySQL также очень богатый набор функций для работы с датой и временем. Практически все, что связано с обработкой дат, можно выполнить средствами MySQL, не прибегая к использованию PHP. Прямая ссылка на описания функций даты и времени: <http://phpclub.ru/mysql/doc/date-and-time-functions.html>.

18.11. Группировка записей. Сложные запросы

Ранее мы выполняли запросы только к одной таблице. В самом простом случае этого достаточно. Но рано или поздно вам потребуется создать сложный проект, в котором придется выбирать записи из нескольких таблиц сразу. Тогда вам будет необходимо или выбирать все записи из нужных таблиц и писать процедуру обработки данных на PHP, или же просто учить SQL. Ведь намного проще передать запрос, а потом вывести его результат, чем вручную обрабатывать данные, — так что пусть за нас это сделает СУБД.

Предположим, что мы создаем небольшой интернет-магазин. Поэтому у нас будет три таблицы: клиенты, товары и заказы (листинг 18.2).

Листинг 18.2. Таблицы базы данных интернет-магазина

```
CREATE TABLE CUSTOMERS
(
C_NO      int          auto_increment,
LOGIN     varchar(32)  NOT NULL,
PASS      varchar(32)  NOT NULL,
EMAIL     varchar(32)  NOT NULL,
FIO       varchar(40)  NOT NULL,
ADDRESS   varchar(50)  NOT NULL,
CITY      varchar(25)  NOT NULL,
PHONE     varchar(11)  NOT NULL,
PRIMARY KEY (C_NO)
);

CREATE TABLE GOODS
(
G_NO      int          auto_increment,
DESCR     varchar(40)   NOT NULL,
PRICE     double(9,2)   NOT NULL,
QTY       int          NOT NULL,
PRIMARY KEY (G_NO)
);

CREATE TABLE ORDERS
(
O_NO      int          auto_increment,
DT        date,
```



```
C_NO          int          NOT NULL,  
G_NO          int          NOT NULL,  
QTY           int          NOT NULL,  
primary key (O_NO)  
);
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *18-2.sql*), представленный в каталоге *Glava_18* сопровождающего книгу электронного архива (см. приложение 4).

Думаю, назначение полей этих таблиц понятно и без моих комментариев. Сделаю только несколько небольших замечаний. Во-первых, мы продавать будем не сахар и не картошку, поэтому количество товара (поле *QTY*) везде целочисленное: две книги, один мобильный телефон и т. д. Во-вторых, индексы определены сразу, следовательно, *C_NO*, *G_NO* и *O_NO* — уникальные номера клиента, товара и заказа соответственно. В-третьих, поле *DESCR* содержит название товара. Конечно, на самом деле оно должно содержать еще и описание, а кроме того, должно быть поле *TITLE* с названием товара и поля *ART* и *PHOTO*, содержащие соответственно артикул товара и имя файла фотографии, но ведь это экспериментальная база и наша цель — не создание реального интернет-магазина, а изучение SQL, поэтому я упростил структуру таблиц до допустимого минимума.

После создания таблиц заполните их данными по своему усмотрению. Предположим, что в нашем магазине зарегистрировалось минимум два клиента, каждый из которых сделал несколько заказов. Нужно вычислить общую сумму всех заказов для каждого клиента. Для этого нам потребуется информация из всех трех таблиц:

- из *CUSTOMERS*, чтобы вывести информацию о клиенте (для простоты будем выводить только фамилию);
- из *GOODS*, ведь нам нужно знать цену каждого товара;
- из *ORDERS*, чтобы узнать, сколько заказали того или иного товара.

Введите следующий запрос:

```
SELECT CUSTOMERS.FIO, SUM( GOODS.PRICE * ORDERS.QTY ) AS A  
FROM CUSTOMERS, GOODS, ORDERS  
WHERE CUSTOMERS.C_NO = ORDERS.C_NO  
GROUP BY ORDERS.C_NO;
```

Сразу скажу, что этот запрос неправильный, однако на его примере рассмотрим несколько важных моментов. Мы выбираем только фамилию клиента, затем функция *SUM* подсчитывает сумму заказа, умножая цену товара (*GOODS.PRICE*) на количество заказанного товара (*ORDERS.QTY*). Результат работы этой функции будет представлен как поле *A* (вы можете назвать это поле по-своему).

Обратите внимание на то, что выборка производится из всех трех таблиц (*CUSTOMERS*, *GOODS* и *ORDERS*). Мы выбираем тех клиентов, которые есть в таблице заказов (*CUSTOMERS.C_NO = ORDERS.C_NO*), а группировку записей осуществляем по

полю `ORDERS.C_NO`. Вы уже заметили, как легко указать СУБД, к какой таблице обращаться: для этого просто нужно написать имя таблицы, а через точку — имя поля! А теперь — почему этот запрос неправильный. Мы выполнили группировку заказов, и если в таблице `ORDERS` есть несколько заказов от одного и того же клиента, вы заметите, что группировка выполняется верно, но сумма заказа вычисляется неправильно. Это происходит потому, что мы забыли указать, что следует подсчитывать только тот товар, который есть в таблице заказов, и для этого потребуется добавить еще одно условие:

```
ORDERS.G_NO = GOODS.G_NO
```

Полная версия SQL-оператора, выполняющего нужное нам действие, выглядит так:

```
SELECT CUSTOMERS.FIO, SUM( GOODS.PRICE * ORDERS.QTY ) AS A
FROM CUSTOMERS, GOODS, ORDERS
WHERE (CUSTOMERS.C_NO = ORDERS.C_NO) AND (ORDERS.G_NO = GOODS.G_NO)
GROUP BY ORDERS.C_NO;
```

Результат выполнения этого оператора представлен на рис. 18.1.

The screenshot shows the phpMyAdmin interface for a database named 'test'. The left sidebar lists various tables, including 'CUSTOMERS' and 'GOODS'. The main panel displays the following SQL query:

```
SELECT CUSTOMERS.FIO, SUM( GOODS.PRICE * ORDERS.QTY ) AS A
FROM CUSTOMERS, GOODS, ORDERS
WHERE (
    CUSTOMERS.C_NO = ORDERS.C_NO
)
AND (
    ORDERS.G_NO = GOODS.G_NO
)
GROUP BY ORDERS.C_NO
LIMIT 0, 30
```

Below the query, the results are displayed in a table:

FIO	A
Denis	950.00
Alex	1300.00

Рис. 18.1. Группировка записей:
вывод общей суммы всех заказов с группировкой по клиенту

Группировку записей в таблице `ORDERS` выполняет оператор `GROUP BY`. Его действие можно ограничить с помощью оператора `HAVING`. По сути, это аналог `WHERE`, но для `GROUP BY`. Синтаксис `HAVING` аналогичен `WHERE`:

`HAVING <условие>`

Предположим, что нам нужно вывести всех клиентов, общая сумма всех заказов которых превышает 500 (рублей, долларов — не важно). Модифицированный запрос будет выглядеть так:

```
SELECT CUSTOMERS.FIO, SUM(GOODS.PRICE * ORDERS.QTY) AS A
FROM CUSTOMERS, GOODS, ORDERS
WHERE (CUSTOMERS.C_NO = ORDERS.C_NO) AND (ORDERS.G_NO = GOODS.G_NO)
GROUP BY ORDERS.C_NO
HAVING A > 500;
```

Теперь, когда вы знаете основные SQL-операторы, можно перейти к рассмотрению функций PHP для работы с MySQL.

18.12. Копирование записей из одной таблицы в другую

Помню, при коллективной работе над одним проектом возникла не очень приятная ситуация. В базе данных было две таблицы: `book_alist` и `book_list`. В первой таблице предполагалось хранить список статей, а во второй — список книг.

Один нехороший человек поместил статьи в таблицу `book_list`. Целых 3800 записей. Сами понимаете, вручную копировать их никто не собирался. Писать PHP-сценарий тоже не хотелось, поэтому возникшую проблему я решил средствами SQL, тем более что это совсем не сложно. В нашем примере в поле `CID` хранится номер категории, в которую были добавлены все статьи. Теперь мы их перемещаем в таблицу `book_alist`:

```
INSERT INTO book_alist
SELECT * FROM book_list WHERE CID = 14
```

Запрос, как видите, очень прост. Мы берем все записи, где `CID = 14`, и вставляем их в таблицу `book_alist`.

Если же структура таблиц разная, то нужно указать поля, которые следует задействовать при выборке и вставке (в реальной задаче обратите внимание на порядок, в котором вы указываете поля: поля из `SELECT` должны соответствовать полям, указанным в `INSERT`):

```
INSERT INTO table1(CID, TITLE, PAGES)
SELECT BID, TNAME, PG FROM table2 WHERE CID = 14
```


18.13. Кэширование запросов

В предыдущих изданиях этой книги в данной главе были описаны только основы SQL. Теперь мы попробуем окунуться в более сложные материи. Ведь книга должна быть интересной не только для новичков, но и для уже состоявшихся PHP-программистов.

Кэширование запросов позволяет существенно повысить производительность PHP-приложений. Ведь рано или поздно все мы сталкиваемся с проблемой оптимизации. Вроде бы написали сценарий, он работает, хорошо работает, но до определенного момента — когда нагрузка на сервер становится такой, что сценарий начинает тормозить.

Здесь мы рассмотрим встроенный в MySQL механизм кэширования запросов. По умолчанию данный режим выключен. Запустите программу `mysql`, чтобы просмотреть параметры сервера прямо сейчас:

```
mysql> show variables like 'query_cache%';
```

Variable_name	Value
query_cache_limit	1048576
query_cache_min_res_unit	4096
query_cache_size	0
query_cache_type	ON
query_cache_wlock_invalidate	OFF

За кэш отвечает параметр `query_cache_size`. Как можно видеть, сейчас он равен 0, т. е. кэш выключен. Для включения кэша запросов выполните запрос:

```
set @@global.query_cache_size= 67108864;
```

В данном случае мы устанавливаем размер кэша равным 64 Мбайт ($64 \times 1024 \times 1024$), чего вполне достаточно для большинства случаев.

Для сохранения изменений нужно отредактировать файл `my.cnf` и добавить в раздел `[mysqld]` строку:

```
query_cache_size=64M
```

Кроме параметра `query_cache_size` можете включить параметр `query_cache_limit`, задающий максимальный объем результата, который может быть помещен в кэш. Результаты, превышающие этот лимит, кэшироваться не будут.

После включения кэш начинает работать автоматически. Больше ничего делать не нужно, запросы менять тоже не нужно.

Алгоритм работы кэша:

- при каждом запросе `SELECT` вычисляется хэш-сумма строки запроса, далее она ищется в кэше, если она там есть — сразу возвращается результат. Если нет, вы-

числяется результат и заносится в кэш. При следующем таком же запросе результат будет взят из кэша, что сократит время выполнения запроса;

- при каждом запросе `INSERT`, `UPDATE`, `REPLACE`, `DELETE`, `TRUNCATE` из кэша удаляются все записи, касающиеся таблицы, которая подверглась обновлению.

При составлении запросов помните, что кэш чувствителен к регистру символов, поэтому запросы `"SELECT * FROM tab"` и `"select * from tab"` — два разных запроса. Чтобы вы потом не удивлялись, почему кэш не работает.

Не кэшируются следующие запросы:

- в которых используются значения локальных переменных;
- использующие недетерминированные функции, например: `RAND()`, `NOW`, `SLEEP()`, `CURTIME()`, `LAST_INSERT_ID()` и т. д.;
- в которых вызываются определенные пользователем процедуры;
- генерирующие предупреждения;
- некоторые другие.

Чтобы отключить кэш для определенного запроса, добавьте в него директиву `SQL_NO_CACHE`, например:

```
SELECT SQL_NO_CACHE * FROM news;
```




ГЛАВА 19

Функции для работы с MySQL

19.1. Подключение к серверу MySQL

Прежде чем начать работу с базой данных, нужно выбрать ее и подключиться к серверу баз данных. Это можно сделать с помощью функций `mysql_connect()` и `mysql_select_db()`:

```
resource mysql_connect ([string server [, string username [, string password  
[, bool new_link [, int client_flags]]]])  
bool mysql_select_db (string database_name [, resource link_identifier])
```

Со второй функцией все понятно — ей следует просто передать имя базы данных и идентификатор соединения (о нем позже, он все равно необязательный). А вот о первой функции поговорим отдельно. Ее первый параметр — это имя сервера базы данных, к которому вы хотите подключиться. Обычно указывают значение `'localhost'`, но не всегда (имя сервера вы сможете уточнить у своего хостинг-провайдера). Два следующих параметра — имя пользователя и пароль — ну с этим, я думаю, все понятно. Параметр `new_link` заставляет функцию `mysql_connect()` открыть еще одно соединение, даже если аналогичное уже открыто. Последний параметр задает флаги клиента и может содержать комбинацию следующих значений:

- ☐ `MYSQL_CLIENT_COMPRESS` — использовать сжатие;
- ☐ `MYSQL_CLIENT_IGNORE_SPACE` — игнорировать лишние пробелы (пробелы бывают например, после имен функций);
- ☐ `MYSQL_CLIENT_INTERACTIVE` — установка времени ожидания до закрытия соединения в секундах: `interactive_timeout` вместо `wait_timeout` (поскольку первое значение обычно больше второго, таким способом можно увеличить тайм-аут).

Вообще, все параметры здесь не являются обязательными. Дело в том, что в файле `php.ini` (если у вас есть доступ к редактированию этого файла) можно указать имя сервера баз данных, имя пользователя и пароль по умолчанию, поэтому для подключения к серверу MySQL допустим вызов функции `mysql_connect()` без параметров:

```
mysql_connect ();
```


Кроме `mysql_connect()` можно использовать функцию `mysql_pconnect()`. Ее параметры такие же, как и у `mysql_connect()`, но разница между этими функциями огромная. Функция `mysql_connect()` создает соединение с сервером баз данных, которое будет автоматически закрыто сразу после того, как сценарий завершит работу (даже не придется вызывать функцию `mysql_close()`). Функция `mysql_pconnect()` открывает так называемое постоянное соединение, которое не будет закрыто после завершения работы сценария. Достигается это путем привязки соединения к PID веб-сервера, а не к сценарию. Поэтому постоянное соединение может быть закрыто в трех случаях:

1. Закрытие соединения с помощью функции `mysql_close()`.
2. Завершение работы веб-сервера.
3. Закрытие соединения удаленным сервером MySQL.

Функция `mysql_pconnect()` работает так: проверяет, есть ли идентификатор соединения. Если он есть, то функция его возвращает (другими словами, соединение уже существует и нет необходимости заново его создавать). Если идентификатора соединения нет, тогда оно будет создано.

Преимущество постоянных соединений заключается в том, что сценарии работают быстрее, поскольку не приходится тратить время на создание соединения. Да, за один вызов сценария вы этого не почувствуете, а вот когда у вас сложный сценарий (например, форум), который создает постоянные соединения, тогда результат будет виден без всяких замеров производительности (ведь соединение потребуется создать только один раз).

Но при использовании функции `mysql_pconnect()` очень легко превысить ограничение на максимальное количество подключений. Именно поэтому многие хостинг-провайдеры не разрешают работу с этой функцией — ведь вы будете не единственным клиентом, и если все клиенты сразу создадут постоянные соединения к серверу баз данных, то он, "бедняга", может и не выдержать.

Для закрытия соединения служит функция `mysql_close()`. Функции нужно передать ID соединения, который мы получили при вызове функции `mysql_connect()`. Но поскольку мы вызывали функцию `mysql_connect()`, а не `mysql_pconnect()`, то от вызова `mysql_close()` можно отказаться — соединение будет завершено автоматически.

Вот пример подключения к серверу баз данных MySQL:

```
// в этом файле определены переменные $HOST, $USER, $PASS и $DB
include db_options.php;

if (!mysql_connect($HOST, $USER, $PASS)) die(mysql_error());
mysql_select_db($DB);

...
```

Если подключиться к серверу невозможно, сценарий выводит сообщение об ошибке (функция `mysql_error()`) и завершает соединение. Кроме `mysql_error()` существ-

вует функция `mysql_errno()`, возвращающая номер ошибки, но она менее информативна.

После подключения к серверу можно вызвать функцию `mysql_select_db()` для выбора базы данных.

Если вам совесть не позволяет завершить работу сценария, не закрыв соединения, тогда следует вызвать функцию `mysql_close()`:

```
bool mysql_close ([resource link_identifier])
```

Ей нужно передать только идентификатор соединения, который можно не указывать при создании лишь одного MySQL-соединения.

19.2. Несколько MySQL-соединений

Предположим, что вам необходимо создать несколько подключений, например к разным базам данных. Сделать это очень легко и просто, важно только не "потерять" идентификаторы соединений. Пример подключения к разным базам иллюстрирует листинг 19.1.

Листинг 19.1. Подключение к разным базам данных

```
$conn1 = mysql_connect($HOST, $USER, $PASS);  
// поскольку мы подключаемся к одному и тому же серверу, используя  
// одно и то же имя пользователя, то нам нужно указать  
// newlink = TRUE,  
// иначе функция mysql_connect() возвратит один и  
// тот же идентификатор, т. е. conn1 = conn2  
$conn2= mysql_connect($HOST, $USER, $PASS, TRUE);  
  
// выбираем разные базы данных  
mysql_select_db($DB, $conn1);  
mysql_select_db($DB2, $conn2);  
  
// выполняем запросы  
...  
  
// закрываем соединения  
mysql_close($conn1);  
mysql_close($conn2);
```

Еще раз отмечу — при работе с несколькими соединениями важно не потерять их идентификаторы, иначе все данные перемешаются, и вы не поймете, откуда что взялось.

19.3. Передача запросов серверу

Теперь рассмотрим функцию `mysql_query()`, которая передает запрос серверу MySQL:

```
resource mysql_query (string query [, resource link_identifier])
```

Строка `query` должна содержать SQL-запрос. Вот пример:

```
$r = mysql_query('select * from users');
```

Переменная `$r` будет содержать результат выполнения запроса. Для его анализа используются функции, представленные в табл. 19.1.

Таблица 19.1. Функции для работы с результатом SQL-запроса

Функция	Описание
<code>int mysql_num_rows (resource result)</code>	Возвращает количество записей результата запроса
<code>array mysql_fetch_array (resource result [, int result_type])</code>	Возвращает следующую запись результата запроса в виде массива. Очень удобная функция, мы будем часто ее использовать. Тип массива указывается с помощью второго параметра: MYSQL_ASSOC — ассоциативный массив (по умолчанию); MYSQL_NUM — численный массив; MYSQL_BOTH — оба типа массива
<code>array mysql_fetch_assoc (resource result)</code>	Возвращает следующую запись результата запроса в виде ассоциативного массива. Аналогична функции <code>mysql_fetch_array()</code> со значением <code>MYSQL_ASSOC</code> в качестве второго параметра. Но, учитывая, что по умолчанию функция <code>mysql_fetch_array()</code> возвращает ассоциативный массив, можно считать функции <code>mysql_fetch_array()</code> и <code>mysql_fetch_assoc</code> аналогами
<code>object mysql_fetch_field (resource result [, int field_offset])</code>	Возвращает полную информацию о конкретном поле записи. Далее мы подробнее рассмотрим эту функцию
<code>array mysql_fetch_row (resource result)</code>	Обрабатывает следующую запись результата запроса и возвращает неассоциативный массив. Функция не очень удобна, поэтому мы ее использовать не будем
<code>object mysql_fetch_object (resource result)</code>	Возвращает следующую запись результата запроса в виде объекта. По удобству эту функцию можно сравнить с <code>mysql_fetch_array()</code> . Какую из них выбрать, вы решите сами, когда мы рассмотрим примеры

Предположим, что у нас есть таблица `users`, содержащая всего три поля:

- ▮ Username;
- ▮ Email;
- ▮ Password.

Рассмотрим следующий пример вывода всех записей этой таблицы с помощью функции `mysql_fetch_array()` (листинг 19.2).

Листинг 19.2. Пример использования `mysql_fetch_array()`

```
// считаем, что к серверу мы уже подключились и базу данных выбрали

// это наш SQL-запрос
$q = 'select * from users';

// получаем результат
$r = mysql_query($q);

// по запросу мы получили все записи, поэтому значение, возвращенное
// mysql_num_rows, равно количеству записей в таблице
echo "Количество записей в таблице: " . mysql_num_rows();

// выводим результат как HTML-таблицу
echo "<TABLE>";

// вызываем mysql_fetch_array в цикле while, второй параметр можно
// не указывать
while ($row = mysql_fetch_array($r, MYSQL_ASSOC)) {

    // забудем об оптимизации – так красивее и проще код
    echo "<TR>";
    echo "<TD>$row[username]</TD>";
    echo "<TD>$row[email]</TD>";
    echo "<TD>$row[password]</TD>";
    echo "</TR>";

}

// таблица users выведена
echo "</TABLE>";
```

Теперь рассмотрим пример вывода этой же таблицы, но с помощью функции `mysql_fetch_object()` (листинг 19.3).

Листинг 19.3. Пример использования `mysql_fetch_object()`

```
// отличия от листинга 19.2 начинаются отсюда

// вызываем mysql_fetch_object в цикле while
while ($row = mysql_fetch_object($r)) {

    echo "<TR>";
    echo "<TD>$row->username</TD>";
```



```
echo "<TD>$row->email</TD>";
echo "<TD>$row->password</TD>";
echo "</TR>";

}

// таблица users выведена
echo "</TABLE>";
```

По быстродействию функции `mysql_fetch_object()` и `mysql_fetch_array()` практически одинаковы, поэтому особой разницы вы не заметите.

Еще я обещал рассмотреть функцию `mysql_fetch_field()`, которая возвращает информацию о конкретном поле (пример приведен в листинге 19.4). Информация возвращается в виде объекта со следующими свойствами:

- ❑ `name` — название поля;
- ❑ `table` — название таблицы, которой принадлежит поле;
- ❑ `max_length` — максимальная длина поля;
- ❑ `not_null` — равно 1, если поле не может быть равно NULL;
- ❑ `primary_key` — равно 1, если поле является первичным индексом;
- ❑ `unique_key` — равно 1, если поле является уникальным индексом;
- ❑ `multiple_key` — равно 1, если поле не уникальный индекс;
- ❑ `numeric` — равно 1, если поле численное (не строка);
- ❑ `blob` — равно 1, для поля типа BLOB;
- ❑ `type` — тип поля;
- ❑ `unsigned` — равно 1, если колонка строго положительная (unsigned);
- ❑ `zerofill` — равно 1, если колонка заполняется нулями (zero-filled).

Листинг 19.4. Пример использования `mysql_fetch_field()`

```
// считаем, что подключены к серверу, и база данных выбрана
$r = mysql_query("select * from users");

// получаем данные о поле
// mysql_num_fields($r) — возвращает количество полей
$i = 0;
while ($i < mysql_num_fields($r)) {
    echo "Информация для поля $i:<br />";
    $obj = mysql_fetch_field($r, $i);
    if (!$obj) {
        echo "Информация недоступна! <br />";
    }
}
```



```

else {
    echo "<pre>
blob:          $obj->blob
max_length:    $obj->max_length
multiple_key:  $obj->multiple_key
name:          $obj->name
not_null:      $obj->not_null
numeric:       $obj->numeric
primary_key:   $obj->primary_key
table:         $obj->table
type:          $obj->type
unique_key:    $obj->unique_key
unsigned:      $obj->unsigned
zerofill:      $obj->zerofill
</pre>";
}
    $i++;
}

```

19.4. Работа с базой данных. Создание базы данных

Мы уже знакомы с функцией `mysql_select_db()`, позволяющей выбрать базу данных. Кроме нее вам будут полезны функции, описанные в этом разделе. Начнем с `mysql_create_db()`:

```

bool mysql_create_db (string database_name
                      [, resource link_identifier])

```

Все, что нужно указать, — это имя создаваемой базы данных в качестве первого параметра. Но вам следует помнить, что у пользователя, от имени которого вы подключаетесь к серверу MySQL, должно хватить полномочий для выполнения этой операции.

Переключением к другой базе данных можно выполнить и не прибегая к функции `mysql_select_db()` — достаточно вызвать функцию `mysql_db_query()`, которой передать два параметра: имя базы данных и строку запроса:

```

resource mysql_db_query (string database,
                        string query [, resource link_identifier])

```

Для удаления базы данных предназначена функция `mysql_drop_db()`:

```

bool mysql_drop_db (string database_name
                   [, resource link_identifier])

```

Так же, как и в случае с `mysql_create_db()`, у вас должно хватить полномочий для выполнения операции.

19.5. Функция `mysql_real_escape_string($content)`

Предположим, что у нас есть следующий оператор:

```
$q = "insert into tab values(0, \"$content\");
```

Этот оператор формирует запрос, добавляющий в таблицу `tab` значение переменной `$content`. Вроде бы все здесь нормально, и запрос должен выполняться без особых проблем. Когда в переменной `$content` нет кавычек, то так оно и будет, например:

```
$content = "Фирма Рога и копыта";  
$q = "insert into tab values(0, \"$content\");  
mysql_query($q) or die(mysql_error());
```

Но если в переменной `$content` есть кавычки, наш запрос не будет выполнен:

```
$content = 'Фирма "Рога и копыта" — лучший производитель стульев';
```

Чтобы избавиться от этой проблемы, нужно вызвать функцию `mysql_real_escape_string()` для обработки значения переменной `$content` перед помещением информации в базу данных:

```
$content = 'Фирма "Рога и копыта" — лучший производитель стульев';  
$content = mysql_real_escape_string($content);  
$q = "insert into tab values(0, \"$content\");  
mysql_query($q) or die(mysql_error());
```

19.6. Получение информации о результате запроса

Функция `mysql_field_name(int $result, int $offset)` возвращает имя поля, находящегося в результате `$result` с номером `$offset`. Нумерация начинается с 0 и задается относительно результата, а не всей таблицы!

Функция `mysql_field_type(int $result, int $offset)` возвращает тип поля с номером `$offset` в результате `$result`. Смещение, аналогично, задается относительно результата, а не таблицы.

Функция `mysql_field_len(int $result, int $offset)` возвращает длину (размер) поля.

Функция `string mysql_field_table(resource result, int field_offset)` возвращает имя таблицы, в которой содержится поле с номером `field_offset` результата `result`.

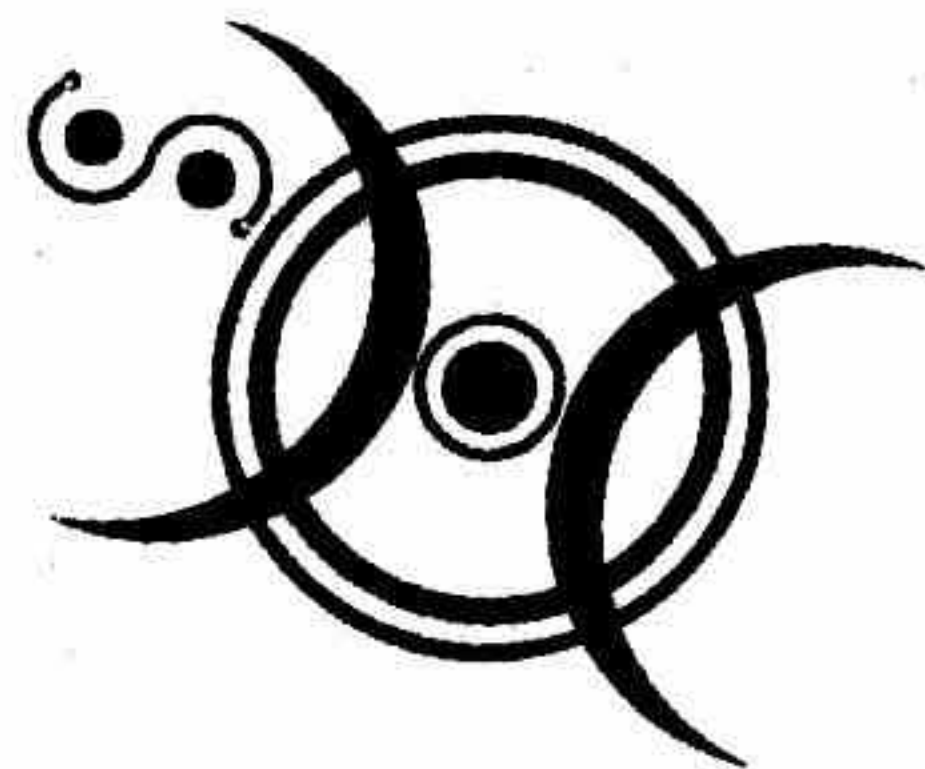
И, наконец, функция `mysql_field_flags(int $result, int $offset)` возвращает перечисленные через пробел флаги (модификаторы) поля (`Primary_Key`, `not_null`, `auto_increment` и т. д.), установленные для поля с номером `$offset`.

Рассмотрим практический пример использования этих функций (листинг 19.5).

Листинг 19.5. Получение информации в результате запроса

```
<?php
// измените имя пользователя и пароль!
mysql_connect("localhost", "root", "123456");
// измените имя БД
mysql_select_db("test");
$result = mysql_query("SELECT * FROM test");
// получаем количество полей
$fields = mysql_num_fields($result);
// получаем количество записей
$rows = mysql_num_rows($result);
// получаем имя таблицы (смещение 0)
$table = mysql_field_table($result, 0);
echo "<PRE>";
echo "Таблица '". $table. "' содержит '". $fields. "' полей и '". $rows. "'
записей\n";
echo "Информация о полях таблицы:\n";
for ($i=0; $i < $fields; $i++) {
    $type = mysql_field_type($result, $i);
    $name = mysql_field_name($result, $i);
    $len = mysql_field_len($result, $i);
    $flags = mysql_field_flags($result, $i);
    echo $type. " ". $name. " ". $len. " ". $flags. "\n";
}
echo "</PRE>";
mysql_free_result($result);
mysql_close();
?>
```


ГЛАВА 20



Альтернативная база данных SQLite

20.1. MySQL vs SQLite: что лучше?

PHP может работать с различными серверами баз данных: MySQL, mSQL, Sybase, SyBase-CT, SQLite, PostgreSQL. Существуют модули для работы даже с InterBase (если нужно, то и Oracle можно заставить работать с PHP).

Какую же СУБД выбрать? Начну с Oracle. Это очень серьезная СУБД, которая вряд ли потребуется для ваших PHP-проектов. Ее возможности настолько велики, что использовать ее для какого-либо PHP-проекта — это все равно, что стрелять из пушки по воробьям, особенно учитывая ограничение на время выполнения сценария. Область применения Oracle — сложные приложения для работы с базой данных, как правило, без веб-интерфейса. Однако если у вас имеется собственный сервер, то вы можете установить на него Oracle и настроить PHP для работы с этой СУБД. Интересующимся, как это сделать, рекомендую обратиться к статье Владимира Пржиялковского, преподавателя Oracle, опубликованной по адресу:

<http://www.dkws.org.ua/phpbb2/viewtopic.php?t=1827>.

СУБД Sybase не очень часто устанавливается на серверах хостинг-провайдеров, поэтому, хотя поддержка этой СУБД в PHP есть, используется она довольно редко. InterBase тоже встречается нечасто. Во-первых, поддержки InterBase по умолчанию в PHP нет, но ее легко добавить, установив соответствующие модули (библиотеки). Во-вторых, если вам и понадобится InterBase, то это будут специфические проекты, предположим, преобразование базы данных InterBase в другой формат, например в MySQL.

Автор СУБД mSQL описывает ее буквально так: "легкая база данных для быстрого доступа к данным, не требующая большого объема оперативной памяти". В свое время mSQL была разработана для одного проекта, который раньше основывался на СУБД PostgreSQL. Тогда разработчиков не устроила скорость работы PostgreSQL. СУБД mSQL, если верить ее создателям, быстрее PostgreSQL в 20 (!) раз. Но, к сожалению, mSQL сейчас не так популярна, как MySQL. Зайдите в

Google и введите запрос: СУБД mSQL — да, результаты вы получите, но в первой строчке вывода будет стоять вопрос: **Может, вы имели в виду СУБД MySQL?..**

PostgreSQL подходит для проектов средней сложности, с которыми по какой-либо причине не справится MySQL. Как уже было отмечено, PostgreSQL — довольно медленная СУБД.

Нам остаются только MySQL и SQLite. Почему так популярен сервер MySQL? Во-первых, он бесплатный и именно поэтому установлен на серверах большинства хостинг-провайдеров. Во-вторых, MySQL обеспечивает довольно приемлемую производительность. В-третьих, MySQL позволяет создавать полноценные сетевые базы данных. В-четвертых, функции для работы с MySQL есть в PHP.

Чем же хорош SQLite, если MySQL и так нас устраивает? На заре появления пятой версии PHP возникла проблема с лицензированием, и дело шло к тому, что пятая версия PHP могла остаться вообще без поддержки MySQL. Нужно было чем-то заменить проблемную СУБД. Такой заменой и стал SQLite. Почему именно SQLite? Он также полностью бесплатен и распространяется без ограничений — на тот момент это было очень важно. Да и производительность у SQLite выше, чем у MySQL.

Правда, как обычно, есть всегда одно "но". Производительность SQLite достигается только за счет того, что он является локальным сервером баз данных. Дело в том, что к MySQL-серверу, если это разрешил администратор, можно подключаться с других узлов сети (указав, с каких именно). Таким образом, физически наш сценарий может быть размещен на одном сервере (веб-сервере), а база данных — на другом. В случае с SQLite сервер баз данных обязательно должен быть установлен на веб-сервере. Сами понимаете, при таком ограничении эффективно управлять сетевыми ресурсами нельзя, поэтому SQLite подойдет только для небольших проектов (вроде блогов) и таких, где немного записей и несложные SQL-запросы.

Вот основные характеристики SQLite:

- ☐ частичное соответствие стандарту SQL-92 (поддерживаются не все функции);
- ☐ максимальный размер базы данных — 2 Тбайт;
- ☐ скорость работы — в два раза быстрее, чем MySQL.

Если вас устраивает SQLite, и он установлен на сервере хостинг-провайдера, почему бы его не использовать? Ведь ваши проекты станут работать быстрее. Но при этом вы также должны учитывать, что SQLite установлен не везде, поэтому, когда со временем появится необходимость перенести ваши сценарии на другой сервер, вполне вероятно, что придется их переписывать с учетом функций MySQL, да и базу данных нужно будет конвертировать в формат MySQL. Поэтому прежде чем сравнивать производительность, подумайте, не понадобится ли вам универсальный проект? Если да, то, не задумываясь, выбирайте MySQL. В этой главе мы рассмотрим функции для работы с обоими серверами.

20.2. Открытие базы данных

Поскольку SQLite — локальный сервер баз данных, то подключение к нему не требуется. Нужно просто указать файл базы данных, которую вы хотите открыть:

```
sqlite_open ( string filename  
             [, int mode [, string &error_message]])
```

Первый параметр — имя файла базы данных. Если указанный файл не существует, функция попытается его создать. Другими словами, эту функцию можно использовать не только для открытия базы данных, но и для ее создания. Второй параметр — это режим доступа к базе данных (режим здесь — это права доступа к файлу в UNIX). По умолчанию установлен режим 0666. Третий параметр (ссылка) будет содержать сообщение об ошибке, если, конечно, ошибка возникнет.

Постоянное соединение можно открыть с помощью функции `sqlite_popen()`. Параметры у нее такие же, как и у `sqlite_open()`.

Чтобы закрыть соединение, вызовите функцию `sqlite_close()`:

```
void sqlite_close ( resource dbhandle )
```

Вот простой пример использования `sqlite_open()`:

```
<?php  
if ($db = sqlite_open('mydb', 0666, $sqliteerror)) {  
    // выполняем запросы к базе данных  
} else {  
    // произошла ошибка  
    die($sqliteerror);  
}  
?>
```

20.3. Передача запросов

Для выполнения запроса `query` к базе данных `dbhandle` существует следующая функция:

```
resource sqlite_query ( resource dbhandle, string query )  
resource sqlite_query ( string query, resource dbhandle )
```

Пусть вас это не удивляет — функцию можно вызывать, указав сначала запрос, а потом идентификатор базы данных, и наоборот.

Приведем небольшой пример вызова функции `sqlite_query()`:

```
sqlite_query($db,  
            'CREATE TABLE users (login varchar(15), pass varchar(25))' );  
sqlite_query($db,  
            "INSERT INTO users VALUES ('den', '12345678' )");
```


20.4. Работа с результатом запроса

Для работы с результатом запроса удобнее всего будет функция `sqlite_fetch_array()`, аналогичная `mysql_fetch_array()`:

```
array sqlite_fetch_array ( resource result
                        [, int result_type [, bool decode_binary]] )
```

Первый параметр — это результат, второй — тип возвращаемого массива. Как и в функции `mysql_fetch_array()`, возвращается ассоциативный массив (`SQLITE_ASSOC`). Если вы хотите получить численный массив, то в качестве второго параметра нужно указать `SQLITE_NUM`, а если оба — то `SQLITE_BOTH`.

Кроме `sqlite_fetch_array()` имеются функции `sqlite_column()` и `sqlite_fetch_single()`. Первая применяется для получения одного поля результата, а вторая — для получения первого поля из результата запроса. Вот прототипы этих функций:

```
sqlite_column ( resource result, mixed index_or_name [, bool decode_binary])
```

```
sqlite_fetch_single ( resource result [, int result_type [, bool
decode_binary]])
```

Последний параметр необходим для двоичного декодирования, обычно его указывают, если база данных создана каким-то посторонним приложением, а не сервером SQLite.

Для управления внутренним указателем результата предназначены функции

`sqlite_next()`, `sqlite_rewind()` и `sqlite_seek()`:

```
sqlite_next ( resource result)
sqlite_rewind ( resource result)
sqlite_seek ( resource result, int rownum)
```

Первая функция перемещает указатель результата на следующую запись (если функция возвратила `FALSE`, значит, больше записей нет), вторая — на первую запись, а третья — переходит к записи с указанным номером (если функция возвратила `FALSE`, то нет записи с указанным номером), т. е. позволяет перейти к произвольной записи результата. Это очень удобно, когда нужно обработать результат последовательно, а произвольно выбрать некоторые записи.

Рассмотрим пример вывода всех записей таблицы (листинг 20.1).

Листинг 20.1. Пример вывода всех записей таблицы

```
<?php
if ($dbhandle = sqlite_open('mydb', 0666, $sqliteerror))
{
    // SQL-запрос
    $sql = "SELECT * FROM users";
    // получаем результат
    $res = sqlite_query($dbhandle , $sql);
```



```
// если количество записей в результате больше 0, то выводим их
if (sqlite_num_rows($res) > 0) {

    while($f = sqlite_fetch_array($res)) {
        echo "$f[login] $f[pass]<br>";
    }
}
else
    echo "В таблице нет записей!";
}
// закрываем базу данных
sqlite_close($dbhandle);
}
?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *20-1.php*), представленный в каталоге *Glava_20* сопровождающего книгу электронного архива (см. приложение 4).

20.5. Список PHP-функций для работы с SQLite

Мы рассмотрели основные функции для работы с SQLite. Остальные функции, которые мы еще не рассмотрели, для удобства сведены в табл. 20.1. Но нужно отметить, что на практике они встречаются не так часто.

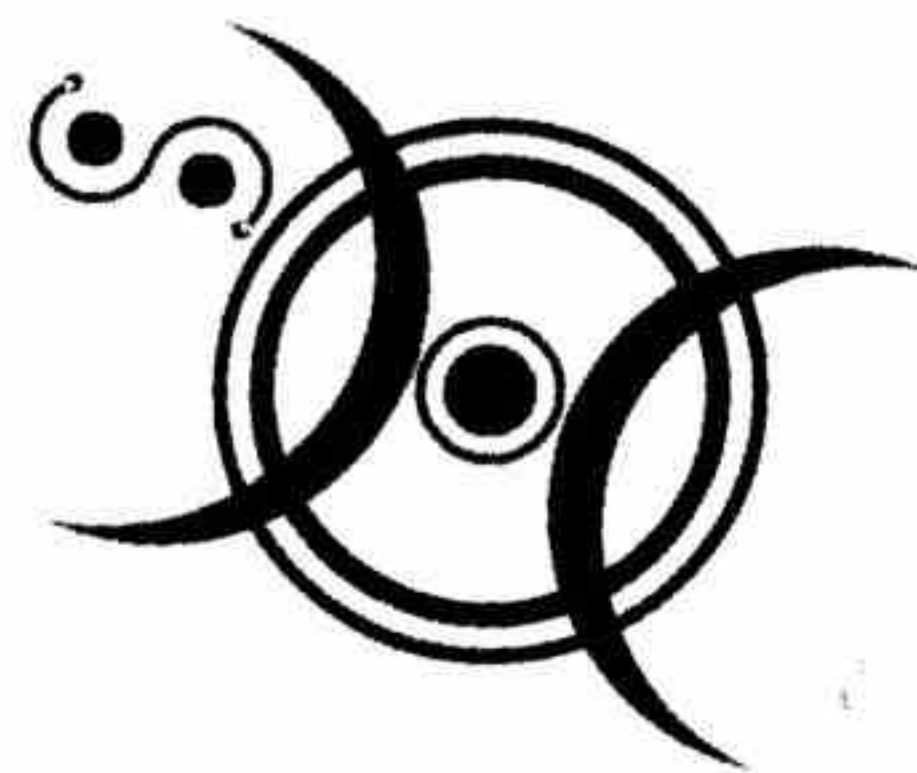
Таблица 20.1. PHP-функции для работы с SQLite

Функция	Описание
<code>array sqlite_array_query (resource db, string query [, int res_type [, bool decode_binary]])</code>	Выполняет запрос и сразу получает результат в виде массива. Аналогична вызову <code>sqlite_query()</code> , а затем вызову <code>sqlite_fetch_array()</code> . Данную функцию желательно использовать, если количество записей результата не превышает 45. В противном случае лучше вызвать функцию <code>sqlite_unbuffered_query()</code>
<code>void sqlite_busy_timeout (resource dbhandle, int milliseconds)</code>	Устанавливает максимальное время ожидания готовности базы данных (в миллисекундах). По умолчанию это время равно 60 с
<code>int sqlite_changes (resource dbhandle)</code>	Возвращает количество записей, измененных в результате последнего запроса
<code>array sqlite_current (resource result [, int result_type [, bool decode_binary]])</code>	Возвращает текущую запись
<code>string sqlite_error_string (int err_code)</code>	Возвращает строку с описанием ошибки. Функции нужно передать код ошибки

Таблица 20.1 (окончание)

Функция	Описание
<code>string sqlite_escape_string (string item)</code>	Экранирует спецсимволы в строке SQL-запроса. Обратите внимание: функцию <code>addslashes()</code> нельзя использовать для экранирования запросов к SQLite
<code>bool sqlite_exec (string query, resource dbhandle)</code>	Выполняет запрос, который не возвращает записей, например, INSERT-запрос. Функция возвращает TRUE, если запрос выполнен. Применение этой функции совсем необязательно, ведь есть <code>sqlite_query()</code>
<code>object sqlite_fetch_object (resource result [, string class_name [, array ctor_params [, bool decode_binary]])</code>	Возвращает текущую запись результата в виде объекта. Аналогична по своему действию функции <code>mysql_fetch_object()</code>
<code>array sqlite_fetch_all (resource result [, int result_type [, bool decode_binary]])</code>	Выбирает все записи и возвращает их в виде массива
<code>sqlite_fetch_string()</code>	Псевдоним для функции <code>sqlite_fetch_single()</code>
<code>string sqlite_field_name (resource result, int field_index)</code>	Возвращает имя поля с указанным номером поля. Нумерация начинается с нуля
<code>bool sqlite_has_more (resource result)</code>	Возвращает TRUE, если в результате еще есть доступные записи
<code>bool sqlite_has_prev (resource result)</code>	Возвращает TRUE, если доступна предыдущая запись результата
<code>int sqlite_key (resource result)</code>	Возвращает индекс текущей записи результата
<code>int sqlite_last_error (resource dbhandle)</code>	Возвращает код последней ошибки
<code>int sqlite_last_insert_rowid (resource dbhandle)</code>	Возвращает идентификатор последней вставленной записи (для записей с автоинкрементом)
<code>string sqlite_libversion (void)</code>	Возвращает версию библиотеки SQLite
<code>resource sqlite_unbuffered_query (resource dbhandle, string query)</code>	Выполняет запрос без буферизации результата. Идеально подходит для последовательной обработки записей (например, вывод записей в браузер), т. е. когда нет необходимости доступа к произвольной записи результата

ГЛАВА 21



Полезные приемы при работе с MySQL 6

21.1. Выбор кодировки

Довольно часто встречается ситуация, когда сайт создается в одной кодировке, а база данных использует другую, — например, в базе данных установлена кодировка koі8-г, а на сайте — cp1251. Получается, что содержимое (в том числе и навигация) сайта выводится в кодировке cp1251, а информация из базы данных — в koі8-г. В результате одновременно прочитав вывод из базы данных и страницу сайта не удастся, кодировки-то разные (рис. 21.1).



Рис. 21.1. Кодировки сайта и базы данных разные

Конечно, можно перекодировать данные с помощью функции `convert_cyr_string()`, например, так:

```
$txt = convert_cyr_string($txt, "k", "w");
```

Но тогда вам придется перекодировать любой текст, выводимый из базы данных, поэтому намного проще сразу установить требуемую кодировку для базы данных. Это можно сделать с помощью следующего запроса:

```
mysql_query('SET NAMES cp1251;');
```

который должен быть первым запросом после выбора базы данных:

```
if (!mysql_connect($HOST, $USER, $PASS))  
    die('Ошибка');
```

```
mysql_select_db($DB);
```

```
mysql_query('SET NAMES cp1251;');
```

21.2. Сортировка: вывод новинок. Вывод случайных записей

В главе 18 мы познакомились со встроенной функцией `RAND()`. Сейчас мы рассмотрим ее подробнее.

Представим, что у нас есть небольшой интернет-магазин. Как и в любом другом магазине, в нем имеется перечень товаров. Когда мы вводим новый товар, запись добавляется в конец таблицы. Это не очень удобно, потому что при просмотре общего списка самые новые товары отображаются последними.

Нам может помочь сортировка записей, которая выполняется оператором `ORDER BY`, работающим в паре с оператором `SELECT`:

```
SELECT *  
FROM books  
ORDER BY bid;
```

В данном случае мы выводим записи из таблицы `books` с сортировкой по полю `bid` (`book id`). По умолчанию выполняется сортировка по возрастанию (модификатор `ASC`). Если необходима сортировка по убыванию, то следует указать модификатор `DESC`:

```
SELECT *  
FROM books  
ORDER BY bid DESC;
```

Вот теперь наши записи станут отображаться в обратном порядке, т. е. сначала будут выводиться те записи, которые были добавлены последними.

Второй очень распространенный прием — вывод случайных записей. Для вывода записей в случайном порядке используется SQL-функция `RAND()`. Рассмотрим пример вывода десяти случайных записей из таблицы `books`:


```
SELECT *
FROM books
ORDER BY RAND()
LIMIT 10;
```

Функцией `RAND()` не нужно злоупотреблять, поскольку она изрядно нагружает сервер, особенно если у вас много обращений к сценарию, использующему SQL-функцию `RAND()`, а выборка производится из большой таблицы.

21.3. Постраничный вывод таблицы

Вывести информацию из базы данных в виде HTML-страницы достаточно просто. Смотреться это тоже будет красиво. Но вот если выводится много записей, скажем, больше 30, то читать такой вывод не очень удобно — попробуйте просмотреть пару сотен записей на одной странице. А ведь в таблице могут быть даже не сотни, а тысячи записей.

Выходом из данной ситуации может служить постраничный вывод из базы данных (рис. 21.2). Отметим, что эта функция присутствует практически во всех серьезных PHP-проектах, поэтому она обязательно должна быть и у вас.

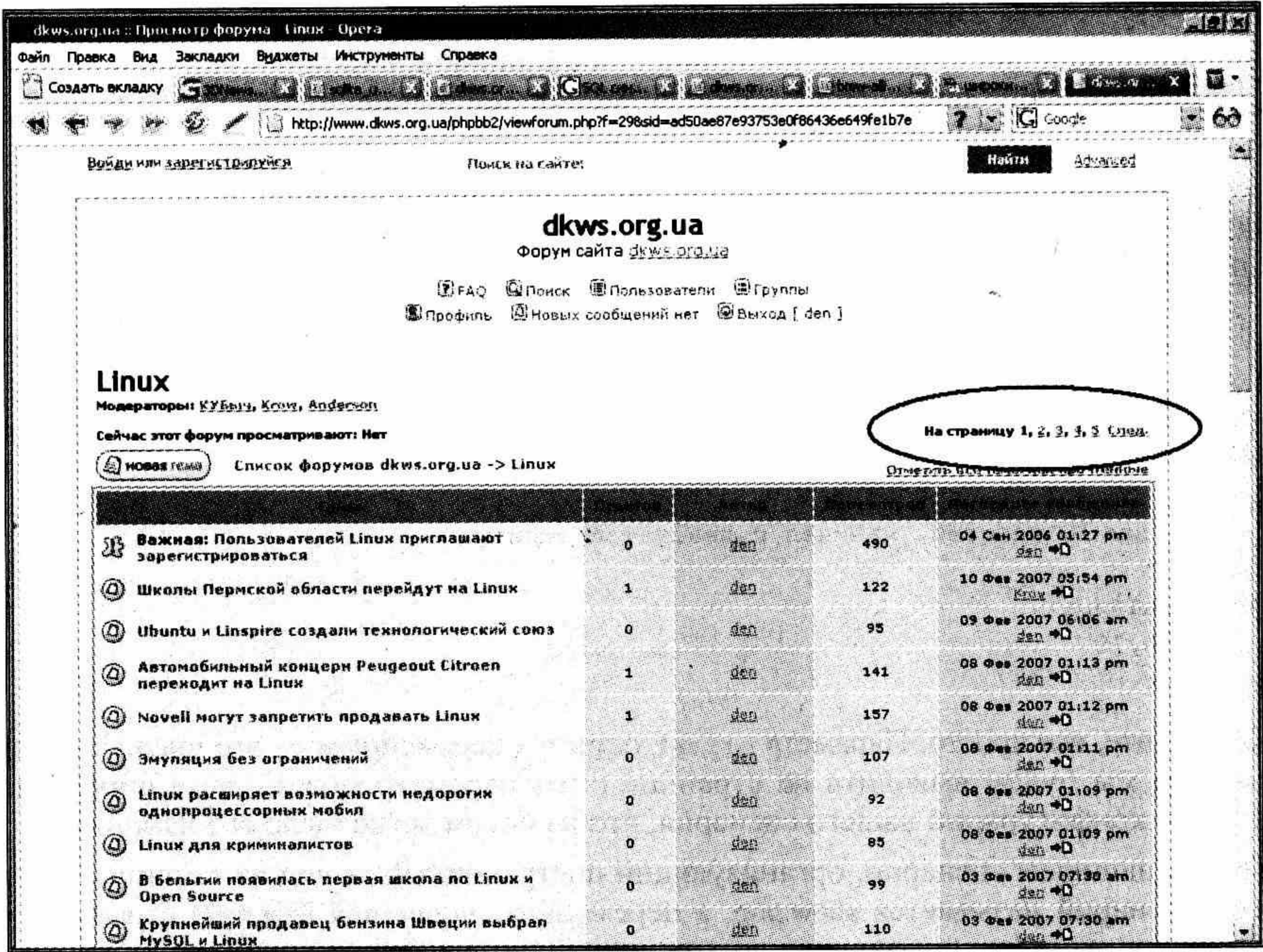


Рис. 21.2. Постраничный вывод форума PHPBB

Существуют два алгоритма организации постраничного вывода: нерациональный и рациональный. Первый заключается в том, что мы получаем все записи, которые нужно вывести, а затем анализируем некую переменную $\$p$, содержащую смещение от начала таблицы. После вывода части записей мы увеличиваем переменную $\$p$ на величину, равную числу выведенных записей, таким образом, при отображении следующей страницы на ней не окажется уже выведенных ранее записей. Нерациональность данного способа заключается в том, что мы получаем все записи таблицы, а это лишняя нагрузка на сервер баз данных, да и сам сценарий будет работать медленно, особенно когда записей в таблице много.

Второй способ заключается в совместном использовании операторов `LIMIT` и `SELECT`. Как мы уже знаем, оператор `LIMIT` позволяет ограничить выбор записей из базы данных (мы так уже делали при обновлении и удалении записей). Рассмотрим синтаксис `LIMIT`:

```
SELECT *  
FROM <таблица>  
[WHERE <условие>]  
LIMIT <start>, <offset>
```

Здесь `start` — это номер начальной записи, а `offset` — это смещение, данный параметр задает, сколько записей нужно вывести, начиная от начальной, заданной параметром `start`. Примеры иллюстрирует листинг 21.1.

Листинг 21.1. Примеры использования оператора `LIMIT`

```
// выводим первые 10 записей  
SELECT *  
FROM <таблица>  
LIMIT 0,10  
  
// выводим 10 записей, начиная с десятой записи  
SELECT *  
FROM <таблица>  
LIMIT 10,10  
  
// выводим 10 записей, начиная с двадцатой записи  
SELECT *  
FROM <таблица>  
LIMIT 20,10
```

Как видите, все просто. Параметр `offset` остается неизменным — это число записей, которое мы будем выводить на странице (этот параметр можно даже определить в файле конфигурации нашего сценария, чтобы потом легко было его изменить).

Теперь напишем сценарий, организующий постраничный вывод из таблицы. Пусть наш сценарий называется `show.php`, а переменная, задающая нужную страницу, — $\$page$. Тогда ссылка на следующую страницу будет выглядеть так:

```
<a href=show.php?page=Номер_страницы>Далее</a>
```


Сценарий начнет выводить по пять объявлений из таблицы, а также ссылки **Назад** и **Далее**. Первая ссылка будет выведена, если номер страницы больше нуля, а вторая — если еще есть записи для вывода. Структура таблицы `ob` очень проста: поле `dt` — дата публикации объявления, а поле `t` — текст самого объявления. Код сценария, а также подробные комментарии приведены в листинге 21.2.

Листинг 21.2. Сценарий постраничного вывода таблицы

```
<?
// подключаемся к серверу, выбираем базу данных и устанавливаем кодировку
mysql_connect("localhost","root","password");
mysql_select_db("database");
mysql_query('SET NAMES cp1251;');

// число объявлений на странице
$N = 5;

// использовать функцию mysql_num_rows() нельзя, потому что
// для того чтобы узнать общее число записей, нужно выполнить
// запрос select * from ob, т. е. получить все записи, а это
// нежелательно, поэтому вызываем функцию count

$r1=mysql_query("select count(*) as rec from ob");
$f = mysql_fetch_row($r1);
$rec = $f[0];           // всего записей в таблице

// если страница не указана, выводим первую
if (!isset($_GET['page'])) $page=0;
else $page = $_GET['page'];

// записи, которые нужно вывести
$records = $page * $N;

// запрос
$q="select * from ob limit ".$records.", $N";

echo "Всего записей $rec<br>";

$r=mysql_query($q);      // выполняем запрос
$n = mysql_num_rows($r); // число записей

// если страница не первая, выводим ссылку "Назад"
if ($page > 0) {
    $p = $page - 1;
    echo "<a href=show.php?page=$p>Назад</a>&nbsp;";
}

$page++;                // увеличиваем страницу
```



```
// выводим ссылку на следующие пять записей (на след. страницу),
// если она есть, то есть число записей, которые нужно вывести,
// и смещение не превышают общего числа записей
if ($records+$N < $rec)
echo "<a href=show.php?page=$page>Далее</a>";

// выводим объявления
for ($i=0; $i<$n; $i++)
{
    $f=mysql_fetch_array($r);

    // выводим дату (поле dt)
    echo "<p><table width=100%>
        <tr><td bgcolor=navy width=10%>
            <font name=tahoma size=2 color=white>$f[dt]</font></td>";

    // выводим текст объявления (поле t)
    echo "<td colspan=2 bgcolor=grey>
        <font name=tahoma size=2 color=white>$f[t]</td>
        </tr></table>";
}

?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *21-2.php*), представленный в каталоге *Glava_21* сопровождающего книгу электронного архива (см. приложение 4).

Работа сценария представлена на рис. 21.3. В главе 44 мы модифицируем код постраничного вывода записей так, чтобы он показывал не ссылки **Назад** и **Далее**, а номера страниц с записями из базы данных — так, как показано на рис. 21.2.

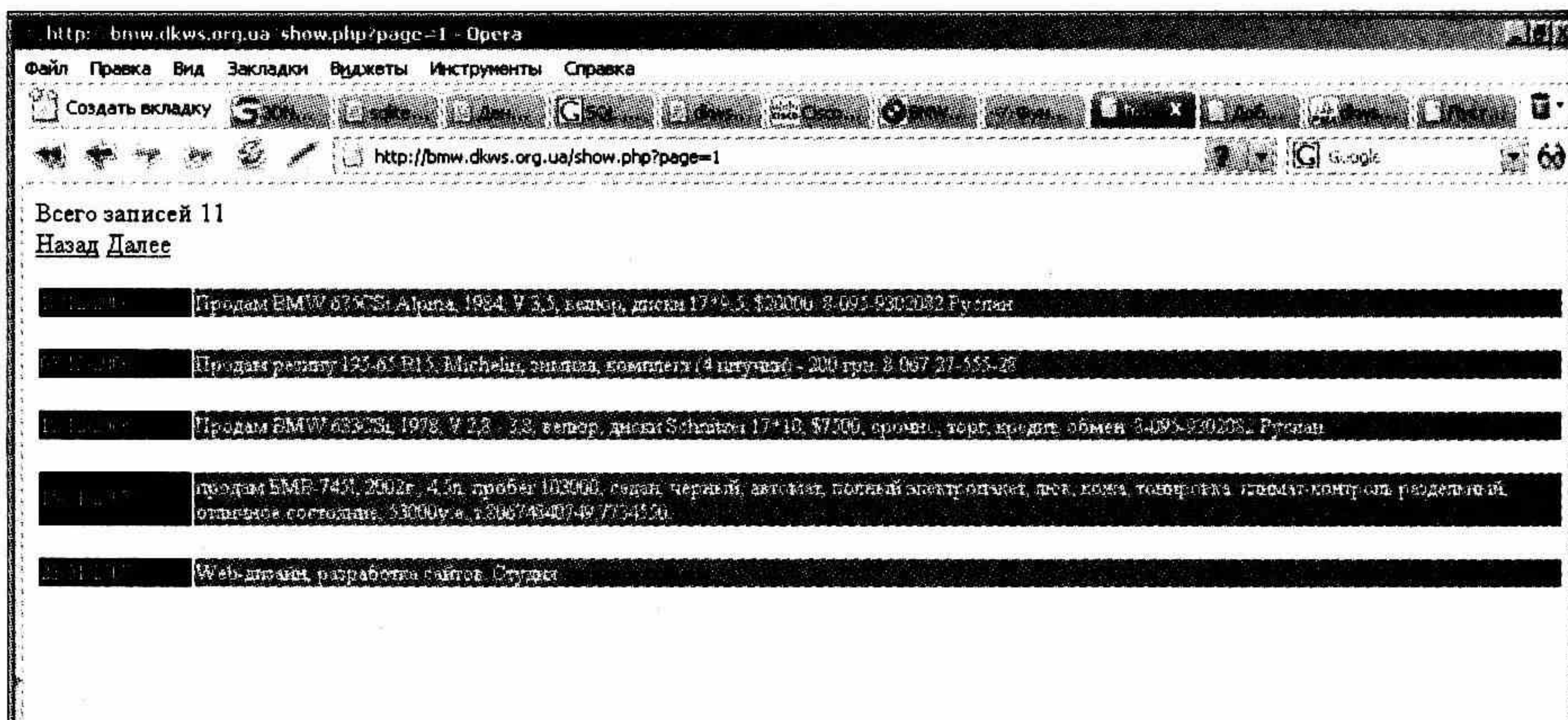
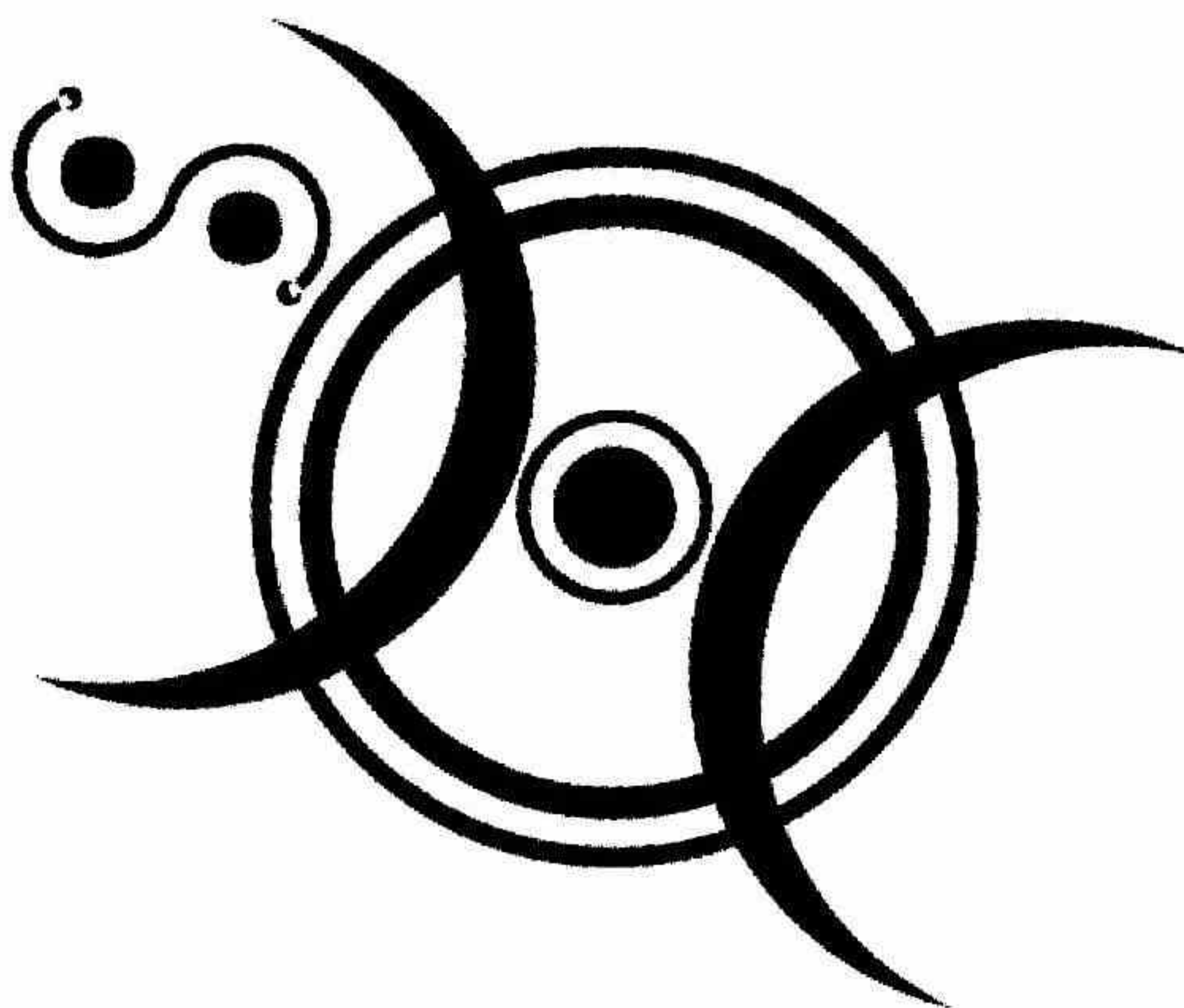


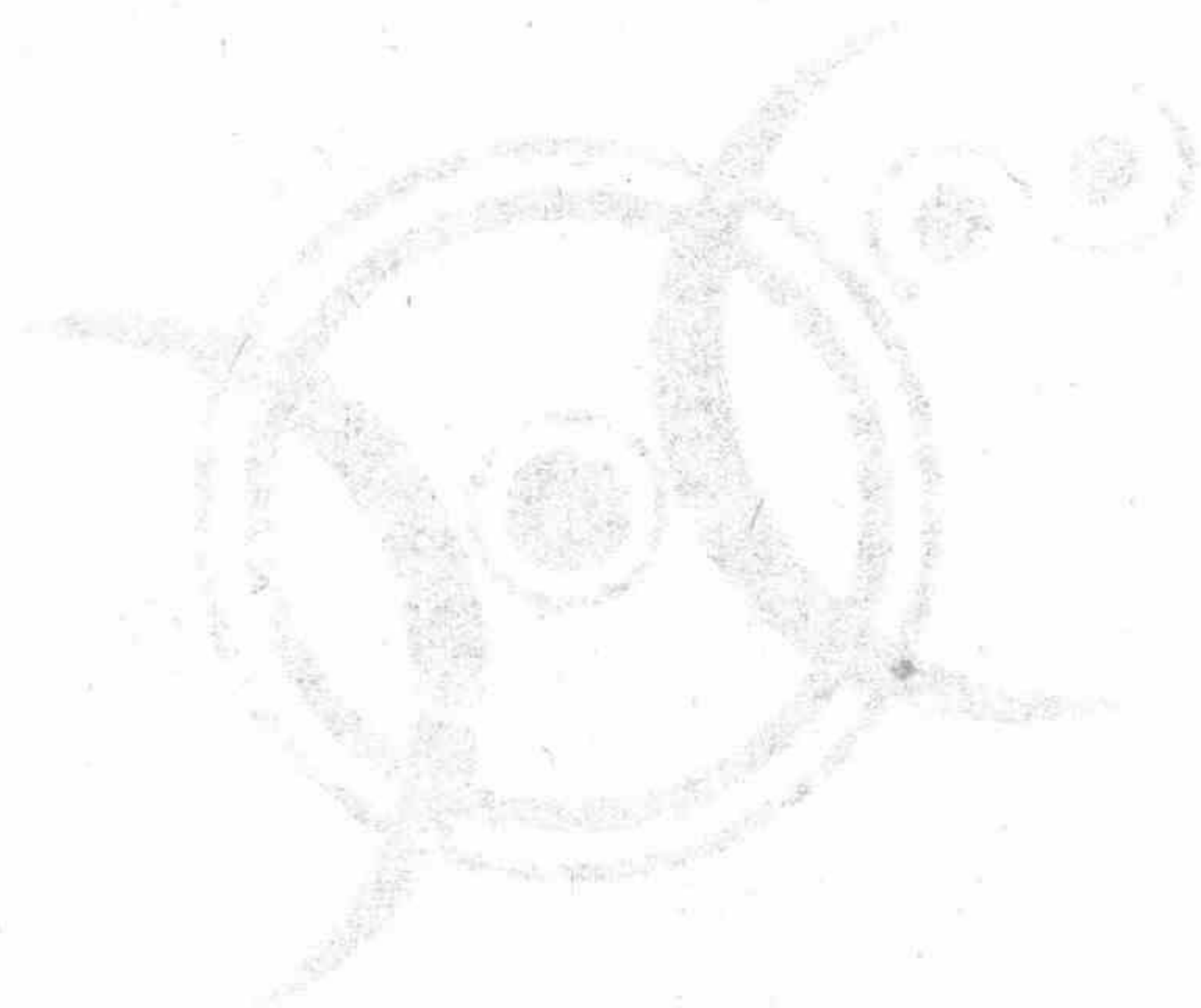
Рис. 21.3. Постраничный вывод таблицы



I. РАЗДЕЛ 6

Инструменты для создания сложных проектов

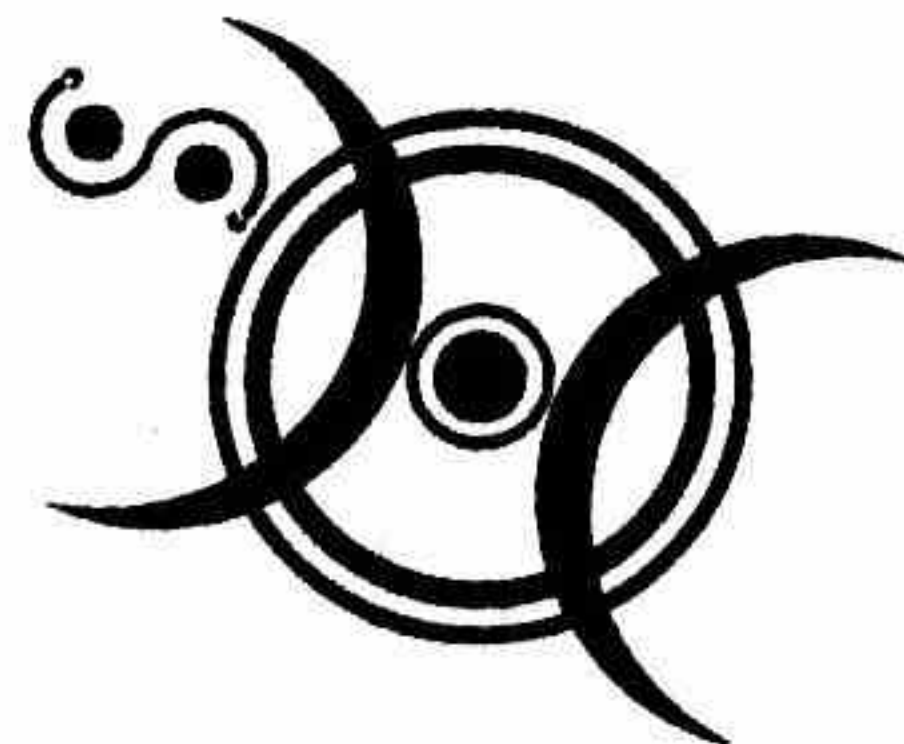
Глава 22.	Разработка собственного шаблонизатора
Глава 23.	Шаблонизатор Smarty
Глава 24.	Объектно-ориентированное программирование
Глава 25.	Механизм сессий
Глава 26.	Введение в PEAR
Глава 27.	Контроль версий
Глава 28.	Тестирование PHP-сценариев



J. P. B. 1934

THE UNIVERSITY OF MICHIGAN
LIBRARY

ГЛАВА 22



Разработка собственного шаблонизатора

22.1. Организация файлов и каталогов проекта

Основное отличие высшей лиги от всех предыдущих — это полный порядок. Порядок должен быть во всем: как в организации, так и в коде самих файлов. У вас в корневом каталоге уже лежит десятка три файлов (PHP, HTML — не важно)? А некоторые PHP-сценарии выводят с помощью `echo` HTML-код? Могу вас поздравить — у вас полный беспорядок.

Прежде всего разнесем файлы по отдельным специализированным каталогам. Понимаю, что когда у вас 30 или даже более различных файлов, вы не знаете, за что и взяться. Начнем с малого — с картинок. Все графические файлы выносим в отдельный каталог, например `images` (или `img` — сокращенно). Конечно, пути к картинкам изменятся, поэтому придется изменить несколько десятков тегов `img`, чтобы дизайн сохранил свою целостность.

После этого вынесем все HTML-файлы в каталог `html`. Но и это еще не все. HTML-файлы ведь содержат какую-то информацию. Поэтому целесообразно рассортировать их по содержимому. Например, статическую информацию, которая редко меняется ("контакты", "о нас", "карта проезда" и т. д.), желательно поместить в подкаталог `static` (`html/static`). Если у вас есть какие-то обзоры, то для них создаем каталог `reviews`, для статей — каталог `articles` и т. д. Думаю, вы поняли принцип: главное, чтобы имена подкаталогов были значимыми. Ведь каталог можно назвать и просто "a" или "bbb". Но через месяц вы не вспомните, что находится в этом каталоге, пока не откроете его. А вот когда каталог называется `articles` или `reviews`, однозначно можно сказать, что в нем лежит.

Аналогичным образом можно, кстати, рассортировать и каталог `images`. Изображения, имеющие отношение к дизайну сайта, пусть остаются в каталоге `images`, а вот картинки для других страниц — например, для статей или обзоров, желательно поместить в каталоги соответственно `images/articles` и `images/reviews`. Фотографии, понятно, помещаем в каталог `images/photos`.

Теперь самое сложное — рассортировать PHP-файлы. Можно, конечно, их оставить в корневом каталоге, если файлов немного. Но мне больше нравится, чтобы в корневом каталоге было как можно меньше файлов — например, только `index.php` и `config.php` (о нем поговорим позже). Все остальные файлы следует рассортировать так:

- ☐ если файл подключается на каком-то этапе работы сценария, тогда помещаем его в каталог `includes` (это подкаталог корневого каталога);
- ☐ если вы написали модульный движок, и PHP-файл фактически является модулем, тогда его нужно поместить в каталог `modules`;
- ☐ если файл относится к панели администрирования, тогда его помещаем в каталог `admin` или `adm` — кому как больше нравится;
- ☐ остальные файлы тоже группируем по смыслу.

В результате у нас должна получиться структура каталогов, подобная изображенной на рис. 22.1.

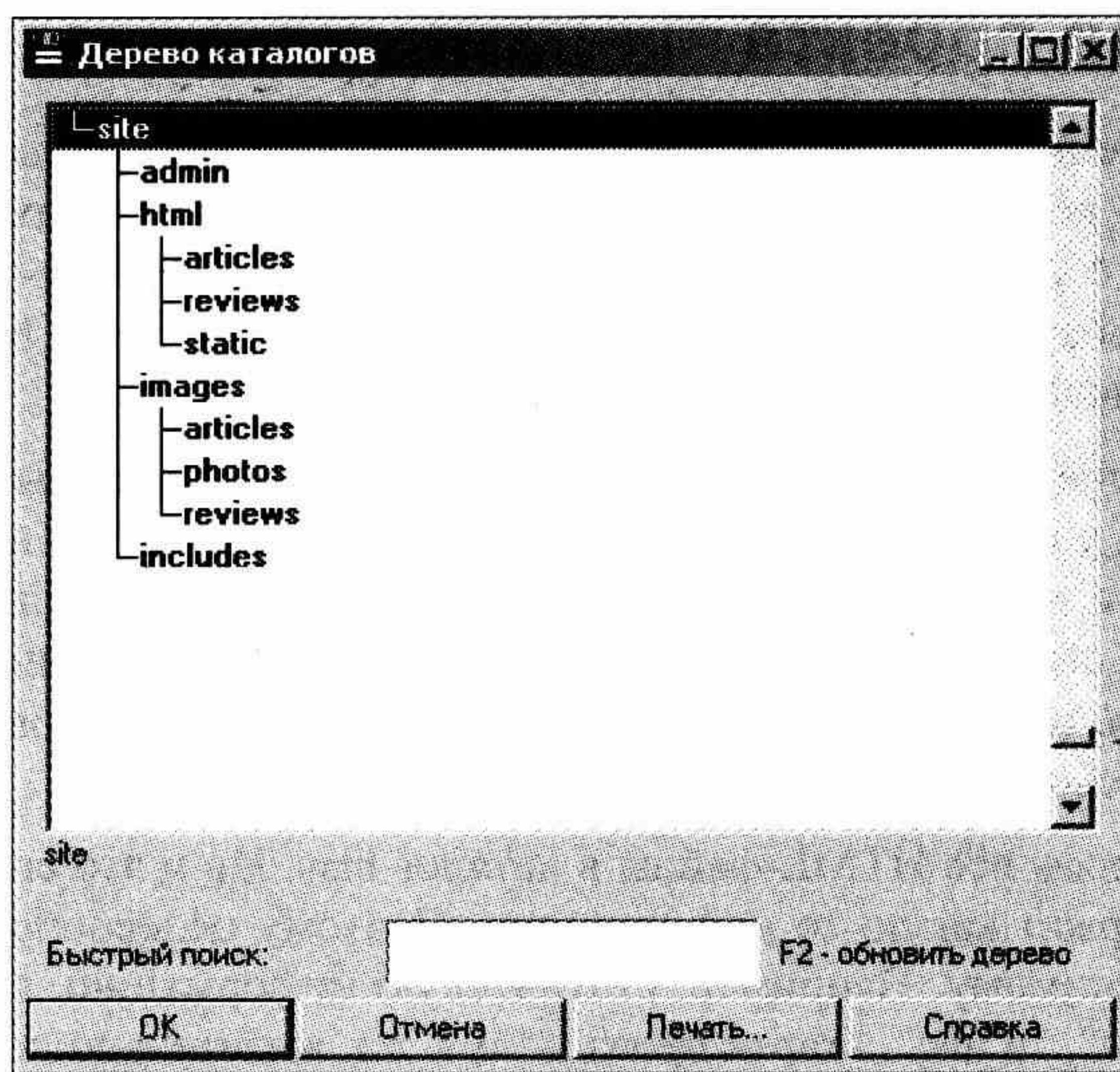


Рис. 22.1. Примерная структура каталогов

Настало время поговорить об именах файлов. Больше всего нас сейчас должны интересовать наши статьи и обзоры. Существуют две методики присвоения имен файлам. Первая заключается в том, что имя должно отображать содержимое файла. Предположим, у нас есть статья об установке Windows 8, тогда файл этой статьи можно назвать, например, так: `windows_8_install.html`.

Удобно? Конечно — взглянул на имя файла и понял, что в нем, даже не открывая его.

Вторая методика предлагает именовать файлы так:

<подраздел_номер>

или так:

<подраздел_дата_номер>

Например, у нас есть подраздел "Статьи" (Articles). Тогда файлы в нем можно именовать так:

articles_номер или article_номер

или так:

article_дата_номер

В этой методике тоже есть свои преимущества:

- эффект педантичности и порядка — ведь все файлы названы строго определенным способом;
- можно легко определить дату опубликования статьи — некоторым людям проще запомнить числа, например, дату, чем символьное название. Вы помните, что статья приблизительно была опубликована в феврале 2007 г., поэтому вам нужно просмотреть файлы article_DD_02_2007_N, где DD — число, N — порядковый номер статьи (ведь в один и тот же день можно опубликовать несколько статей).

Лично мне проще ориентироваться, когда название файла отображает его содержание, поэтому для своего сайта я выбрал именно такую методику. Она не всегда оптимальна, но в моем случае вполне приемлема. Однако если вы пишете серьезный проект — например, систему управления контентом сайта (CMS), тогда такой подход отпадает. Рассмотрим ситуацию. Заходит на сайт пользователь и хочет опубликовать статью с названием "Установка Windows 8, или Проверяем, как будет тормозить система на 512 Мбайт оперативки". Вот теперь подумайте, какой алгоритм придется написать, чтобы это название было трансформировано в windows_8_install. Правильно. Очень сложный. Я не говорю, что эта задача невыполнима, но она не стоит потраченного на нее времени. Ведь статья будет выводиться одинаково вне зависимости от того, как назван файл. Поэтому здесь целесообразнее будет такой формат имени файла для статьи:

article_ДД_ММ_ГГГГ_Номер

Согласитесь, такое имя файла сформировать намного проще. Можно и еще больше упростить задачу, назвав файлы статей так:

article_номер

Хотя, скорее всего, вы свои статьи будете хранить в базе данных, поэтому проблема с именами файлов для статей отпадет. Но останется проблема с именами файлов изображений, которые привязаны к статьям. Ведь картинку в базу данных не поместишь. Точнее, поместить-то можно (есть тип поля BLOB — не забываем), но тогда нужно будет придумать, как их оттуда извлекать при выводе той или иной статьи. Намного проще в таблице базы данных хранить HTML-код статьи, а графические файлы — отдельно на диске, в каталоге /images/articles.

Как именовать картинки? Предлагаю следующий способ:

ID-статьи_номер_картинки_в_статье<.расширение>

Предположим, что в базе есть статья с номером 25, в которой три картинки формата JPEG. Тогда файлы картинок должны называться так: 25_01.jpg, 25_02.jpg, 25_03.jpg. Просто и удобно.

22.2. Выносим параметры в отдельный файл

Наверняка в ваших PHP-сценариях будут какие-то константы, влияющие на работу всего сценария, — например, имя сервера MySQL, имя пользователя, пароль, количество сообщений на странице и т. д. Все эти параметры целесообразно вынести в отдельный файл, который обычно называется `config.php`.

Делается это из следующих соображений:

- ❑ если у вас два или более PHP-файлов задействуют эти параметры, то изменение параметров будет производиться проще, ведь параметр нужно будет изменить только один раз — в конфигурационном файле, а не в каждом файле проекта;
- ❑ даже если эти параметры необходимы только в одном файле, все равно желательно их вынести в `config.php`, чтобы пользователю, который потом скачает и будет использовать ваш проект, не пришлось копаться в вашем коде в поисках нужных параметров.

Рассмотрим небольшой пример файла `config.php` (листинг 22.1).

Листинг 22.1. Пример файла `config.php`

```
<?php

// имя сервера MySQL
$SERVER = 'localhost';

// имя пользователя и пароль
$USER = 'user';
$PASSWORD = '12345678';

// база данных
$DB = 'cms';

// число сообщений на странице (для постраничного вывода)
$messages = 10;

// e-mail администратора:
$mailto = 'webmaster@somehost.ru';

?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл `config.php`), представленный в каталоге *Glava_22* сопровождающего книгу электронного архива (см. приложение 4).

22.3. Подключение дополнительных файлов

22.3.1. Инструкции *include* и *require*

Инструкции `include`, `require`, `include_once`, `require_once` предназначены для подключения дополнительных файлов — например, того же файла конфигурации `config.php`.

Обычно дополнительные РНР-файлы подключаются с помощью инструкции `include`:

```
include "имя_файла.php";
```

Мы все привыкли к этой инструкции. Разберемся, как ее обрабатывает интерпретатор. Сначала РНР преобразует сценарий во внутреннее представление, именно поэтому его называют компилирующим интерпретатором или интерпретирующим компилятором, т. е. РНР — это нечто среднее между интерпретатором и компилятором. После преобразования РНР начинает выполнять код, строка за строкой. Если ему встретится инструкция `include`, то он загружает файл, указанный в этой инструкции, и полностью переключается на его выполнение — этот файл также преобразуется во внутреннее представление, а потом поэтапно выполняется.

Инструкция `require` работает иначе — файл, подключенный с ее помощью, транслируется до выполнения сценария. Иными словами, на момент выполнения сценария файл уже оттранслирован. Отсюда вывод — если использовать `require`, то ваши сценарии будут работать быстрее. Поэтому инструкция `require` предпочтительнее.

22.3.2. Альтернативный способ подключения сценариев

Нужно отметить, что у инструкций `include` и `require` имеется определенная особенность — подключенный файл становится неотъемлемой частью сценария. Предположим, у нас есть основной файл `index.php` (листинг 22.2) и дополнительный — `connect.php` (листинг 22.3).

Листинг 22.2. Основной файл — `index.php`

```
<?php

// выводим заголовок
echo join('', file('header.html'));

// подключаемся к базе данных
include "connect.php";

// выводим футер
echo join('', file('footer.html'));

?>
```


Листинг 22.3. Дополнительный файл — connect.php

```
<?php

if (!mysql_connect("localhost", "root", ""))
    die('Ошибка подключения к серверу');

mysql_select_db("db");

?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовые варианты (файлы *index.php* и *connect.php*), представленные в каталоге *Glava_22* сопровождающего книгу электронного архива (см. приложение 4).

Допустим, что произошла ошибка подключения к серверу баз данных. Будет выполнена инструкция `die()`, которая прервет выполнение не дополнительного, а основного сценария, поскольку дополнительный сценарий инструкцией `include` просто встраивается в основной. В результате мы увидим сообщение об ошибке, но не увидим нижнюю часть дизайна. Это нехорошо. Выход из этой ситуации найти можно. Есть даже два выхода: во-первых, можно изменить сценарий *connect.php*:

```
if (mysql_connect("localhost", "root", ""))
    mysql_select_db("db");
else echo "Ошибка подключения к базе данных";
```

Другими словами, сделать так, чтобы не было инструкции `die()`. Во-вторых, можно подключать файлы (сценарии) с помощью инструкции `file()`. Но только при подключении сценария, который не выполняет сервисных или подготовительных действий для основного сценария. Так что в нашем случае инструкцию `file()` использовать нельзя — ведь сценарий *connect.php* как раз и подключает основной сценарий к базе данных.

А вот при подключении сценария, который не выполняет сервисных или подготовительных действий для основного сценария, а просто выводит информацию и завершает свою работу, можно обойтись инструкцией `file()`. Пусть имеется сценарий *news.php*, выводящий новости (листинг 22.4).

Листинг 22.4. Сценарий news.php

```
<?php

if (!mysql_connect("localhost", "root", ""))
    die('Ошибка подключения к серверу');

mysql_select_db("db");
$r = mysql_query('select * from news');
```



```
while ($row = mysql_fetch_array($r))  
    echo "<p>$row[dt] $row[txt]";  
  
?>
```

В этом случае сценарий `news.php` можно подключить с помощью инструкции `file()`:

```
echo join('', file('http://server.ru/news.php'));
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл `news.php`), представленный в каталоге *Glava_22* сопровождающего книгу электронного архива (см. приложение 4).

Обратите внимание — нужно указывать URL файла, а не путь к файлу, иначе вместо результата работы будет выведен исходный код файла.

Еще раз повторяюсь, что данный способ не подходит для сервисных сценариев, без включения кода которых в основной сценарий дальнейшее продолжение работы последнего невозможно.

22.3.3. Инструкции *include_once* и *require_once*

Помимо инструкций `include` и `require` существуют инструкции однократного включения: `include_once` и `require_once`, которые работают так же, как и обычные `include` и `require`, но перед подключением файла проверяют, не был ли он уже подключен. Дело в том, что развивая большой проект, вы можете забыть, когда нужно подключать тот или иной файл, и ошибочно подключить какой-то PHP-файл несколько раз. Чтобы такого не произошло, рекомендуется применять инструкции `include_once` и `require_once`.

22.4. Шаблоны

Давайте на минутку остановимся, чтобы задуматься, какую работу мы проделали. Во-первых, мы навели порядок с именами файлов и каталогов. Во-вторых, вынесли лишний код из PHP-файлов в отдельные файлы — так намного удобнее. Теперь настал черед вынести из наших сценариев HTML-код.

Даже если вы начинающий PHP-программист, то, скорее всего, будете хранить HTML-код отдельно от сценария в HTML-файле. Ведь так удобнее. А в нужном месте с помощью инструкции `file()` можно прочитать содержимое HTML-файла и оператором `echo` вывести его в браузер. Но иногда встраивать HTML-код в сценарий все же приходится.

Предположим, в нашем сценарии есть следующие операторы:

```
echo "<p>Добро пожаловать <b>$user</b>";  
echo "<p>Дата вашего последнего визита: <b>$last</b>";
```



```
echo "<p>Дата регистрации: <b>$reg_date</b>";  
echo "<p>Вами оставлено сообщений: <b>$messages</b>";  
echo "<p>Новых личных сообщений: <b>$personal</b>";
```

Другими словами, наш PHP-сценарий содержит HTML-код, который нельзя поместить в обычный HTML-файл, поскольку его нужно модифицировать и передавать в браузер.

Если вы сами планируете поддерживать свой сценарий, то вроде бы ничего страшного здесь нет. Но вы только представьте лицо дизайнера, не имеющего никакого отношения к PHP и к программированию вообще, когда он откроет PHP-файл, чтобы изменить какой-то HTML-код. Ведь настоящий сценарий — это не эти простые пять строк, а сотни строк PHP-кода, среди которого иногда мелькает HTML-код. Разбираться во всем этом сложно, к тому же велика вероятность ошибки. Поэтому следует вообще исключить HTML-код из PHP-файла. Для этого нам придется использовать шаблоны, вынося HTML-код в отдельные файлы, как это сделано для нашего HTML-кода, который вынесен в файл `welcome.tpl` (листинг 22.5).

Листинг 22.5. Файл `welcome.tpl`

```
<p>Добро пожаловать <b>{USER}</b>  
<p>Дата вашего последнего визита: <b>{LAST}</b>  
<p>Дата регистрации: <b>{REG_DATE}</b>  
<p>Вами оставлено сообщений: <b>{MESSAGES}</b>  
<p>Новых личных сообщений: <b>{PERSONAL}</b>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл `welcome.tpl`), представленный в каталоге `Glava_22` сопровождающего книгу электронного архива (см. приложение 4).

Наверняка вы уже сталкивались с такими файлами (вспомните, как редактировать тему форума PHPBB). Дизайнерам сайтов данный формат тоже знаком, поэтому особого труда с редактированием файла шаблона у них не будет.

Далее в этой главе мы напишем очень простой шаблонизатор, т. е. сценарий, позволяющий обрабатывать TPL-шаблоны. Однако возможностей нашего шаблонизатора вполне хватит для создания на его основе более серьезных проектов.

Итак, шаблон у нас уже есть. Напишем файл `index.php`, который будет работать с шаблоном, используя наш еще не созданный шаблонизатор (листинг 22.6).

Листинг 22.6. Файл `index.php`

```
<?php  
// подключаем шаблонизатор  
require "template.php";  
  
// определяем переменные, которые нужно внедрить в HTML-код  
$USER = "Денис";
```



```
$LAST = "24.02.2007";  
$REG_DATE = "01.02.2007";  
$MESSAGES = 34;  
$PERSONAL = 0;  
  
// открываем шаблон  
$tpl->get_tpl('welcome.tpl');  
  
// устанавливаем переменные шаблона  
$tpl->set_value('USER',$USER);  
$tpl->set_value('LAST',$LAST);  
  
$tpl->set_value('REG_DATE',$REG_DATE);  
$tpl->set_value('MESSAGES',$MESSAGES);  
  
$tpl->set_value('PERSONAL',$PERSONAL);  
  
// запускаем парсинг шаблона  
$tpl->tpl_parse();  
// выводим HTML  
echo $tpl->html;  
?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *22-6.php*), представленный в каталоге *Glava_22* сопровождающего книгу электронного архива (см. приложение 4).

Вот теперь полный порядок — из PHP-файла исключен HTML-код. Для открытия шаблона применяется метод `get_tpl()`, а для установки значения шаблона — `set_value()`. Первый параметр метода — это имя переменной в шаблоне, а второй — значение этой переменной. Метод `tpl_parse` выполняет парсинг нашего шаблона. Все, что нам остается сделать, — это вывести уже готовый HTML-код (рис. 22.2).

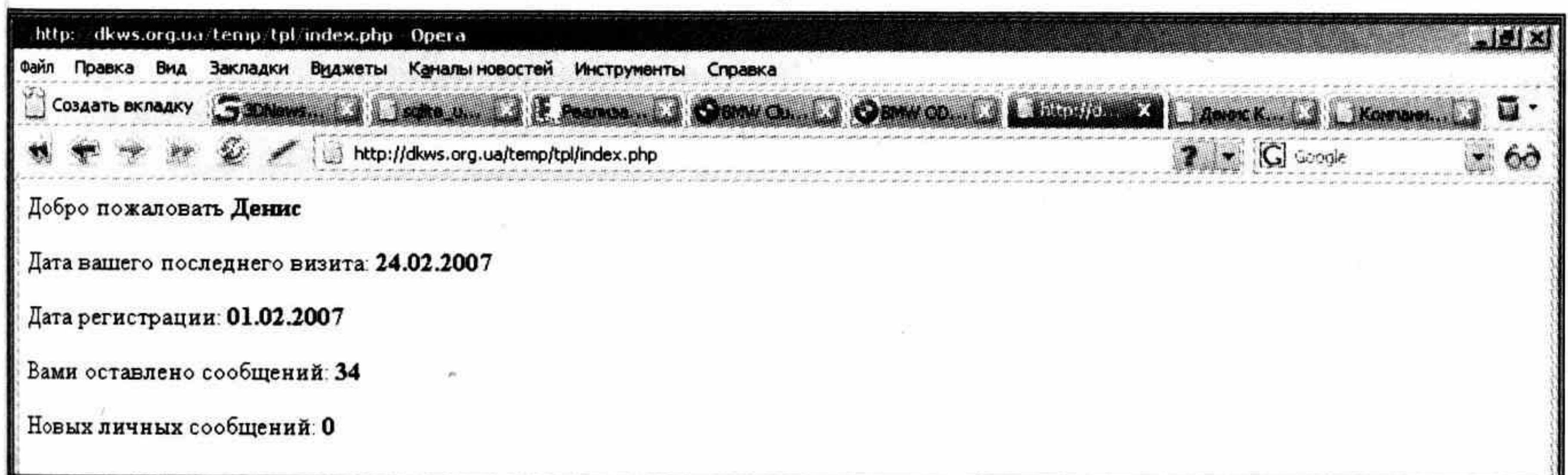


Рис. 22.2. Шаблонизатор в действии

Осталось рассмотреть только сам файл шаблонизатора `template.php` (листинг 22.7).

Листинг 22.7. Шаблонизатор

```
<?php
// класс шаблона
class template_class
{
    var $values      = array();      // переменные шаблона
    var $html;        // HTML-код

// функция загрузки шаблона
    function get_tpl($tpl_name)
    {
        if(empty($tpl_name) || !file_exists($tpl_name))
        {
            return false;
        }
        else
        {
            $this->html = join('',file($tpl_name));
        }
    }

// функция установки значения
    function set_value($key,$var)
    {
        $key = '{' . $key . '}';
        $this->values[$key] = $var;
    }

// парсинг шаблона
    function tpl_parse()
    {
        foreach($this->values as $find => $replace)
        {
            $this->html = str_replace($find, $replace, $this->html);
        }
    }
}

// экземпляр класса
$tpl = new template_class;
?>
```

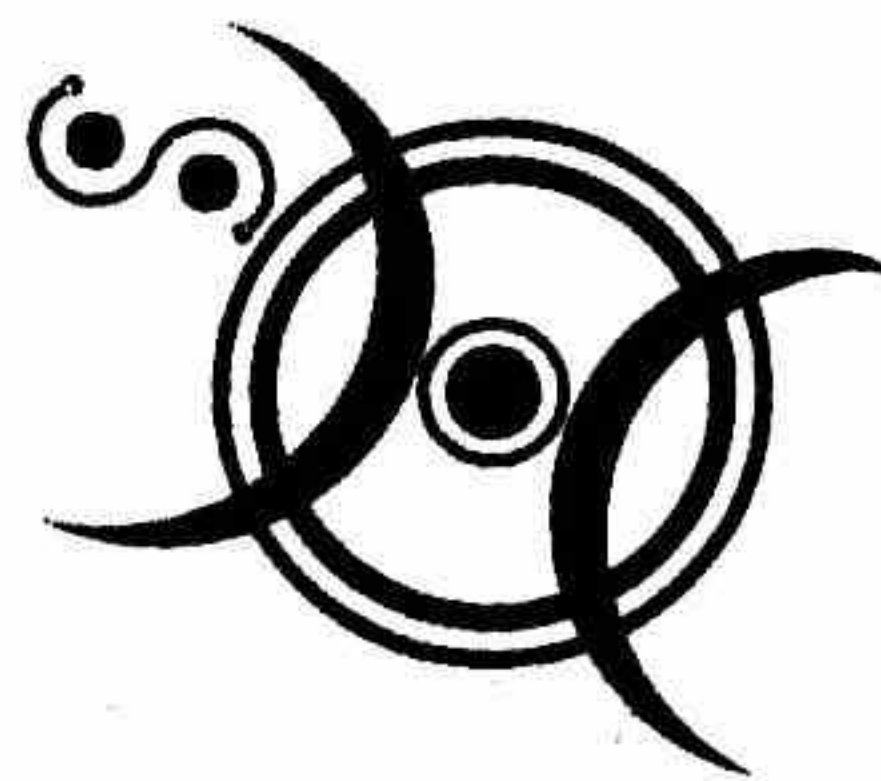
ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *template.php*), представленный в каталоге *Glava_22* сопровождающего книгу электронного архива (см. приложение 4).

Как видите, наш шаблонизатор примитивен. Весь парсинг шаблона заключается в обычной замене строк вида {название_переменной} переданными значениями. Но, тем не менее, нужно отметить, что шаблонизатор работает, а большего нам и не надо. Если вам потребуется больше свободы, то вам поможет следующая глава, в которой мы рассмотрим шаблонизатор Smarty. Его возможности намного шире, но в то же время он значительно сложнее нашего.

Напоследок добавлю, что для поддержания порядка нужно создать каталог `templates` и поместить в него все TPL-файлы.

ГЛАВА 23



Шаблонизатор Smarty

23.1. Что такое Smarty?

Мы не зря создали в предыдущей главе несложный шаблонизатор — так вам будет проще понять, что такое Smarty. Smarty, как и наш простенький шаблонизатор, позволяет решить задачу исключения HTML-кода из сценария. Преимущества очевидны — код становится проще, а дизайнер сайта получает возможность удобного редактирования HTML-кода, поскольку ему не нужно бояться, что он нечаянно изменит PHP-код, и сценарий потом не будет работать.

У нашего простенького шаблонизатора (см. главу 22) есть существенный недостаток — если шаблоны будут объемными, то он станет работать довольно медленно. У Smarty такого недостатка нет, поскольку он является компилирующим шаблонизатором. Иными словами, Smarty сначала создает PHP-код из шаблона, а потом выполняет его. Причем код создается всего один раз и при последующих запусках сценария только выполняется. За счет этого сильно сокращается время обработки шаблона. Кроме того, поскольку шаблон преобразуется в PHP-код, то этот код может быть оптимизирован с помощью Zend Accelerator (<http://www.zend.com>) или PHP Accelerator (<http://www.php-accelerator.co.uk>), если один из этих оптимизаторов у вас установлен.

Но скорость выполнения — это еще далеко не все. По сути Smarty представляет собой... язык программирования. Хотя вы можете возразить: мол, искали шаблонизатор, чтобы уберечь нашего дорогого дизайнера от знакомства с PHP, а вернулись к тем же условным операторам!

23.2. Установка Smarty

Шаблонизатор Smarty доступен по адресу www.smarty.net. В каталоге Glava_23 сопровождающего книгу электронного архива (см. приложение 4) вы найдете файл Smarty-2.6.26.zip с последней стабильной (2.6.26) версией Smarty на момент написания этих строк, а также файл Smarty-3.0b5.zip с самой новой (но бета) версией шаблонизатора. Там же в файле Smarty-2.6.7-docs.pdf содержится официальное руководство по Smarty на русском языке в формате PDF.

Для установки Smarty потребуется веб-сервер с установленным PHP 4.0.6 или более свежей версией. Распакуйте содержимое каталога `libs` на свой сервер. Куда именно распаковать, зависит от вас. Если у вас собственный сервер, к которому обеспечен полный доступ, то желательно распаковать каталог `libs` за пределами корневого каталога веб-сервера — например, в каталог `/usr/local/lib/php/`. Кроме того, каталог `/usr/local/lib/php/libs` вы должны указать в переменной `include_path` файла конфигурации `php.ini`.

ПРИМЕЧАНИЕ

Значение переменной `include_path` можно определить и во время выполнения сценария с помощью функции `ini_set()`, например:

```
ini_set("include_path", ".:../:/include:../include");
```

Если же вы арендуете хостинг, тогда выбора у вас нет, и вам следует распаковать каталог `libs` в каталог `public_html` — это корневой каталог документов веб-сервера. В дальнейшем мы будем иметь в виду именно такое размещение.

В каталоге `public_html` нужно создать четыре каталога:

- `templates` — будет содержать шаблоны Smarty;
- `templates_c` — сюда Smarty будет записывать откомпилированные шаблоны;
- `cache` — каталог кэша;
- `config` — конфигурационный каталог.

К каталогам `templates_c` и `cache` необходимо установить права доступа 777:

```
chmod 777 templates_c
chmod 777 cache
```

Если у вас свой сервер, вы можете ограничить права доступа до 770, но при условии, что вы точно знаете, от имени какого пользователя запускается веб-сервер. Если у вас веб-сервер Apache, то он, в зависимости от версии, работает или от имени пользователя `apache`, или от имени пользователя `nobody`. Вот пример установки прав для пользователя `nobody`:

```
chown nobody:nobody /var/www/templates_c/
chmod 770 /var/www/templates_c/
chown nobody:nobody /var/www/cache/
chmod 770 /var/www/cache/
```

Разумеется, путь у вас будет другой, поэтому не следует вслепую копировать приведенные команды.

Вернемся к нашим четырём каталогам. Их имена сугубо условные, т. е. вам совсем не обязательно называть их именно так. Каталог `templates` можно назвать, например, `skin`, а каталог `templates_c` — `skin_c`. Но далее в этой книге мы будем подразумевать, что созданы именно эти каталоги: `templates`, `templates_c`, `cache` и `config`.

Теперь проверим, работает ли Smarty. В каталог `templates` поместите файл `test.tpl`, содержащий всего одну строку:

```
Привет, {$name}!
```


В каталоге `public_html` создайте файл `smarty-test.php` (листинг 23.1).

Листинг 23.1. Файл `smarty-test.php`

```
<?php

// определяем переменную SMARTY_DIR
define('SMARTY_DIR', 'libs/');

// загружаем библиотеку Smarty
require_once('libs/Smarty.class.php');

// создаем объект Smarty
$smarty = new Smarty();

// указываем каталоги
$smarty->template_dir = 'templates/';
$smarty->compile_dir = 'templates_c/';
$smarty->config_dir = 'configs/';
$smarty->cache_dir = 'cache/';
// определяем значение константы name в шаблоне
$smarty->assign('name', 'Denis');
// отображаем шаблон
$smarty->display('test.tpl');
?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл `23-1.php`), представленный в каталоге *Glava_23* сопровождающего книгу электронного архива (см. приложение 4).

Константа `SMARTY_DIR` содержит путь к Smarty. Если вы не знаете полного пути (например, в случае с хостингом, а не выделенным сервером), тогда лучше указать относительный путь, как показано в листинге.

Затем мы подключаем библиотеку Smarty. Если бы путь к Smarty был указан в `include_path`, можно было бы написать просто:

```
require_once('Smarty.class.php');
```

Но для большей универсальности рекомендуется подключать библиотеку так:

```
require_once(SMARTY_DIR . 'Smarty.class.php');
```

Теперь в поле адреса вашего браузера введите следующий URL:

```
http://ваш_сайт/smarty_test.php
```

Если вы увидите строку **Привет, Denis!** (рис. 23.1), значит, Smarty работает. Если же вместо заветной строки вы увидели список ошибок, то первым делом следует проверить:

- ☐ права доступа к каталогам `template_c` и `cache_c`;
- ☐ правильность указания константы `SMARTY_DIR`.

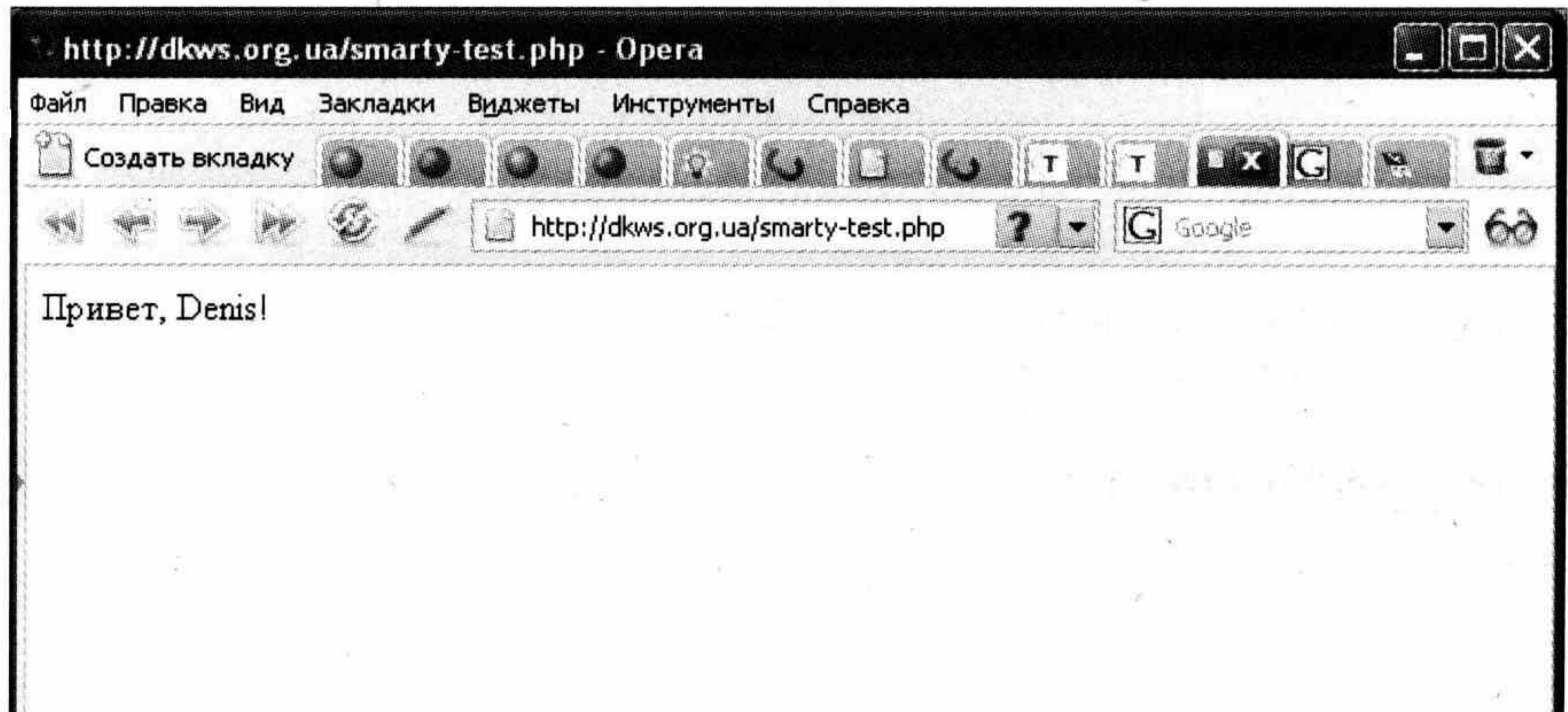


Рис. 23.1. Smarty работает!

23.3. Создание файла setup.php

Для большей гибкости рекомендуется создать файл `setup.php`, в который следует вынести все подготовительные действия: установку константы `SMARTY_DIR`, подключение библиотеки `Smarty`, установку служебных каталогов. Рассмотрим пример файла `setup.php` (листинг 23.2).

Листинг 23.2. Файл `setup.php`

```
<?php

define('SMARTY_DIR', 'libs/');

require(SMARTY_DIR . 'Smarty.class.php');

class MySmarty extends Smarty {

    function MySmarty ()
    {
        // конструктор класса (имя конструктора совпадает с именем
        // класса)
        // конструктор автоматически вызывается при создании нового
        // экземпляра

        $this->Smarty();

        $this->template_dir = 'templates/';
        $this->compile_dir  = 'templates_c/';
        $this->config_dir   = 'config/';
        $this->cache_dir    = 'cache/';
    }
}
```



```
$this-> caching = true;  
}  
  
}  
?>
```

Тогда наш файл `smarty_test.php` (вторую редакцию назовем `smarty_test2.php`) будет выглядеть так, как показано в листинге 23.3.

Листинг 23.3. Файл `smarty_test2.php`

```
<?php  
  
require('setup.php');  
  
$smarty = new MySmarty;  
  
$smarty->assign('name', 'Denis');  
  
$smarty->display('test.tpl');  
  
?>
```

Как видите, использование файла `setup.php` немного упрощает код основного сценария.

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовые варианты (файлы `23-2.php` и `23-3.php`), представленные в каталоге *Glava_23* сопровождающего книгу электронного архива (см. приложение 4).

23.4. Разработка шаблонов Smarty

В этом разделе мы поговорим о правилах написания TPL-шаблонов Smarty. С основными правилами мы уже знакомы, поскольку ранее применяли шаблонизатор (см. главу 22). Но тот шаблонизатор был очень прост, если не сказать, примитивен. Возможности Smarty намного шире.

23.4.1. Комментарии в шаблонах

В шаблонах Smarty вы можете использовать комментарии, которые не будут отправлены в браузер. Пример приведен в листинге 23.4.

Листинг 23.4. Пример использования комментариев

```
<html>  
<body>
```



```
{* простой комментарий *}
{*
    многострочный
    комментарий
*}

<!-- это HTML-комментарий, он будет отправлен браузеру -->

</body>
</html>
```

23.4.2. Переменные в Smarty

Наш простой шаблонизатор самостоятельно устанавливал значения переменных шаблона, т. е. непосредственно в самом шаблоне нельзя было установить значение той или иной переменной. Smarty же позволяет устанавливать значения переменных в шаблоне, кроме того — в шаблоне задавать переменные-массивы и статические данные.

Если вам нужно отобразить простую переменную, то она записывается так:

```
{$var}
```

В случае с числовыми и ассоциативными массивами код Smarty записывается так:

```
{* 5 элемент числового массива *}
{$arr[4]}

{* отображает значение элемента key ассоциативного массива assoc,
    подобно как в PHP $assoc[key] *}
{$assoc.key}
```

Если имя ключа ассоциативного массива не известно (помните, `$f[$key]`), то код Smarty выглядит так:

```
{$assoc.$key}
```

Кроме простых переменных и переменных-массивов Smarty поддерживает еще и объекты. Свойство объекта отображается с помощью оператора `->`:

```
{$object->var}
```

С помощью этого оператора можно также отобразить значение, возвращаемое методом объекта:

```
{$object->method() }
```

Оператором `assign` можно присвоить значение переменной:

```
{assign var=foo value="строка"}{$foo}
```

В результате такой Smarty-код выведет значение 'строка'. Об операторе `assign` мы еще поговорим.

При необходимости Smarty позволяет передать и статическое значение, хотя намного проще обойтись в этом случае без Smarty. Статические значения передаются так:

```
{"строка"}
```

23.4.3. Файлы конфигурации шаблонов

Для каждого шаблона вы можете создать файл конфигурации. Да, звучит дико, но так оно и есть. Файлы конфигурации шаблонов позволяют полностью исключить HTML-код из PHP-сценариев. Также они позволяют более гибко работать с самими шаблонами. Предположим, что вам нужно задать некоторые параметры страницы — например: заголовок страницы (пусть это будет переменная `title`), цвет фона (переменная `BgColor`) и размер рамки таблицы (`BorderSize`). Если устанавливать значения этих переменных через метод `assign()` объекта Smarty, то для их редактирования веб-дизайнеру придется открывать PHP-файлы, чего он делать не должен. Если же держать эти значения в самом шаблоне, получается не очень наглядно, да и не совсем гибко. Для достижения максимальной наглядности необходимо использовать файлы конфигурации.

Итак, создадим файл конфигурации `main.conf` в каталоге конфигураций (у нас это каталог `config`). Пример такого файла приведен в листинге 23.5.

Листинг 23.5. Пример файла конфигурации (`main.conf`)

```
title = "Мой сайт"
BgColor = "black"
BorderSize = 1
```

Подключить файл конфигурации к шаблону можно с помощью функции `config_load`:

```
{config_load file="main.conf"}
```

В самом шаблоне переменные конфигурации записываются так:

```
{#имя_переменной_из_файла_конфигурации#}
```

Пример шаблона с использованием файла конфигурации приведен в листинге 23.6.

Листинг 23.6. Использование файла конфигурации

```
{config_load file="main.conf"}
<html>
<title>{#title#}</title>

<body bgcolor="{#BgColor#}">
<table border="{#BorderSize#}" >
```



```
<tr>
    <td>Пользователь</td>
    <td>ФИО</td>
    <td>E-mail</td>
</tr>
</table>
</body>
</html>
```

23.4.4. Служебная переменная `{$smarty}`

Служебная переменная `{$smarty}` позволяет получить доступ к специальным переменным, в том числе и к переменным, которые были переданы с помощью методов GET и POST. Честно говоря, вряд ли дизайнер будет заниматься передачей переменных, поскольку это задача программиста. Но, тем не менее, Smarty позволяет работать с такими переменными. Доступ к ним осуществляется через переменную `{$smarty}`:

```
{$smarty.get.var}
{$smarty.post.var}
```

В первом случае осуществляется доступ к переменной `var`, переданной с помощью метода GET. Во втором случае Smarty получает доступ к переменной `var`, которая была передана методом POST.

Передать переменные нужно тому сценарию, который задействует шаблоны Smarty. Предположим, у вас есть шаблон `index.tpl`, в котором вы хотите с помощью Smarty вывести переданные переменные способом GET. Шаблон `index.tpl` используется файлом `index.php`. Так вот переменные следует передать этому файлу:

```
http://ваш_сервер/index.php?var=значение
```

Smarty также позволяет получить доступ к переменным окружения, переменным сервера, Cookies и переменным сессии:

```
{* доступ к переменным окружения *}
{$smarty.env.имя_переменной}
```

```
{* доступ к переменным сервера *}
{$smarty.server.имя_переменной}
```

```
{* доступ к переменным Cookies *}
{$smarty.cookies.имя_переменной}
```

```
{* доступ к переменным сессии *}
{$smarty.session.имя_переменной}
```

С помощью этой служебной переменной можно вывести текущее время и имя текущего шаблона:


```
{* выводим текущее время и дату с помощью
  модификатора date_format *}
{$smarty.now|date_format:"%d-%m-%Y %H:%M:%S"}
```

```
{* выводим имя шаблона *}
{$smarty.template}
```

Остальные возможности переменной `{$smarty}` мы рассматривать не будем. Как уже было отмечено, дизайнеру вряд ли придется заниматься выводом тех или иных значений, поскольку это работа программиста: подготовить все для дизайнера и с помощью метода `assign` присвоить переменным шаблона значения. Другое дело, когда речь идет о жестком разграничении кода и шаблонов, когда дизайнер вообще не имеет права доступа к коду. Если дожидаться программиста времени нет, а нужно немного модифицировать сценарий, добавив вывод переменной, то можно просто вывести значение этой переменной с помощью `Smarty`.

23.4.5. Модификаторы переменных

Модификаторы применяются к переменным `Smarty`, со значениями которых они могут производить несложные операции. С одним модификатором, `date_format`, мы уже знакомы. Модификаторы указываются так:

```
{переменная|модификатор:параметр1:параметр2}
```

Параметров может быть несколько, а может и вообще не быть. Если они есть, то разделение параметров осуществляется с помощью точки с запятой.

В табл. 23.1 представлены возможные модификаторы и примеры их использования.

Таблица 23.1. Примеры использования модификаторов переменных

Модификатор	Описание
<code>capitalize</code>	Делает первые буквы каждого слова прописными, единственный параметр определяет, будет ли действие модификатора распространяться на слова с цифрами. По умолчанию данный параметр равен <code>false</code> . Пример использования: <code>{ \$articleTitle capitalize }</code> <code>{ \$articleTitle capitalize:true }</code>
<code>cat</code>	Добавляет к переменной строку, указанную в качестве единственного параметра: <code>{ \$articleTitle cat:" эта строка будет добавлена к переменной " }</code>
<code>count_characters</code>	Подсчитывает и возвращает количество символов в строке. Пример: <code>{ \$articleTitle count_characters }</code>
<code>count_paragraphs</code>	Подсчитывает и возвращает число абзацев в тексте
<code>count_sentences</code>	Подсчитывает и возвращает число предложений в тексте

Таблица 23.1 (продолжение)

Модификатор	Описание
count_words	Подсчитывает и возвращает число слов в тексте
date_format	Выводит дату в заданном с помощью параметра формате. По умолчанию используется формат %b %e, %Y. Формат даты указывается так же, как и для функции date() PHP. Примеры: <pre>{Smarty.now date_format} {Smarty.now date_format:"%A, %B %e, %Y"} {Smarty.now date_format:"%H:%M:%S"}</pre>
default	Очень полезный модификатор, позволяющий задать значение по умолчанию, если переменная не содержит значения: <pre>{ArticleTitle default:"no title"}</pre>
escape	"Защищает" специальные символы в переменной. Для правильной "защиты" нужно с помощью первого параметра указать ожидаемое содержимое переменной: html — HTML-код (по умолчанию); url — адрес URL; javascript — JavaScript; quotes — используется для защиты одиночных кавычек; hex — служит для преобразования каждого символа в шестнадцатеричное представление; hexentity — используется для преобразования каждого символа в шестнадцатеричное представление в формате HTML
	Примеры: <pre>{ArticleTitle escape} {ArticleTitle escape:"html"} {* escapes & " ' < > *} {ArticleTitle escape:"htmlall"} {* escapes ALL html entities *} {ArticleTitle escape:"url"} {ArticleTitle escape:"quotes"}</pre>
indent	Вставляет заданное число указанных символов в начало строки. По умолчанию вставляются четыре пробела. Первый параметр задает число символов, а второй — сам символ. Примеры: <pre>{ArticleTitle indent} {ArticleTitle indent:5} {ArticleTitle indent:1:"\t"}</pre>
lower	Переводит символы в нижний регистр
nl2br	Заменяет все символы \n или комбинации символов \r\n на тег

Таблица 23.1 (окончание)

Модификатор	Описание
regex_replace	Выполняет поиск подстроки с помощью заданного регулярного выражения и производит замену найденной подстроки на строку, заданную вторым параметром: <code>{ \$переменная regex_replace: "рег_выражение": "строка" }</code>
replace	Используется для замены в строке. Первый параметр — это подстрока, которую нужно заменить, а второй параметр — это замена
spacify	Вставляет пробел между каждым символом в строке
strip	Заменяет все повторяющиеся пробелы, символы новой строки и табуляции на один пробел или заданную строку. Строка задается с помощью первого параметра. Примеры: <code>{ \$articleTitle strip }</code> <code>{ \$articleTitle strip: "&nbsp;" }</code>
strip_tags	Удаляет все HTML-теги из строки
truncate	Обрезает строку. Первый символ указывает, сколько символов оставить в результирующей строке (по умолчанию 80). Второй символ — это текст, который будет добавлен к строке, если она была урезана. Третий параметр (может принимать true или false) определяет, как нужно обрезать строку — по границе слова (false) или по символу (true): <code>{ \$articleTitle truncate }</code> <code>{ \$articleTitle truncate: 30 }</code> <code>{ \$articleTitle truncate: 30: "..."</code> <code>{ \$articleTitle truncate: 30: "...", false }</code>
upper	Переводит символы в верхний регистр
wordwrap	Разбивает текст на строки. Первый параметр задает, сколько символов должно быть в строке (по умолчанию — 80), второй определяет символ (строку), который будет добавляться для разделения текста на строки (по умолчанию \n), а последний позволяет уточнить, нужно ли переносить по словам (false) или по строкам (true)

Вы можете применять любое количество модификаторов к переменной, разделяя их вертикальной чертой, например:

```
{ $articleTitle | upper | spacify }
```

Если переменная `$articleTitle` содержала текст "о вреде курения", то после применения этих двух модификаторов будет выведен текст "О В Р Е Д Е К У Р Е Н И Я".

23.4.6. Стандартные (встроенные) функции Smarty

С одной стандартной функцией, `{config_load}`, мы уже знакомы. Кроме нее имеются следующие стандартные функции:

- `{capture}`
- `{foreach}`, `{foreachelse}`
- `{if}`, `{elseif}`, `{else}`
- `{include}`
- `{include_php}`
- `{insert}`
- `{ldelim}`, `{rdelim}`
- `{literal}`
- `{php}`
- `{section}`, `{sectionelse}`
- `{strip}`

Обратите внимание: `{if}`, `{elseif}` и `{else}` называются не операторами, а функциями.

В Smarty вы можете создавать свои пользовательские функции, однако нельзя модифицировать уже имеющиеся или создавать функции с такими же именами. Все стандартные функции мы рассматривать не будем. Рассмотрим лишь те, которые действительно полезны и часто применяются.

Функции `{include}` и `{insert}`

Начнем с функции `{include}`, позволяющей подключить другой шаблон:

```
{include file='template.tpl'}
```

Кроме параметра `file` можно указать еще и параметр `assign`:

```
{include file='template.tpl' assign='val'}
```

В этом случае вывод шаблона, указанного параметром `file`, будет присвоен переменной `val`, которую можно вывести так:

```
{$val}
```

Функция `{include_php}` ранее применялась для подключения PHP-файлов. Сейчас использовать эту функцию настоятельно не рекомендуется. Тем не менее она доступна для совместимости.

Функция `{insert}` похожа на `{include}`, но включаемый шаблон не кэшируется.

Функция `{foreach}`

Функция `{foreach}` служит для организации циклов. Цикл `{foreach}` реализует проход по массиву (списку или ассоциативному). Кроме `{foreach}` имеется еще цикл

{section}, но мы его рассматривать не будем. Во-первых, циклы в шаблонах — редкость. Да, иногда нужно вывести массив, но для этого вполне хватит возможностей {foreach}. Во-вторых, синтаксис {section} сложнее, чем {foreach}, поэтому последний удобнее.

Каждый тег {foreach} должен закрываться тегом {/foreach}. Функции {foreach} можно передать четыре параметра, два из них являются обязательными: from и item. Первый — это массив значений, второй — переменная, в которую будет записываться значение текущего элемента массива. Функция {foreachelse} выполняется, если параметр from не содержит значений.

Кроме from и item существуют параметры key и name. Параметр key задает имя переменной, которая будет выступать в качестве ключа, а name — название цикла foreach для доступа к его свойствам. Вряд ли вы будете этим заниматься, поэтому на практике достаточно трех параметров (from, item и key).

Рассмотрим небольшой пример использования циклов в шаблонах. Предположим, что мы в сценарии определили массив и передали его в шаблон:

```
$arr = array( 1,2,3, 4, 5 );
$smarty->assign('arr', $arr);
```

В шаблоне мы можем вывести значения этого массива так:

```
{foreach from=$arr item=it}
    {$it}<br />
{/foreach}
```

В результате обработки шаблона будут выведены цифры от 1 до 5 по одной в каждой строке:

```
1
2
3
4
5
```

Теперь поговорим об ассоциативных массивах. Определим массив в PHP-сценарии так:

```
$assoc-list = array(
    array('name' => 'Denis', surname = 'Kolisnichenko'),
    array('name' => 'Ivan', surname = 'Petrov')
)
$smarty->assign('list', $assoc-list);
```

В шаблоне Smarty вывести массив можно так:

```
{foreach from=$list item=it }
    <p>
    {foreach from=$it key=key item=item }
        {$key}: {$item} &nbsp;
```



```

    {$key}: {$item}<br>
  {/foreach}
{/foreach}

```

Обратите внимание, как обрабатывается наш массив. Ведь переданный в шаблон список `list` является списком ассоциативных массивов, т. е. каждый элемент данного списка тоже будет ассоциативным массивом. Сначала мы получаем каждый элемент массива `list` как переменную `$it`, а затем обрабатываем эту переменную, как отдельный массив.

Функции `{if}`, `{elseif}`, `{else}`

Настало время рассмотреть функции `{if}`, `{elseif}` и `{else}` — наиболее употребительные функции Smarty. Они позволяют организовать обработку условий так же гибко, как и аналогичная конструкция в PHP. Функции `{if}`, `{elseif}` и `{else}` для описания условий используют квалификаторы (`<`, `>` и т. д.), которые должны быть отделены от остального содержимого пробелами. Квалификаторы аналогичны существующим в PHP, но допустим также и альтернативный синтаксис (табл. 23.2).

Таблица 23.2. Квалификаторы условий Smarty

Квалификатор	Альтернативный синтаксис	Эквивалент PHP	Пример	Описание
<code>==</code>	<code>eq</code>	<code>==</code>	<code>\$a eq \$b</code>	равно
<code>!=</code>	<code>ne</code> , <code>neq</code>	<code>!=</code>	<code>\$a neq \$b</code>	не равно
<code>></code>	<code>gt</code>	<code>></code>	<code>\$a gt \$b</code>	больше
<code><</code>	<code>lt</code>	<code><</code>	<code>\$a lt \$b</code>	меньше
<code>>=</code>	<code>gte</code> , <code>ge</code>	<code>>=</code>	<code>\$a ge \$b</code>	больше или равно
<code><=</code>	<code>lte</code> , <code>le</code>	<code><=</code>	<code>\$a le \$b</code>	меньше или равно
<code>===</code>		<code>===</code>	<code>\$a === 0</code>	проверка идентичности
<code>!</code>	<code>not</code>	<code>!</code>	<code>not \$a</code>	отрицание
<code>%</code>	<code>mod</code>	<code>%</code>	<code>\$a mod \$b</code>	остаток от деления
<code>is [not] div by</code>		<code>\$a % \$b == 0</code>	<code>\$a is not div by 4</code>	возможно деление без остатка
<code>is [not] even</code>		<code>\$a % 2 == 0</code>	<code>\$a is not even</code>	[не]четно
<code>is [not] even by</code>		<code>(\$a / \$b) % 2 == 0</code>	<code>\$a is not even by \$b</code>	[не]четно значению
<code>is [not] odd</code>		<code>\$a % 2 != 0</code>	<code>\$a is not odd</code>	[не]нечетно
<code>is [not] odd by</code>		<code>(\$a / \$b) % 2 != 0</code>	<code>\$a is not odd by \$b</code>	[не]нечетно значению

Рассмотрим пример:

```
{* не забываем о пробелах ДО и ПОСЛЕ квалификаторов *}
{if $user == "admin"}
Добро пожаловать, дорогой администратор!
{elseif $user == "guest"}
    Привет!
{else}
Привет, {$user}!
{/if}
```

В функции {if} можно вызывать PHP-функции, например:

```
{if isset($var) }
    {$var}
{/if}
```

Функция {capture}

Функция {capture} служит для чтения в одну переменную содержимого шаблона для последующей обработки этого содержимого:

```
{capture name=template}
    {include file='template.tpl'}
{/capture}
```

Далее обратиться к содержимому шаблона можно через переменную \$smarty.capture.template:

```
{{$smarty.capture.template} }
```

Функция {php}

Иногда мне кажется, что разработчики Smarty перегибают палку. Сейчас как раз такой момент. Мы создавали шаблонизатор, учились его использовать и все ради того, чтобы вынести в шаблон весь HTML-код из нашего сценария. А разработчики Smarty возьми да и придумай функцию {php}, которая предназначена... для вставки PHP-кода в шаблон. Получается, что ушли от HTML-кода в сценарии, а получили PHP-код в шаблоне. Вы можете вызвать эту функцию так:

```
{php}
PHP-код
{/php}
```

Функция {strip}

А вот эта функция действительно нужна дизайнерам. Сам несколько раз сталкивался с тем, что лишние пробелы и переносы влияют на конечный результат — дизайн отображается не так, как задумано. И функция {strip} удаляет повторяющиеся пробелы и символы переносов. Например, следующий код, описанный внутри {strip}, будет выведен как одна строка:


```
{strip}
<html>
<head>
    <title>Моя страничка</title>
</head>
{/strip}
```

Оставшиеся функции мы рассматривать не будем — уж очень редко они используются.

23.4.7. Пользовательские функции Smarty

В предыдущем разделе мы рассмотрели встроенные функции Smarty. На этом наше знакомство с функциями не заканчивается. Теперь мы рассмотрим пользовательские функции Smarty, которые еще называют плагинами. В Интернете вы можете найти дополнительные плагины для Smarty. Плагины — это специальным образом оформленные PHP-сценарии, находящиеся в каталоге `Smarty-x-y-z/libs/plugins`. Рассматривать создание собственных плагинов мы не будем, поскольку я считаю настоящим кощунством программирование на "языке шаблонизатора".

Итак, рассмотрим самые полезные пользовательские функции.

Функция `{assign}`

Функция `{assign}` нам уже знакома. Она позволяет создать Smarty-переменную и присвоить ей значение. Напомню ее синтаксис:

```
{assign var="имя_переменной" value="значение"}
{$имя_переменной}
```

Функция `{cycle}`

Предположим, что нам требуется вывести таблицу, причем один ряд нужно вывести с одним цветом фона, а другой — с другим. Реализовать такое переключение можно с помощью функции `cycle`:

```
{section name=rows loop=$data}
<tr bgcolor="{cycle values="red, blue"}">
    <td>{$data[rows]}</td>
</tr>
{/section}
```

Функцию `{cycle}` удобно использовать в паре с функцией `{section}`, позволяющей организовать цикл. Значения, по которым производится цикл, задаются с помощью параметра `values`.

В данном случае чередуются красный и синий цвета. Варианты значений разделяются запятой. Если вас этот разделитель не устраивает, с помощью параметра `delimiter` вы можете задать другой:

```
<tr bgcolor="{cycle delimiter=";" values="red; blue"}">
```


Функция {fetch}

Функция {fetch} позволяет получить любой файл и включить его в шаблон. Функции можно передать два параметра: `file` — задает имя файла и `assign` — задает имя переменной, которой будет присвоен вывод.

Примеры:

```
{fetch file="/var/www/1.html"}
{fetch file="http://server.ru/some_file.css"}
{fetch file="ftp://user:password@ftp.server.ru/public/file.txt"}
```

Функции {html_checkboxes} и {html_radios}

Функция {html_checkboxes} позволяет вывести группу флажков по указанной информации. Напомню синтаксис флажка:

```
<input type=checkbox name=check value=значение [checked]>
```

Подпись для флажка можно задать с помощью тега LABEL:

```
<label>
<input type=checkbox name=имя_группы value=значение>Подпись
</label>
```

Функции {html_checkbox} можно передать следующие параметры:

- ❑ `name` — имя группы флажков;
- ❑ `values` — массив значений флажков;
- ❑ `output` — массив названий (подписей);
- ❑ `separator` — строка, разделяющая каждый флажок;
- ❑ `selected` — выбранный флажок (флажки);
- ❑ `labels` — если `true`, то к выводу будут добавлены теги LABEL (по умолчанию `true`).

Рассмотрим пример вызова функции {html_checkbox}. Пусть у нас есть PHP-код:

```
$smarty->assign('nums', array(1, 2, 3));
$smarty->assign('labels',
    array('Вариант 1', 'Вариант 2', 'Вариант 3'));
$smarty->assign('selected', 1);
```

Функцию {html_checkboxes} вызовем так:

```
{html_checkboxes name="check" values=$nums selected=$selected output=$labels
separator="<br>"}
```

В результате получим следующий HTML-код:

```
<label><input type="checkbox" name="check" value="1" checked="checked"
/>Вариант 1</label><br>
<label><input type="checkbox" name="check" value="2" />Вариант 2</label><br>
<label><input type="checkbox" name="check" value="3" />Вариант 3</label><br>
```


Кроме функции `{html_checkboxes}` можно использовать функцию `{html_radios}`, которая выводит группу зависимых переключателей. Параметры те же, но параметр `selected` может быть только строкой, а не массивом, поскольку активным является лишь один зависимый переключатель.

Функция `{html_image}`

Функция `{html_image}` генерирует HTML-код для картинки. Вот пример ее использования:

```
{html_image file="photo.jpg"}
```

На выходе получим (при условии, что размер фото составляет 640×480 пикселей):

```

```

Функция `{html_select_date}`

Функция `{html_select_date}` очень удобна — она генерирует форму для выбора даты. У этой функции довольно много параметров, влияющих на формирование формы, поэтому они сведены в табл. 23.3.

Таблица 23.3. Некоторые параметры функции `{html_select_date}`

Параметр	Тип	По умолчанию	Описание
prefix	string	Date_	Префикс названий переменных для передачи даты (переменные года, месяца, дня)
time	timestamp	текущее время	Время
start_year	string	текущий год	Начальный год. Можно указать относительно текущего года
end_year	string	аналогично start_year	Конечный год. Можно указать относительно текущего года
display_days	boolean	true	Если true, то будет выведен список дней
display_months	boolean	true	Если true, то будет выведен список месяцев
display_years	boolean	true	Если true, то будет выведен список лет
month_format	string	%B	Задаёт формат названий месяцев (см. модификаторы функции <code>date()</code>)
day_format	string	%02d	Задаёт формат названий дней (см. модификаторы функции <code>date()</code>)
day_value_format	string	%d	Задаёт формат отображения дней (см. модификаторы функции <code>date()</code>)
year_as_text	boolean	false	Если true, то значение года будет выведено текстом
reverse_years	boolean	false	Если true, то годы будут выведены в обратном порядке

Таблица 23.3 (окончание)

Параметр	Тип	По умолчанию	Описание
field_array	string	null	Название переменной (например, var), которая будет содержать выбранные значения в виде массива: var [Day], var [Year], var [Month]
day_size	string	null	Атрибут size тега SELECT для дней
month_size	string	null	Атрибут size тега SELECT для месяцев
year_size	string	null	Атрибут size тега SELECT для лет
field_order	string	MDY	Порядок следования тегов SELECT (МДГ)
field_separator	string	\n	Разделитель полей (тегов SELECT)
month_value_format	string	%m	Формат значения месяца. По умолчанию — %m (номер месяца)

Пример вызова функции (результат показан на рис. 23.2):

```
{html_select_date}
```

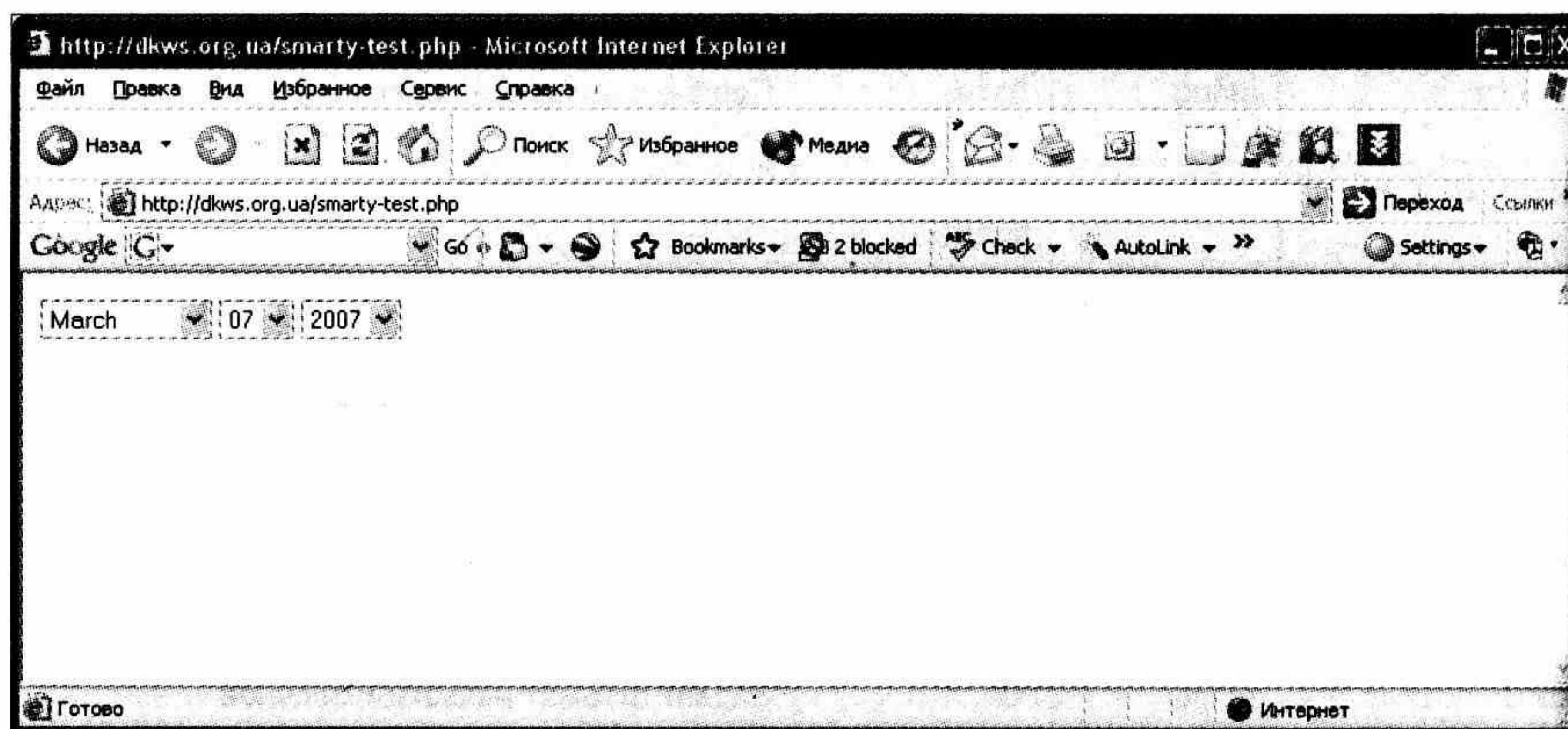


Рис. 23.2. Результат работы функции {html_select_date}

Функция {html_select_time}

Функция {html_select_time} подобна предыдущей, только она отображает форму для выбора времени, а не даты. Обычно нужно установить только один ее параметр — use_24_hours для отображения времени в 24-часовом формате:

```
{html_select_time use_24_hours=true}
```


Некоторые параметры этой функции приведены в табл. 23.4.

Таблица 23.4. Полезные параметры функции `{html_select_time}`

Параметр	Тип	По умолчанию	Описание
prefix	string	Date_	Префикс названий переменных для передачи даты (переменные года, месяца, дня)
time	timestamp	текущее время в формате	Время
display_hours	boolean	true	Если true, то будут выведены часы
display_minutes	boolean	true	Если true, то будут выведены минуты
display_seconds	boolean	true	Если true, то будут выведены секунды
display_meridian	boolean	true	Если true, то будут показаны символы am/pm (для 12-часового формата)
use_24_hours	boolean	true	Использовать 24-часовой формат времени
minute_interval	integer	1	Интервал минут
second_interval	integer	1	Интервал секунд

Функция `{html_table}`

Функция `{html_table}` значительно упрощает вывод информации в виде таблицы. Все, что вам потребуется, — это указать массив данных и размерность таблицы (обычно только число столбцов). Параметр `loop` задает массив данных, который нужно вывести в виде таблицы. Параметры `cols` и `rows` позволяют указать соответственно число колонок и строк в таблице. Обычно надо указать только параметр `cols`, который по умолчанию равен 3. Параметр `inner` задает, как будет выводиться массив — по рядам или по столбцам, по умолчанию `inner="cols"`.

Следующие параметры позволяют задать оформление таблицы:

- ▣ `table_attr` — задает дополнительные атрибуты таблицы (по умолчанию `"border=1"`);
- ▣ `tr_attr` — задает дополнительные атрибуты строки (тег `TR`);
- ▣ `td_attr` — задает дополнительные атрибуты колонки (тег `TD`).

Примеры вызова функции:

```
{html_table loop=$data}
```

```
{html_table loop=$data cols=5 table_attr='border="0"')}
```

На этом мы заканчиваем обзор дизайнерской части Smarty. Конечно, мы рассмотрели не все функции. Например, мы не упомянули функцию `math`, позволяющую

выполнять произвольные расчеты в шаблоне, но даже разработчики Smarty не рекомендуют ее использовать, все расчеты делать в сценарии, а в шаблон выводить только результат. Также не рассмотрели мы функцию `popup`, позволяющую создать всплывающее окно. А зачем? Ведь есть JavaScript. Тем более, что эта функция сама формирует JavaScript-код, а потом выводит его. То же самое можно сказать и о других функциях. Нет, они не бесполезны, просто могут понадобиться очень редко.

23.5. Smarty для программиста

В этом разделе мы посмотрим на шаблонизатор Smarty глазами программиста, и именно — рассмотрим специальные переменные и полезные методы класса Smarty.

23.5.1. Специальные переменные

Обращение к переменным Smarty осуществляется через объект Smarty, т. е. эти переменные являются полями класса. В табл. 23.5 приведен список полей класса Smarty.

Таблица 23.5. Список полей класса

Переменная	Описание
<code>template_dir</code>	Задаёт имя каталога для шаблонов. По умолчанию используется значение <code>./templates</code> , т. е. вы вообще можете не указывать значение этого поля при создании сценария, если ваши шаблоны находятся в каталоге <code>templates</code>
<code>compile_dir</code>	Задаёт имя каталога для откомпилированных шаблонов. По умолчанию <code>./templates_c</code>
<code>config_dir</code>	Позволяет указать имя каталога, содержащего конфигурационные файлы шаблонов. По умолчанию <code>./configs</code>
<code>plugins_dir</code>	Позволяет указать каталог с плагинами. По умолчанию <code>SMARTY_DIR/plugins</code>
<code>compile_check</code>	Если <code>true</code> (по умолчанию), то при каждом запуске шаблона Smarty будет проверять, изменился он или нет; если изменился, то Smarty перекомпилирует шаблон. Не рекомендуется присваивать этой переменной <code>false</code> , поскольку если вы сделаете это, то не увидите изменений в шаблоне, пока (при условии, что вы его изменяли) не установите эту переменную в <code>true</code>
<code>force_compile</code>	Если <code>true</code> (по умолчанию <code>false</code>), то шаблоны будут перекомпилироваться при каждом вызове, независимо от того, изменялись они или нет
<code>caching</code>	Если данный параметр установлен в 1, тогда Smarty будет кэшировать вывод шаблонов. По умолчанию данный параметр установлен в 0, т. е. кэширование выключено. Рекомендуется включить кэширование, если ваши шаблоны генерируют большие объёмы PHP-кода, в результате вы получите ощутимый прирост производительности
<code>cache_dir</code>	Задаёт каталог для кэша. По умолчанию <code>./cache</code>

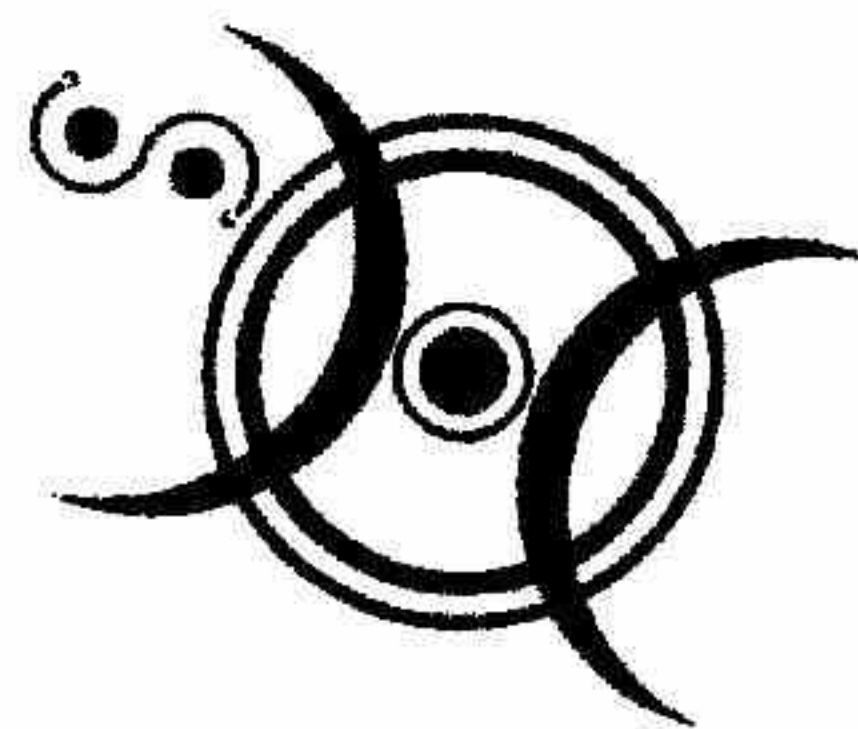
Таблица 23.5 (окончание)

Переменная	Описание
cache_lifetime	Время жизни кэша в секундах. Для использования этого параметра нужно установить caching в 1. Если установить для cache_lifetime значение -1, то время жизни кэша будет неограниченное
config_overwrite	Если true, то переменные из конфигурационного файла будут перекрывать все остальные. По умолчанию false
security	Если установлен в true, позволяет предотвратить взлом вашего сценария через шаблоны. Рекомендуется устанавливать в true, если шаблоны редактируют не совсем надежные люди. Параметр запрещает выполнение PHP-кода с помощью функции {php}; в функции {if} запрещает использование PHP-функций, кроме указанных в \$security_settings; шаблоны могут быть подключены только из каталога, который указан в \$secure_dir; запрещено использовать PHP-функции в виде модификаторов, кроме тех, которые указаны в \$security_settings
secure_dir	Массив всех каталогов, которые считаются безопасными
security_settings	Задаёт параметры безопасности. Массив, содержащий следующие элементы: PHP_HANDLING — если true, то параметр \$php_handling, который определяет, как будет выполняться PHP-код, не проверяется на безопасность; IF_FUNCS — массив имен PHP-функций, которые могут быть использованы в условиях IF; INCLUDE_ANY — если true, то любой шаблон может быть подключен независимо от списка secure_dir; PHP_TAGS — если true, то теги {php} разрешены в шаблонах; MODIFIER_FUNCS — массив имен PHP-функций, которые можно использовать в качестве модификаторов; ALLOW_CONSTANTS — если true, то допускаются константы вида {\$smarty.const.name}. Из соображений безопасности установлен в false: <pre>\$smarty->security_settings [PHP_HANDLING]=false; \$smarty->security_settings [IF_FUNCS]=array('isset');</pre>
php_handling	Задаёт, как будет обрабатываться PHP-код в тегах {php}. Значения: SMARTY_PHP_ALLOW — PHP-код разрешен; SMARTY_PHP_REMOVE — удалить теги {php} из шаблона; SMARTY_PHP_PASSTHRU — показывает теги без обработки
error_reporting	Если не 0, тогда будет включен режим отладки

23.5.2. Полезные методы класса *Smarty*

Из всех методов *Smarty* мы пока знаем лишь два: `assign` и `display`. Первый позволяет присвоить значение переменной шаблона, а второй — отобразить шаблон. По большому счету, вам какие-либо другие методы и не потребуются. Разве что полезным будет метод `template_exists`, возвращающий `true`, если указанный в качестве параметра шаблон существует. С остальными методами вы можете при необходимости ознакомиться в руководстве по использованию *Smarty* на сайте <http://smarty.php.net/>.

ГЛАВА 24



Объектно-ориентированное программирование

24.1. Основы ООП

Четкого определения объектно-ориентированного программирования (далее — ООП) нет даже в Википедии. Кто-то говорит, что это философия программирования. Можно соглашаться с этим, можно не соглашаться. А вообще, ООП — это способ разработки программ. Может, вы помните из истории вычислительной техники, что первые программы вводились на перфокартах. Так вот, это тоже способ написания программ. Затем перфокарты канули в прошлое, процесс обучения программированию стал проще, поскольку уже не нужно было привязываться к аппаратным средствам, а сам процесс разработки программ стал существенно быстрее.

Но очень скоро программы на языках высокого уровня разрослись так, что стали совсем нечитабельными, поскольку отсутствовала четко выделенная структура кода. Нужно было что-то делать. Появилось так называемое процедурное программирование — это тоже способ написания программ, когда отдельные части программы объединяются в подпрограммы: процедуры и функции. В чем преимущество процедурного программирования? Предположим, что нам нужно написать программу, которая начисляет зарплату (и все отчисления) для трех сотрудников маленькой фирмы. Если бы не было процедур и функций, то код расчета заработной платы пришлось бы повторять три раза — для каждого сотрудника. В этом тривиальном случае, если быть внимательным, ошибку допустить практически невозможно, но ведь на практике задачи ставятся намного сложнее, а чем сложнее программа, тем больше вероятность логической ошибки.

В случае с процедурным программированием расчет зарплаты можно организовать в виде функции (или процедуры) и вызывать при необходимости. Процедурное программирование существенно облегчило жизнь программистам, но опять-таки только на определенное время. Со временем появилась та же проблема — длина исходного кода. Новым способом стало объектно-ориентированное программирование (ООП).

Базовым понятием ООП является класс. Класс объединяет в себе код (методы класса) и данные (свойства класса). Класс — это как бы тип переменной. Переменная типа класса называется экземпляром класса (или проще — объектом). И свойства

класса, и методы класса называются членами класса. Механизм объединения в единое целое данных и методов для их обработки называется инкапсуляцией.

В ОПП кроме инкапсуляции часто используется полиморфизм. Возможно, вам знакомы функции `abs()`, `fabs()` и `labs()`. Все эти три функции — различные варианты функции `abs`, просто предназначены для работы с разными типами данных. Алгоритм функций одинаковый, просто различны типы входящих и выходящих данных. ООП позволяет создать три различных варианта функции, но объединить их под одним именем. В результате у нас будет одна функция `abs()`, которую можно использовать для всех трех типов данных. Полиморфизм значительно упрощает написание сложных программ.

В ООП объект может наследовать свойства другого объекта. Не нужно думать, что наследование — это копирование объекта: при копировании создается точная копия исходного объекта, а при наследовании один объект дополняется членами другого объекта.

Поддержка ООП появилась в PHP 4, хотя весьма ограниченная. И только PHP 5 обладает уже полноценной поддержкой ООП. А в последних версиях, 5.4/5.5, поддержка ООП стала еще лучше.

24.2. Классы и объекты

Итак, класс — это тип переменной, а экземпляр класса (объект) — переменная этого типа. Создать класс просто:

```
class имя_класса {  
  
    // члены класса  
  
}
```

Напишем небольшой класс `Message`, который будет использоваться для отправки сообщений:

```
<?  
  
class Message {  
  
    // свойства класса  
    var $text;  
    var $subject;  
    var $to;  
  
    // методы класса  
    function load_text($filename) {  
  
        $this->text = join('',file($filename));  
  
    }
```



```
function send() {  
    mail($this->$to, $this->subject, $this->text);  
}  
  
}
```

Наш класс довольно прост, но вполне работоспособен. Свойство `$text` — это текст сообщения, `$subject` — тема, `$to` — получатель. Вы можете присвоить текст явно, а можете загрузить его из файла с помощью метода `load_text()`. Для отправки используется метод `send()`. Рассмотрим пример кода, реализующего отправку сообщения с помощью созданного нами класса:

```
// создаем экземпляр класса  
$mes = new Message;  
  
$mes->to = 'user@server.ru';           // получатель  
$mes->subject = 'Hi!';                  // тема  
$mes->load_text('message.txt');         // загружаем текст  
// текст можно указать явно:  
// $mes->text = 'Привет! Как дела?';  
  
$mes->send();                           // отправляем сообщение
```

Полный пример кода приведен в листинге 24.1.

Листинг 24.1. Класс и пример кода (полный листинг)

```
<?php  
  
class Message {  
  
    // свойства класса  
    var $text;  
    var $subject;  
    var $to;  
  
    // методы класса  
    function load_text($filename) {  
        $this->text = join('', file($filename));  
    }  
  
    function send() {  
        mail($this->$to, $this->subject, $this->text);  
    }  
}  
  
$mes = new Message;  
  
$mes->to = 'user@mail.ru';  
$mes->subject = 'Hi!';
```



```
$mes->load_text('message.txt');  
$mes->send();  
  
?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *24-1.php*), представленный в каталоге *Glava_24* сопровождающего книгу электронного архива (см. приложение 4):

Обратите внимание на то, как осуществляется доступ к свойствам класса с помощью оператора `->`. Доступ к члену класса внутри класса осуществляется с помощью оператора `->` и указателя `$this`. Указатель `$this` служит для доступа к членам класса, т. е. может использоваться как для свойств, так и для методов класса.

В PHP 5.5 появилось ключевое слово `class`, позволяющее получить имя класса. Например:

```
// объявление пространства имен должно быть первым оператором  
// в программе (см. п. 24.5.10)  
namespace A;  
  
...  
class B {  
}  
  
echo B::class;           // выводит A\B
```

24.3. Конструкторы и деструкторы класса

Конструктор — это специальный метод, выполняющий инициализацию объекта. Он, например, устанавливает нужные соединения, открывает файлы, присваивает свойствам объекта начальные значения. Одним словом, подготавливает объект к нормальной работе. Имя метода конструктора совпадает с именем класса. Для нашего класса `Message` конструктор будет выглядеть так:

```
function Message() {  
    // имя пользователя по умолчанию  
    $this->to = 'user@mail.ru';  
    // тема по умолчанию  
    $this->subject = 'Привет!';  
    // текст по умолчанию  
    $this->text = 'Привет! Как дела?';  
}
```

Конструктор вызывается сразу при объявлении объекта класса:

```
$mes = new Message;
```

Поскольку конструктор установил свойства объекта по умолчанию, то для отправки письма пользователю по умолчанию (**user@mail.ru**) нужно только вызвать метод `send()`:

```
$mes->send();
```


В PHP 5 можно указывать единое имя для конструкторов — `__construct()`:

```
function __construct() {  
    // имя пользователя по умолчанию  
    $this->to = 'user@mail.ru';  
    // тема по умолчанию  
    $this->subject = 'Привет!';  
    // текст по умолчанию  
    $this->text = 'Привет! Как дела?';  
}
```

Кроме конструкторов в ООП существуют еще и деструкторы. Деструктор — это метод класса, который выполняет деинициализацию объекта, например освобождение используемых ресурсов, закрытие соединений и т. д. В PHP 4 деструкторы не поддерживаются, а все используемые объектом ресурсы освобождаются автоматически. С одной стороны, удобно, с другой — не очень правильно по отношению к ООП. В PHP 5 деструкторы поддерживаются. Деструктор класса называется `__destruct()`:

```
function __destruct() {  
    print "Destroying " . $this->name . "\n";  
}
```

Свойство `name` мы не объявляли, оно объявляется по умолчанию (автоматически) и должно содержать имя объекта, но это имя мы должны присвоить сами:

```
$mes = new Message;  
$mes->name = 'Message object';
```

24.4. Наследование классов. Полиморфизм

Как уже отмечалось, наследование — это операция расширения путем создания нового класса на базе существующего и добавления новых свойств и методов. Наследование определяется с помощью служебного слова `extends`, после которого следует имя родительского класса (того класса, который будет взят за основу при создании нового — дочернего класса):

```
class Имя_дочернего_класса extends Имя_родительского_класса {  
  
}
```

Рассмотрим небольшой пример наследования:

```
class ParentClass {  
    function F1() {  
        echo "Родительский класс";  
    }  
}
```



```
class ChildClass extends ParentClass {  
  
    function F1() {  
        echo "Дочерний класс";  
    }  
  
    function F2() {  
        echo "Дочерний класс. Функция F2";  
    }  
  
}  
  
$p = new ParentClass;  
$c = new ChildClass;  
  
$p->F1();           // выведет "Родительский класс"  
$c->F1();           // выведет "Дочерний класс"  
$c->F2();           // выведет "Дочерний класс. Функция F2"
```

Дочерний класс `ChildClass` расширяет родительский класс `ParentClass`, переопределяя метод `F1()` и добавляя новый метод `F2()`. Метода `F2()` в родительском классе не было, он появился только в дочернем классе `ChildClass`.

Обратите внимание на то, что дочерний класс переопределил метод `F1()` родительского класса. Это и есть пример полиморфизма, т. е. перезагрузки методов и обычных функций.

Основы ООП, можно сказать, мы уже рассмотрели. Да, не очень впечатляет, особенно по сравнению с C++. Но таковы возможности PHP 4. Поэтому теперь настало время поговорить о возможностях PHP 5 и самых новых версий — 5.4/5.5.

24.5. Новые возможности PHP 5 и 5.4/5.5

24.5.1. Область видимости членов класса

В PHP 5 вы можете определить область для переменных — членов класса. Вы можете использовать частные (`private`) и защищенные (`protected`) переменные. Защищенные свойства доступны только методам своего класса и производным (дочерним) классов. Вы не можете установить значение защищенного свойства ни посредственно. Частные (`private`) свойства доступны только методам своего класса. Вы не можете установить значения этих свойств ни непосредственно, ни через дочерний класс.

Вспомним наш класс `Message`. Объявим его свойства так (заодно можно установить значения по умолчанию):

```
private $to = "user@mail.ru";  
protected $subject = "Привет!";  
protected $text = "Как дела?";
```


После этого мы не сможем уже обратиться к данным свойствам так, как мы это делали раньше:

```
$mes->to = "user@mail.ru";
```

Для установки приватных (частных) и защищенных свойств нам придется писать собственные методы. Напишем метод `set_attr()`, который устанавливает все наши свойства:

```
function set_attr($to, $subj, $text) {  
  
    $this->to = $to;  
    $this->subject = $subject;  
    $this->text = $text;  
  
}
```

Свойства `$subject` и `$text` будут доступны дочерним классам, а вот свойство `$to` — частное, оно не будет доступно производным классам.

Аналогично вы можете указать область видимости и для методов класса. Модификаторы `private` и `protected` нужно указать перед служебным словом `function`:

```
private function send() { };
```

или

```
protected function send() { };
```

Если модификаторы `private` и `protected` не указаны, члены класса считаются публичными (`public`) — общедоступными.

24.5.2. Абстрактные классы и методы

PHP 5 поддерживает абстрактные классы и методы. Абстрактные методы — это всего лишь объявления методов, не содержащие реализации метода. Абстрактный класс — это класс, поддерживающий абстрактные методы. Если вы хотите использовать абстрактные методы, вы должны объявить абстрактный класс. В обычном классе вы не можете объявить абстрактные методы. Абстрактные методы и классы объявляются с помощью служебного слова `abstract`:

```
abstract class AbstractClass {  
    abstract public function test();  
}
```

Вы не можете создать экземпляр (объект) абстрактного класса. Можно создать обычный класс на базе абстрактного, объявив его дочерним, а потом уже создать объект класса. Но вы должны реализовать каждый метод абстрактного класса в дочернем классе.

```
class MyClass extends AbstractClass {  
    // реализация абстрактного метода test  
    public function test() {
```



```
        echo "проверка";  
    }  
}
```

```
$o = new MyClass;  
$o->test();
```

24.5.3. Служебное слово *final*

Служебное слово `final` используется для описания методов класса, которые не могут быть переопределены в классе-потомке (в дочернем классе):

```
class MyClass {  
    final function test() {  
        echo "проверка";  
    }  
}
```

Служебное слово `final` можно использовать и при описании класса. Тогда этот класс нельзя будет наследовать:

```
final class MyClass {  
    // определение класса  
}
```

Если вы попытаетесь наследовать класс `MyClass`, то получите ошибку времени выполнения.

24.5.4. Клонирование объектов

Мы уже говорили о наследовании классов. Теперь поговорим о клонировании объектов, а именно — о создании копии объекта. PHP 4 просто осуществлял побайтовое копирование объекта, создавая точные копии всех его свойств. Начиная с PHP 5, появилась возможность управления клонированием объекта.

Управляет клонированием объекта специальный метод — `__clone()`. Этот метод есть у каждого класса по умолчанию, он просто побайтно копирует все свойства и методы объекта. Клонирование осуществляется с помощью оператора `clone`:

```
$obj2 = clone $obj1;
```

В руководстве по PHP 5 упоминается, что можно клонировать объект вызовом метода `__clone()`:

```
$obj2 = obj1->__clone();
```

Однако последние версии PHP 5 завершают выполнение такого сценария с рекомендацией использовать оператор `clone`:

```
Fatal error: Cannot call __clone() method on objects — use 'clone $obj'  
instead in D:\PHP\class.php on line 23
```

Рассмотрим листинг 24.2, в котором приведен пример клонирования класса.

Листинг 24.2. Пример клонирования класса

```
<?php
class MyClass {
    public $id = 0;           // значение по умолчанию

    function MyClass() {
        // конструктор присваивает свойству id значение 10
        $this->id = 10;
    }
    function __clone() {
        // при клонировании будет скопировано имя объекта
        // и присвоено свойству id значение 0
        $this->name = $that->name;
        $this->id = 0;
    }
}

$obj = new MyClass();       // создаем класс

print $obj->id . "\n";      // выведет 10

$obj = clone $obj;         // клонируем класс

// выведет 0, потому что __clone() присвоит свойству id ноль
print $obj->id . "\n";
?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *24-2.php*), представленный в каталоге *Glava_24* сопровождающего книгу электронного архива (см. приложение 4).

24.5.5. Обработка исключительных ситуаций

В PHP 5 появилась возможность обработки исключительных ситуаций. Для обработки исключительной ситуации служит блок `try` (его называют блоком обработки исключительной ситуации) — точнее, блок `try` перехватывает исключение, а сама обработка осуществляется в блоке `catch`. Для порождения исключения используется оператор `throw`. Рассмотрим типичный пример блока `try-catch`:

```
<?php
try {
    // в блоке try производятся потенциально опасные действия,
    // которые могут породить исключение
} catch (Exception $e) {
```



```
// перехваченное исключение помещается в переменную $e, здесь
// осуществляется обработка исключения
// Exception — это тип исключения
}
?>
```

Породить исключение можно с помощью оператора `throw`:

```
throw new Exception(<сообщение>)
```

Впоследствии (при обработке исключительной ситуации) переданное сообщение можно получить с помощью метода `getMessage()` класса `Exception`:

```
echo $exception->getMessage();
```

Рассмотрим простой пример обработки исключения (листинг 24.3):

Листинг 24.3. Обработка исключительной ситуации

```
<?
$a = 10; $b = 0;

try {
    if ($b==0) throw new Exception('EDivideZero');
    $a = $a / $b;
}
catch (Exception $exception) {

    if ($exception->getMessage()=='EDivideZero')
        die('EDIVIDEZERO');

}
?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *24-3.php*), представленный в каталоге *Glava_24* сопровождающего книгу электронного архива (см. приложение 4).

Если делитель (`$b`) = 0, то мы генерируем исключение с сообщением **EDivideZero**. Затем мы анализируем (в блоке `catch`) полученное сообщение и выполняем какие-то действия. В нашем случае мы выводим сообщение **EDIVIDEZERO** и завершаем работу сценария (функция `die()`).

В PHP 5.5 появилось служебное слово `finally`. Блок `finally` указывается после блока `try/catch` и выполняется всегда, когда завершается блок `try/catch`, независимо от того, было ли порождено исключение или нет. Блок `finally` поможет программистам писать более гибкий код и избегать лишних конструкций, предназначенных для очистки памяти в момент обработки исключения.

Рассмотрим, как можно использовать служебное слово `finally`:

```
$db = mysqli_connect();
try {
```



```
// выполняем действия, которые могут выкинуть исключение
} finally {
    // закрываем соединение, независимо от того, было ли исключение
    // или нет
    mysqli_close($db);
}
```

Без `finally` код был бы менее изящным. Вам пришлось бы писать вот такие конструкции:

```
$db = mysqli_connect();
try {
    // выполняем действия, которые могут выкинуть исключение
} catch (Exception $e) {
    mysqli_close($db);
    throw $e;
}
mysqli_close($db);
```

Обратите внимание на более громоздкий блок `catch`, который будет выполнен в случае, если будет сгенерировано исключение, а также на дополнительный оператор вызова `mysqli_close()`. Ведь блок `catch` будет выполнен, только если было выкинуто исключение, а закрыть соединение с БД нужно в любом случае.

Все бы хорошо, но вам нужно знать некоторые особенности использования `finally`. Следует быть осторожным с использованием инструкции `return` в блоке `try` — ведь блок `finally` будет выполнен в любом случае:

```
function A() {
try {
    return 2;
} finally {
    echo "Привет!";
}
echo "Пока!";
}
```

При выполнении этого кода вы увидите слово **Привет!**, но не увидите **Пока!**, поскольку выполнение функции будет прервано инструкцией `return`.

Теперь рассмотрим листинг 24.4, в котором приведен пример вложенных блоков `try/catch/finally`.

Листинг 24.4. Вложенные блоки `try/catch/finally`

```
<?php

function test() {
    try {
        echo "1";
```



```
try {
    echo "2";
    throw new Exception("ex");
} catch (Exception $e) {
    echo "3";
} finally {
    echo "4";
    throw new Exception("ex");
}
} catch (Exception $e) {
    echo "5";
} finally {
    echo "6";
}
echo "7";
test();
?>
```

Этот сценарий выведет строку 1234567. Давайте разберемся, почему так. Сначала будет выполнен первый блок `try`. Нет никаких исключений и предпосылок для того, чтобы оператор `echo "1";` не был выполнен. Поэтому сценарий выведет 1.

Далее открывается вложенный блок `try`. Он сразу же выводит 2, поскольку ничего этому не мешает. После вывода 2 будет выкинуто исключение `"ex"`, а следующий блок `catch` как раз ждет это исключение. При обработке исключения блок `catch` выводит 3 и больше ничего не делает. Блок `finally`, принадлежащий вложенному блоку `try`, будет выполнен в любом случае. Он выведет 4 и породит следующее исключение, которое будет обработано внешним блоком `catch`, поскольку выполнение вложенного блока `try` уже завершено, — получается, что исключение возникло во внешнем блоке `try`. Его блок `catch` выведет 5, а блок `finally`, принадлежащий внешнему `try`, выведет 6. Наконец, последний оператор функции `test()` выведет 7, поэтому мы получим нашу строку: 1234567.

24.5.6. Константы-члены класса

В PHP 5 вы можете использовать константы-члены класса, причем обратиться к константе можно без объявления объекта класса. Доступ к константе можно получить так:

```
имя_класса::имя_константы
```

Вам не нужно указывать знак доллара (\$) ни для имени класса, ни для имени константы. Вот небольшой пример:

```
<?php
class Foo {
    const constant = "constant";
```



```
public $id = 0;
}

echo "Константа = " . Foo::constant . "\n";
?>
```

24.5.7. Статические члены класса

Статические члены класса можно вызывать за пределами контекста объекта, без объявления объекта с помощью оператора `::`, как мы поступали с константой. Для объявления статических членов класса используется служебное слово `static`:

```
class Stat {
    static $static_member = 5;
    public static function aStaticMethod() {
        // ...
    }
}

// обращаемся к статическим членам класса без объявления объекта
Stat::aStaticMethod();
echo Stat::$static_member;
```

24.5.8. Оператор *instanceof*

Оператор `instanceof` позволяет определить, является ли тот или иной объект экземпляром определенного класса. Использовать `instanceof` нужно так:

```
if (объект instanceof класс) ...
```

Например:

```
$mes = new Message;

if ($mes instanceof Message)
    echo "Объект mes является экземпляром класса Message";
```

24.5.9. Итераторы

Оператор цикла `foreach` можно использовать для перебора всех элементов ассоциативного массива примерно так:

```
foreach($assoc_array as $key => $value) { ... };
```

В PHP 5 этот оператор можно применить и для итерации по свойствам объекта. Мы можем перебрать и обработать все свойства объекта в цикле так:

```
foreach (объект as переменная_имя_свойства => переменная_значение_свойства) {};
```


Например:

```
$mes = new Message;
foreach ($mes as $prop_name => $prop_value)
    echo "$prop_name = $prop_value\n";
```

Как видите, возможности ООП в PHP 5.4/5.5 значительно шире, чем в PHP 4, и теперь можно с чистой совестью назвать PHP объектно-ориентированным языком программирования.

24.5.10. Пространства имен

Пространства имен — это способ избежать коллизий между функциями и классами без использования префиксов в именах методов и классов.

Рассмотрим пример использования пространства имен. Пространство имен объявляется инструкцией `namespace`:

```
namespace DenSpace;
```

Инструкция `namespace` должна быть первым оператором в вашей программе, то есть ее нужно вызывать сразу после `<?php`:

```
<?php
namespace DenSpace;
```

Затем можно объявить какой-нибудь класс. Я объявил класс `XMLWriter`, который уже существует в PHP, — налицо коллизия имен, PHP должен сообщить об этом и прервать выполнение сценария:

```
class XMLWriter
{
    // члены и метода класса
}
```

PHP 5 при объявлении этого класса сообщил:

Cannot redeclare class xmlwriter

В PHP 5.4/5.5 все проходит гладко, только при использовании нашего класса нужно указывать наше же пространство имен:

```
$xml = new DenSpace::XMLWriter();
```

24.5.11. Типажи (traits)

Основная новинка 5.4 в области ООП — это типажи (traits), или, как их еще называют, — примеси. Они служат заменой мультинаследования. Каждый типаж задается отдельным классом с помощью ключевого слова `trait`.

Внутри типажа вы можете определять методы, которые будут доступны в классе, к которому подключите типажи. Подключение типажей происходит с помощью ключевого слова `use`. Имена типажей задаются через запятую.

Если совпадают имена методов класса и типажей, то будет вызван метод класса, а не типажа, поэтому будьте внимательны, поскольку PHP даже не сообщит вам об этом — не будет ни ошибки, ни даже уведомления (notice).

Чтобы типаж всегда имел доступ к подключенному классу, нужно использовать конструкцию `parent`.

Теперь самое главное — рассмотрим пример:

```
// определяем класс A
class A {
    public function foo() {
        return 'foo';
    }
}

// определяем типаж B
trait B {
    public function bar() {
        // вызываем родительский метод foo, добавляем к его результату
        // строку 'bar'
        return parent::foo() . ' bar';
    }
}

// класс C расширяет A
class C extends A {
    // чтобы не описывать члены и методы, мы просто
    // сообщаем, что будем использовать типаж B
    use B;
}

// создаем объект класса C
$c = new C();
echo $c->foo(); // выведет "foo"
echo $c->bar(); // выведет "foo bar"
```

24.5.12. Вызов метода или свойства класса выражением

В PHP 5.4 появилось еще одно очень приятное нововведение. Теперь, чтобы вызвать свойство или метод класса, не нужно его запоминать в отдельную переменную.

Рассмотрим пример. Пусть у нас есть класс A, в нем — метод `foo_bar`:

```
class A {
    public static function foo_bar() {
        return '123';
    }
}
```


Вызвать этот метод можно так:

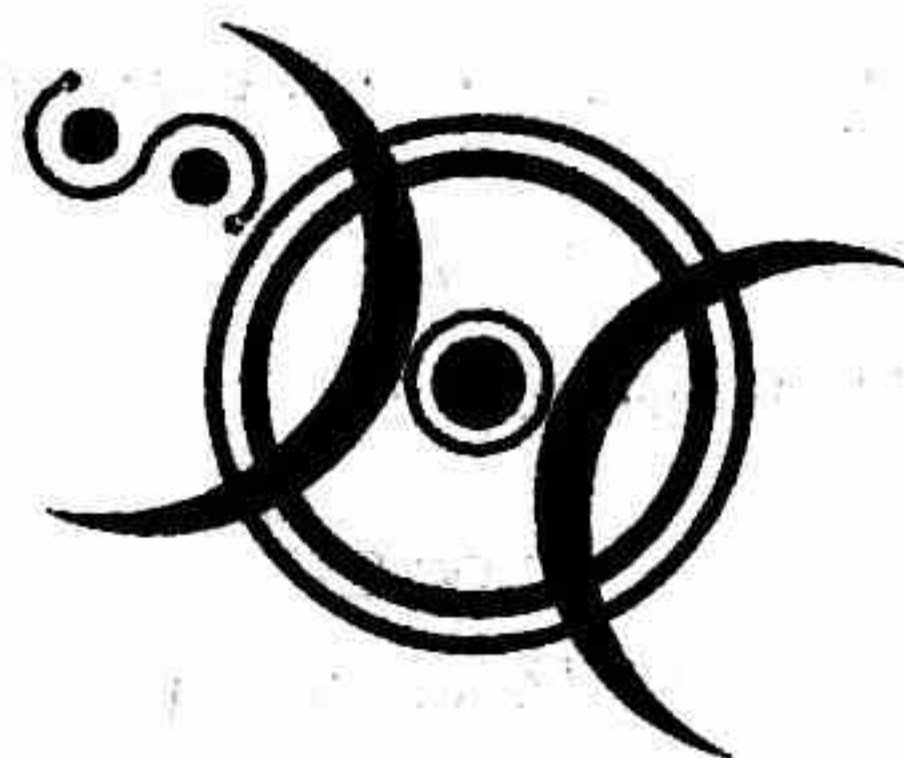
```
echo A::{"foo_var"}();
```

Или даже так:

```
$foo = 'foo';  
$bar = 'bar';  
echo A::{$foo . '_' . $bar}();
```

Не скажу, что это самое важное изменение в PHP 5.4, но я просто не мог не упомянуть о нем.

ГЛАВА 25



Механизм сессий

25.1. Для чего нужны сессии?

В этой главе мы поговорим о правильной аутентификации пользователя. Аутентификация нужна в любом серьезном проекте — будь то вход на форум или в панель администрирования сайта. Важно не только проверить имя пользователя и пароль, но проделать это правильно и после этого обеспечить дальнейшую корректную работу сценария.

Допустим, у нас есть сценарий `login.php` (листинг 25.1), выводящий форму регистрации и проверяющий правильность имени пользователя и пароля, если пользователь передал их с помощью формы.

Листинг 25.1. Сценарий аутентификации пользователя — `login.php`

```
<?php

if (!isset($login))
{
    // выводим форму входа на сайт
    echo join('', file('login-form.html'));
}
else
{
    // проверяем имя пользователя и пароль
    include "config.php";
    if (!mysql_connect($HOST, $USER, $PASS)) die('Error');
    mysql_select_db($DB);
    $r = mysql_query("select * from users where login=\"$login\"");
    $f = mysql_fetch_array($r);

    if ($f['password'] == md5($pass))
    {
        // выводим меню администратора
        echo join('', file('admin-menu.html'));
    }
}
```



```
else die('Неправильное имя пользователя или пароль');  
}  
  
?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *25-1.php*), представленный в каталоге *Glava_25* сопровождающего книгу электронного архива (см. приложение 4).

Основную задачу наш сценарий выполнил — проверил имя пользователя и пароль и вывел меню администратора (HTML-код меню вынесен из сценария и помещен в отдельный HTML-файл). Наверняка наш код будет содержать ссылки на другие PHP-сценарии — например, на сценарии управления пользователями, статьями и т. п. Возникает вопрос: как эти дополнительные сценарии узнают, что сценарий *login.php* проверил имя пользователя и пароль и разрешил доступ? Правильно. Никак. И что же, опять выводить форму для ввода имени пользователя и пароля?

Первое, что приходит в голову, — это использование Cookies. В случае корректной аутентификации мы устанавливаем Cookie-переменную *\$AUTH*. Если такая переменная установлена, значит, пользователь аутентифицирован.

Это простое решение, но не безопасное. Ведь злоумышленник может написать сценарий, который установит переменную с таким же именем. Тогда он сможет запустить любой сценарий, входящий в состав панели администрирования, даже без ввода имени пользователя и пароля.

Да и вообще применение Cookies не очень удобно → что делать, если Cookies отключены? Поэтому такое решение нельзя назвать универсальным. Но в PHP есть механизм сессий — более гибкий, чем Cookies. К тому же, если браузер не поддерживает Cookies (или они отключены), сценарии на основе сессий будут преспокойно работать.

25.2. Как работает механизм сессий?

В механизме сессий для идентификации пользователя имеется идентификатор сессии — *session identifier (SID)*. С его помощью PHP определяет, какой пользователь запустил сценарий. Идентификатор Cookies хранится в Cookies браузера, а все переменные сессии — на сервере. Получается, что SID — это номер хранилища (временного файла), в котором PHP хранит переменные сессии. Чувствуете разницу? Если в варианте с Cookies мы можем узнать имя переменной и установить ее в собственном браузере (для этого достаточно написать сценарий, вызывающий функцию *setCookie()*), то в случае с сессиями такой фокус не пройдет. Ведь переменные сессии хранятся на сервере, а не на нашем компьютере. Конечно, возможность перехвата сессии все же существует — например, когда пользователь по неосторожности оставит ссылку со своим SID где-то на форуме, но это равносильно тому, что он бы оставил там свое имя пользователя и пароль. Поэтому нужно соблюдать элементарную безопасность, господа!

Как уже было сказано, SID — это имя временного файла. Как и другие временные файлы, наши хранилища находятся на сервере в каталоге /tmp.

Инициализировать сессию можно с помощью функции `session_start()`:

```
session_start();
```

Для регистрации переменной в сессии предназначена функция `session_register()`:

```
bool session_register (mixed name [, mixed ...])
```

Эта функция принимает переменное число аргументов, т. е. позволяет за один вызов зарегистрировать несколько переменных в нашей сессии. Для аутентификации пользователя нам потребуется зарегистрировать три переменные: `$auth`, `$username`, `$password` (для чего они будут использоваться — поговорим позже).

```
session_register("a", "username", "password");
```

Имена переменных следует указывать без знака доллара. В сессии можно несколько раз регистрировать одну и ту же переменную, не боясь, что ее значение обнулится. После регистрации с переменной сессии можно работать как с обычной переменной, а ее последнее значение при завершении работы сценария будет записано в сессию.

Функция `session_is_registered()` позволяет проверить, зарегистрирована ли в сессии некоторая переменная:

```
bool session_is_registered(string name)
```

Если вы хотите удалить из сессии переменную, тогда укажите функцию `session_unregister()`:

```
session_unregister(string $name)
```

Удалить все переменные из сессии можно, вызвав функцию:

```
session_unset();
```

Вы можете присвоить собственное имя сессии с помощью функции `session_name()`, которую нужно вызвать до инициализации сессии, т. е. до обращения к функции

```
session_start();  
session_name('MySession');  
session_start();
```

Если имя сессии не задано (т. е. вы вообще не вызывали эту функцию), то по умолчанию в PHP будет задано имя `PHPSESSID`.

Не нужно путать SID и имя сессии. Имя сессии — это название группы сессий, к которой принадлежит тот или иной SID. Кстати, SID можно узнать с помощью функции `session_id()`:

```
string session_id([string sid])
```

Если параметр `sid` не задан, то функция возвращает SID, в противном случае она устанавливает заданное значение в качестве SID. Свой SID указывать не рекомендуется, поскольку вряд ли вы придумаете SID сложнее, чем это сделает за вас PHP.

25.3. Обход Cookies

Некоторые пользователи из соображений безопасности (правда, это больше иллюзия безопасности) отключают Cookies. Тогда к ссылкам желательно дописывать константу SID, содержащую идентификатор сессии:

```
<?php
session_start();
session_register("auth", "username", "password");

$login = "denis";

?>
<a href=admin.php?<?=SID?>>Ссылка</a>"
```

Но если вы забудете это сделать, ничего страшного — PHP выполнит это за вас. Другими словами, если Cookies выключены, все будет работать, причем вам для этого ничего не придется делать.

25.4. Сценарий аутентификации

Мы вплотную подошли к реализации нашего сценария аутентификации. Еще раз обратимся к листингу 25.1. Сейчас мы его дополним. Первоначально в случае правильности введенных имени пользователя и пароля просто выводилось меню администратора. Новый вариант будет создавать сессию с именем MyAdminSessionN, где N — число от 1 до 999. После этого сценарий регистрирует в сессии три переменные:

- ❑ \$auth — признак успешной аутентификации;
- ❑ \$username — имя пользователя;
- ❑ \$password — пароль, зашифрованный с помощью MD5.

Имя сессии будет передаваться нашим вспомогательным сценариям панели администрирования: users.php (управление пользователями) и articles.php (управление статьями). Эти сценарии станут подключаться к сессии, имя которой заранее неизвестно (ведь мы применим генератор случайных чисел для генерирования числа N), и использовать эти переменные. Если переменная \$auth отсутствует, тогда будет выведено сообщение о том, что сценарий нельзя запустить напрямую. Получается, что у нас есть двойная защита: с помощью имени сессии и с помощью SID. Если переменная \$auth установлена, тогда сценарий проведет повторную аутентификацию пользователя, основываясь на переменных \$username и \$password. Чтобы не хранить пароль пользователя в открытом виде, мы его шифруем с помощью алгоритма MD5, — просто при сравнении не станем вызывать MD5 для шифрования этого пароля, ведь мы будем сравнивать зашифрованное значение с зашифрованным. Сказанное иллюстрирует листинг 25.2.

Код файла 25.2. Полный вариант login.php

```
<?php

if (!isset($login))
{
    // выводим форму входа на сайт
    echo join('', file('login-form.html'));
}
else
{
    // проверяем имя пользователя и пароль
    include "config.php";
    if (!mysql_connect($HOST, $USER, $PASS)).die('Error');
    mysql_select_db($DB);

    $r = mysql_query("select * from users where login=\"$login\"");
    $f = mysql_fetch_array($r);

    if ($f['password'] == md5($pass))
    {
        // запускаем генератор случайных чисел
        mt_srand(time() + (double)microtime()*55);
        // генерируем $N
        $N = mt_rand(0, 999);
        $sname = "MyAdminSession$N";

        // устанавливаем имя сессии
        session_name($sname);
        // запускаем сессию
        session_start();
        // регистрируем переменные
        session_register("auth", "username", "password");

        // устанавливаем значения переменных:
        $auth = 1;
        $username = $login;
        $password = md5($pass);

        // выводим меню администратора
        echo "<p><a href=users.php?sname=$sname>Пользователи</a>
            <p><a href=articles.php?sname=$sname>Статьи</a>";
    }
    else die('Неправильное имя пользователя или пароль');
}

?>
```


ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *25-2.php*), представленный в каталоге *Glava_25* сопровождающего книгу электронного архива (см. приложение 4).

Для упрощения сценария я не стал исключать из него HTML-код, но на практике вам нужно будет подключить шаблонизатор и создать собственный шаблон, чтобы избавиться от HTML-кода в сценарии. Создание шаблонизатора рассмотрено в главе 22.

Как видите, каждому сценарию мы передаем только один параметр — имя нашей сессии. На рис. 25.1 показано, как будет выглядеть такая ссылка.

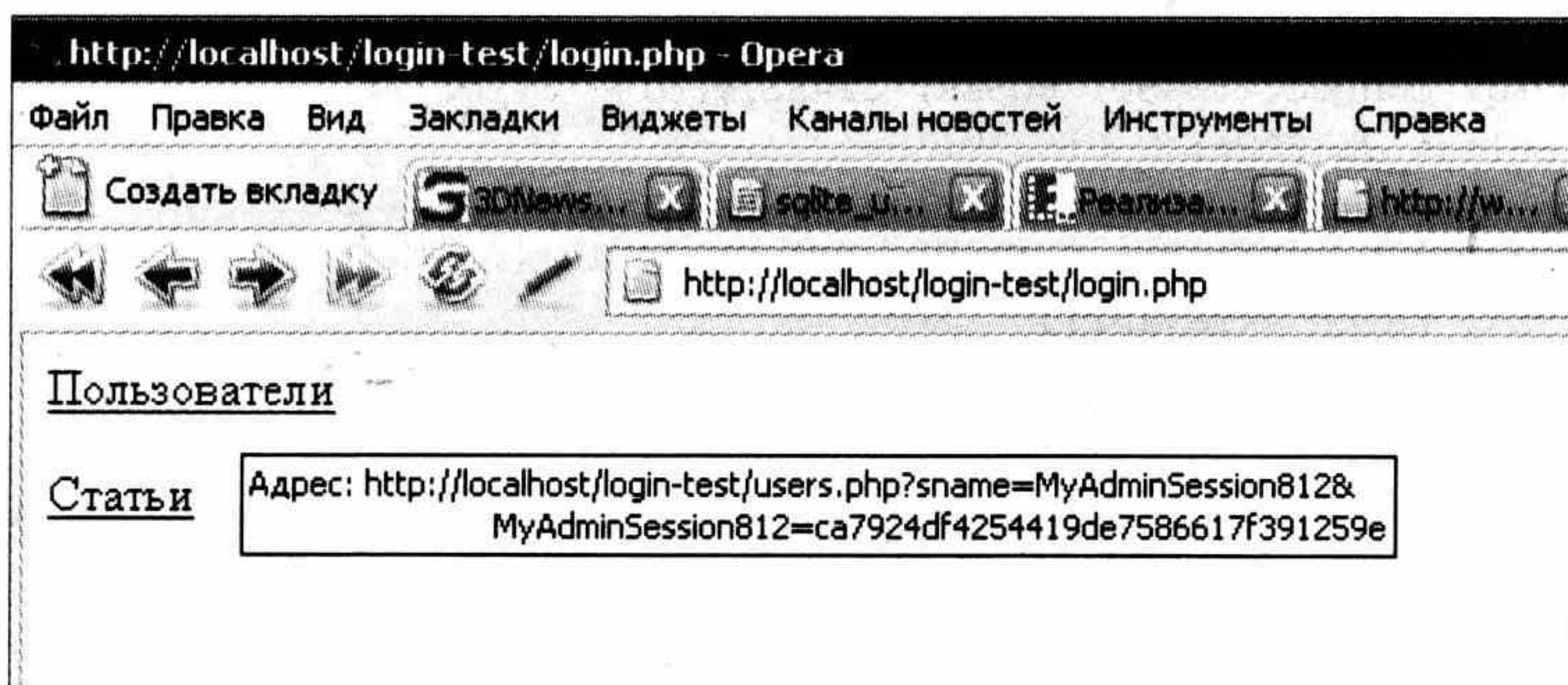


Рис. 25.1. Передача имени и идентификатора сессии другому сценарию

Обратите внимание, что PHP автоматически дописал SID к каждой ссылке, которую выводил наш сценарий, поэтому нам не нужно беспокоиться о работе сценария, если пользователь отключит Cookies.

Теперь приступим к созданию вспомогательных сценариев. Точнее, одного, т. к. второй можно написать "по образу и подобию", ведь схема аутентификации будет одинаковой во всех вспомогательных сценариях. В листинге 25.3 представлен сценарий *users.php*.

Листинг 25.3. Сценарий *users.php*

```
<?php

if (!isset($sname)) die('Этот сценарий не может быть запущен напрямую!');

// подключаемся к нашей сессии
session_name($sname);
// инициализируем сессию
session_start();
echo "Имя сессии: " . session_name();
echo "<p>Каталог для хранения сессии: " . session_save_path();
```



```
// выводим значения переменных сессии
echo "<p>AUTH: $auth
      <br>USERNAME: $username
      <br>PASSWORD: $password";

if ($auth != 1) die('Ошибка аутентификации');

// параметры для MySQL
include "config.php";

if (!mysql_connect($HOST, $USER, $PASS)) die('MySQL Error');
mysql_select_db($DB);
$r = mysql_query("select * from users where login=\"$username\"");
$f = mysql_fetch_assoc($r);

// поскольку $password уже зашифрована, то функцию md5()
// вызывать не нужно
if ($f[password] === $password)
{
    echo "<p>Приступаем к редактированию пользователей";
}
else die('Ошибка аутентификации');

?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *25-3.php*), представленный в каталоге *Glava_25* сопровождающего книгу электронного архива (см. приложение 4).

Как видите, сценарий сначала подключается к нашей сессии. Если параметр *\$sname* не установлен или будет задан неправильно, подключиться к сессии и прочитать ее переменные не удастся. Затем для контроля выводятся значения переменных сессии (рис. 25.2) и производится повторная аутентификация пользователя (за тем

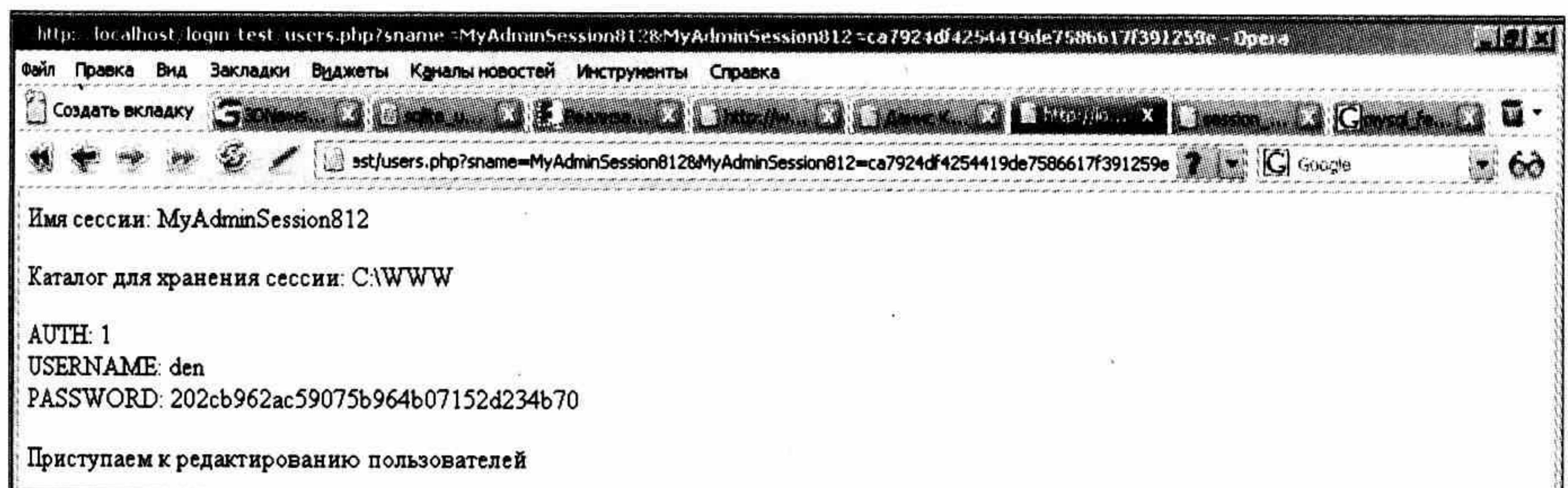


Рис. 25.2. Результат работы сценария *users.php*

лишь исключением, что форма для ввода имени пользователя и пароля не выводится). Если аутентификация успешна, то выводится строка: **Приступаем к редактированию пользователей**. Вы можете смело ее удалить и в ее блоке написать код, позволяющий редактировать учетные записи ваших пользователей.

Наверное, вас заинтересовало, почему мы не передавали пароль в открытом виде? Смотрите, в базе данных он хранится зашифрованным. Если бы мы передали его в открытом виде (без шифрования), то во временном каталоге его можно было бы увидеть, как говорится, невооруженным глазом. Файл сессии — это обычный текстовый файл, который можно просмотреть в любом редакторе, особо при этом не напрягаясь (рис. 25.3). Поэтому всю важную информацию, передаваемую через сессию, желательно шифровать.

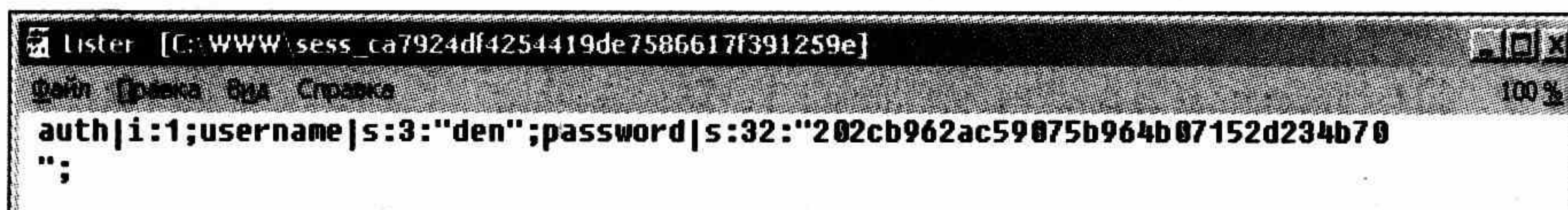
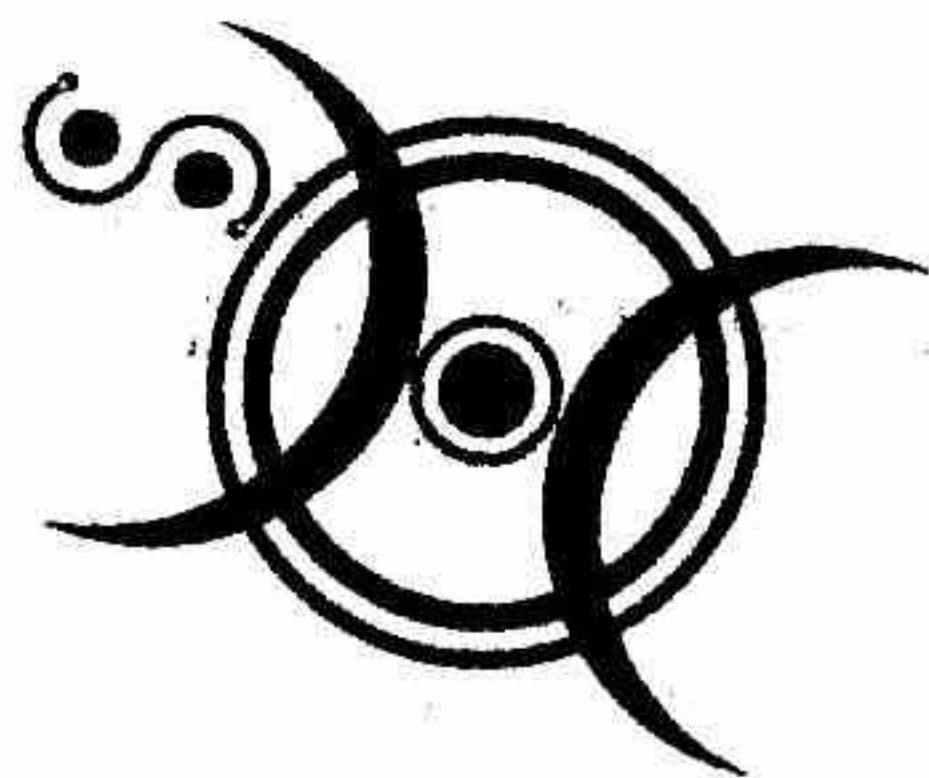


Рис. 25.3. Наш файл сессии

ГЛАВА 26



Введение в PEAR

26.1. Серьезные проекты и PEAR

PEAR (PHP Extension and Application Repositor) — это база расширений и приложений для PHP. Понять, что представляет собой PEAR, лучше на примере. Предположим, что у вас есть довольно серьезный проект, состоящий, скажем, из 100 PHP-сценариев (на самом деле это не очень много). Ваш проект основан на сервере баз данных MySQL, что не удивительно — MySQL является самым популярным сервером баз данных и часто используется в паре с PHP.

Вот пример фрагмента кода, который будет, вероятнее всего, присутствовать в каждом из ваших 100 сценариев:

```
include "config.php";

mysql_connect($server, $user, $pswd) or die(mysql_error());
mysql_select_db($db) or die(mysql_error());

// запрос
$q = "select * from main_table";
$r = mysql_query($q);

while ($row = mysql_fetch_array($r))
{
    echo $row[field_1];
}
```

Здесь мы видим: установку соединения, выбор базы данных, передачу запроса и обработку результата. Все вроде бы хорошо. До одного прекрасного момента — пока вы (или ваше руководство) не надумаете перейти на другой сервер баз данных — например, на Oracle. Тогда вам придется изменить все 100 сценариев. Причем простая замена имен функций с `mysql_` на `ora_` не пройдет.

Что же делать? Первое, что приходит в голову, — создать класс вида:

```
<?php
class DBClass
```



```
{  
  
function DBClass()  
{  
}  
  
function connect($dbhost, $dbname, $login, $password)  
{  
    $con = mysql_connect($dbhost, $login, $password);  
    if($con === false)  
        return false;  
  
    return mysql_select_db($dbname, $con);  
}  
  
function query($sql)  
{  
    return mysql_query($sql);  
}  
...  
}
```

Этот класс нужно подключить ко всем сценариям и использовать только его методы класса при обращении к БД. Тогда для перехода на другой сервер баз данных вам нужно будет изменить только этот файл. Что намного проще, чем редактировать все ваши 100 сценариев. К тому же ваш проект станет намного более универсальным, ведь можно написать несколько разных классов — под разные СУБД, тогда перенос вашего проекта на другой сервер, где используется другая СУБД, не составит никакого труда.

Если вы действительно надумали создавать подобный класс, то могу заверить вас, что вы изобретаете велосипед. Поскольку все уже сделано за вас — вам следует лишь выбрать нужный вам класс из базы PEAR.

PEAR — это больше, чем просто набор классов. PEAR — это библиотека открытого кода для PHP-пользователей, система управления пакетами и распространения этих пакетов среди разработчиков, базовые классы PHP-кода, библиотека дополнительных модулей для PHP (The PHP Extension Code Library, PECL) и многое другое. Настоятельно рекомендую ознакомиться с официальным сайтом PEAR: <http://pear.php.net/manual/ru/introduction.php>.

26.2. Пример использования класса *DB*

Прежде всего нужно определиться, какие PEAR-пакеты вам нужны. Для этого посетите следующую страницу: <http://pear.php.net/packages.php>.

На момент написания этих строк в базе PEAR было 547 пакетов — этого хватит для большинства проектов. Не спешите качать все подряд. Нам понадобится только пакет PEAR, содержащий классы PEAR и PEAR_Error, на которые и опираются большинство (однако не все) пакетов из базы PEAR.

Последняя версия пакета PEAR доступна по адресу: <http://pear.php.net/package/PEAR/download>. В каталоге Glava_26 сопровождающего книгу электронного архива (см. приложение 4) вы найдете файл PEAR-1.9.0.tgz с версией 1.9.0 этого пакета.

Для работы PEAR настоятельно рекомендуется версия PHP 5.2.0 или более новая — хотя базовый класс PEAR может работать и с PHP 4, нет никакой гарантии, что другие классы смогут работать со старыми версиями PHP.

Распакуйте архив PEAR в какой-нибудь каталог, найдите в нем файл PEAR.php. Затем перейдите в каталог веб-сервера, где хранится ваш проект, и создайте в нем каталог pear. В этот каталог нужно скопировать найденный вами файл PEAR.php. Остальные файлы на данном этапе нам не нужны — для работы класса DB вполне хватит одного файла PEAR.php — нечего захламлять сервер ненужными файлами. Необходимо отметить, что это неполная установка PEAR, с полной установкой вы можете ознакомиться по адресу: <http://pear.php.net/manual/ru/installation.php>.

Последняя версия пакета DB доступна по адресу: <http://pear.php.net/package/DB/download>. В каталоге Glava_26 сопровождающего книгу электронного архива (см. приложение 4) вы найдете файл DB-1.7.14RC1.tgz с текущей версией этого пакета.

Распакуйте полученный архив. Мы опять не будем копировать на веб-сервер все его файлы, а скопируем только те, которые нам необходимы: подкаталог DB (он появится, когда вы распакуете архив) и файл DB.php.

Далее откройте файл DB.php и в самом начале найдите строку:

```
require_once 'PEAR.php'
```

Удалите или закомментируйте ее. Эта строка подключала файл PEAR.php, но поскольку мы устанавливали PEAR выборочно, то будем подключать его вручную в основном файле.

Теперь создайте файл test_pear.php и поместите его в корневой каталог сервера. У вас должна получиться следующая структура файлов и каталогов:

```
/-
|_ test_pear.php
|_ pear
|   |_ PEAR.php
|   |_ DB.php
|   |_ DB
|       |_ common.php
|       |_ dbase.php
|       ...
```

Отредактируем файл test_pear.php. Текст этого файла с моими комментариями представлен в листинге 26.1.

Листинг 26.1. Файл test_pear.php

```
<?php
require_once('pear/PEAR.php');
require_once('pear/DB.php');
```



```
// мы будем использовать MySQL
// oci8 - Oracle
// msql - MS SQL
// sqlite - SQL Lite
// sybase - Sybase
// список других доступных "драйверов" можно найти в каталоге pear/DB
$myDB =& DB::factory('mysql');

// подключаемся к базе данных
$myDB -> connect (DB::parseDSN('mysql://логин:пароль@сервер/база_данных'));

// произвольный запрос
$q = 'SELECT * FROM main_table';

// получаем результат
$result = $myDB -> query($q);

// проверяем, получен ли результат
if(!DB::isError($result) && $result -> numRows() > 0) {
    while($row = $result -> fetchRow(DB_MOD_ASSOC)) {
        // результат получен, выводим его.
        print('field_1 = '.$row['field_1']);
    }
}
?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *26-2.php*), представленный в каталоге *Glava_26* сопровождающего книгу электронного архива (см. приложение 4).

Рассмотрим листинг 26.1 подробнее. Первым делом мы подключили необходимые нам файлы:

```
require_once('pear/PEAR.php');
require_once('pear/DB.php');
```

Далее мы выбираем тип СУБД:

```
$myDB =& DB::factory('mysql');
```

После этого устанавливаем соединение с базой данных:

```
$myDB -> connect (DB::parseDSN('mysql://логин:пароль@сервер/база_данных'));
```

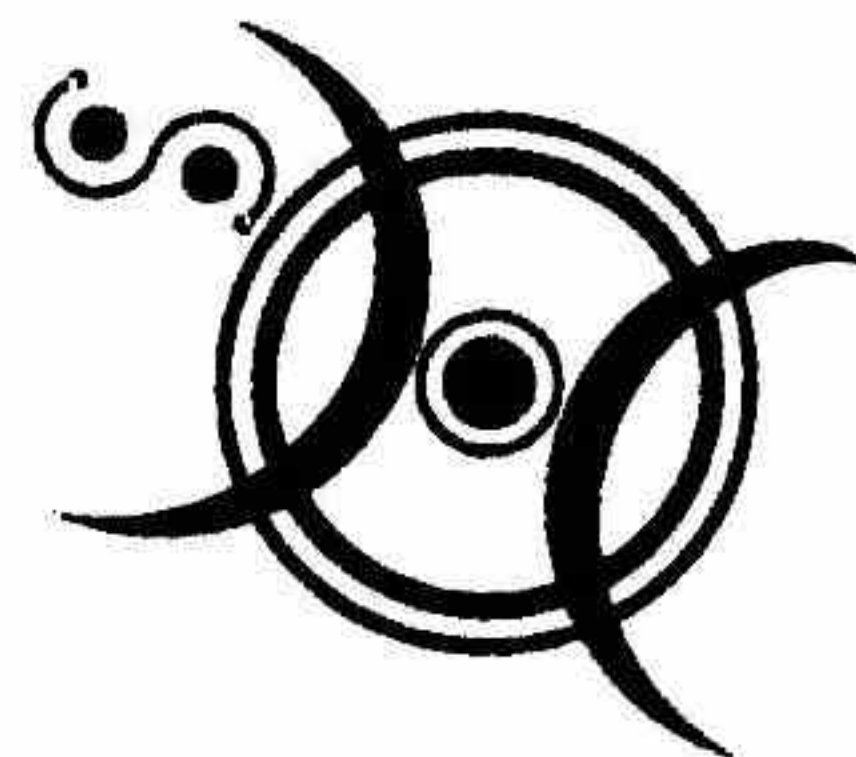
Затем выполняем произвольный запрос:

```
$q = 'SELECT * FROM main_table';
$result = $myDB -> query($q);
```


Все остальное — это обработка результата. С другими методами класса `DB` можно ознакомиться в официальной документации (к счастью, она на русском языке): <http://pear.php.net/manual/ru/package.database.db.php>. Документация (на русском языке) по другим классам PEAR доступна по адресу: <http://pear.php.net/manual/ru/>.

Для большего удобства (и чтобы сэкономить вам трафик) комплект документации на русском языке (файл архива `pear_manual_ru.zip`) я разместил в каталоге `Glava_26` сопровождающего книгу электронного архива (см. *приложение 4*).

ГЛАВА 27



Контроль версий

27.1. Выбор системы контроля версий

Если вы работаете над крупным проектом, вам просто необходима система контроля версий (далее — просто система). Система будет полезна как программисту-одиночке (фрилансеру), так и группе программистов, работающих над одним проектом. В последнем случае такая система не просто приятное дополнение к набору инструментов, а необходимость. Итак, разберемся, зачем вам нужна система контроля версий. По меньшей мере, есть три причины использовать контроль версий в вашем проекте:

- ☐ система позволяет сэкономить время и сберечь ваши нервы, если вы случайно испортите какие-то файлы или внесете изменения, после которых ваш проект вообще откажется работать. Да, можно закомментировать старый код перед тем, как писать новый. И новый код всегда можно будет потом удалить и раскомментировать старый, но зачем себя так утруждать, если есть система контроля версий, которая поможет восстановить предыдущее состояние файла? В Windows 7 встроен механизм теневых копий файла, позволяющий восстановить предыдущее содержимое файла, но Windows 7 используют не все, да и с оболочкой системы контроля версий работать удобнее;
- ☐ при групповой работе над проектом всегда видно, кто, когда и какие изменения вносил в файлы проекта. Вася Пупкин уже не сможет избежать ответственности за код с ошибками, внесенный в проект;
- ☐ вы имеете возможность безболезненно и с наименьшими затратами времени объединить фрагменты разных проектов в новый проект.

Существуют различные системы контроля версий. Одной из самых первых систем контроля версий (но она продолжает использоваться на данный момент многими программистами) является CVS (Concurrent Versions System). Прочитать о CVS можно в Википедии по адресу: <http://ru.wikipedia.org/wiki/CVS>. В настоящее время CVS считается устаревшей, а ей на смену пришла система Subversion, прочитать о которой можно на другой странице Википедии: <http://ru.wikipedia.org/wiki/Subversion>.

Изначально обе эти системы предназначались для работы в UNIX-подобных системах (Linux, FreeBSD), а под Windows были разработаны лишь графические клиенты для них. Это означает, что вам понадобится выделенный UNIX-сервер для репозитория — специального хранилища для вашего проекта и истории изменений. Небольшим компаниям (которые обычно используют Windows на своих рабочих станциях) или фрилансерам выделенный сервер для проектов экономически нецелесообразен. Можно, конечно, хранить свой репозиторий на бесплатных интернет-серверах, но при работе над коммерческими проектами, когда вы не хотите передавать свой код третьей стороне, это недопустимо, да и создаст лишнюю нагрузку на интернет-канал — если у вас неограниченное соединение, то придется потратиться еще и на оплату трафика. Кроме того, обе эти системы (CVS и Subversion) довольно сложны и обеспечивают намного больше функций, чем обычно требуется. Если вам нужна сложная система контроля версий, в Интернете вы найдете множество статей (в том числе и на русском языке) по использованию CVS и Subversion. Не следует забывать и об официальной документации. Не хочется, чтобы эта глава превратилась в очередной обзор функций CVS или Subversion — за эту книгу вы платили деньги, а такой обзор можно прочитать в Интернете бесплатно. Поэтому сейчас мы поговорим о совершенно другой системе контроля версий — Mercurial, которую можно установить на рабочей станции под управлением Windows, и этот же компьютер будет хранить основной репозиторий вашего проекта. Для многих программистов такое решение является наиболее приемлемым.

Скачать оболочку TortoiseHG для Mercurial можно по адресу:

http://sourceforge.net/project/showfiles.php?group_id=199155&package_id=236242&release_id=654501.

Поскольку Mercurial встроена в инсталлятор TortoiseHG, то отдельно ее устанавливать не нужно.

27.2. Практическое использование TortoiseHG (Mercurial)

Предположим, вы решили создать в папке project новый проект. Пока в ней есть только "пустой" файл index.php следующего содержания:

```
<?  
?>
```

Откройте файловый менеджер (Проводник, Total Commander и т. п.), щелкните правой кнопкой на папке project и выберите команду **TortoiseHG | Create Repository Here** (рис. 27.1). В открывшемся окне нажмите кнопку **Create** (рис. 27.2), и вы увидите сообщение, что репозиторий создан.

Следующий шаг — создание первой версии проекта. Для этого опять щелкните правой кнопкой на папке project и выберите команду **HG Commit** (рис. 27.3).

В открывшемся окне (рис. 27.4) выберите файлы, по которым нужно отслеживать изменения. У нас пока есть только один файл (index.php), поэтому выберем его. После этого нажмите кнопку **Commit**.

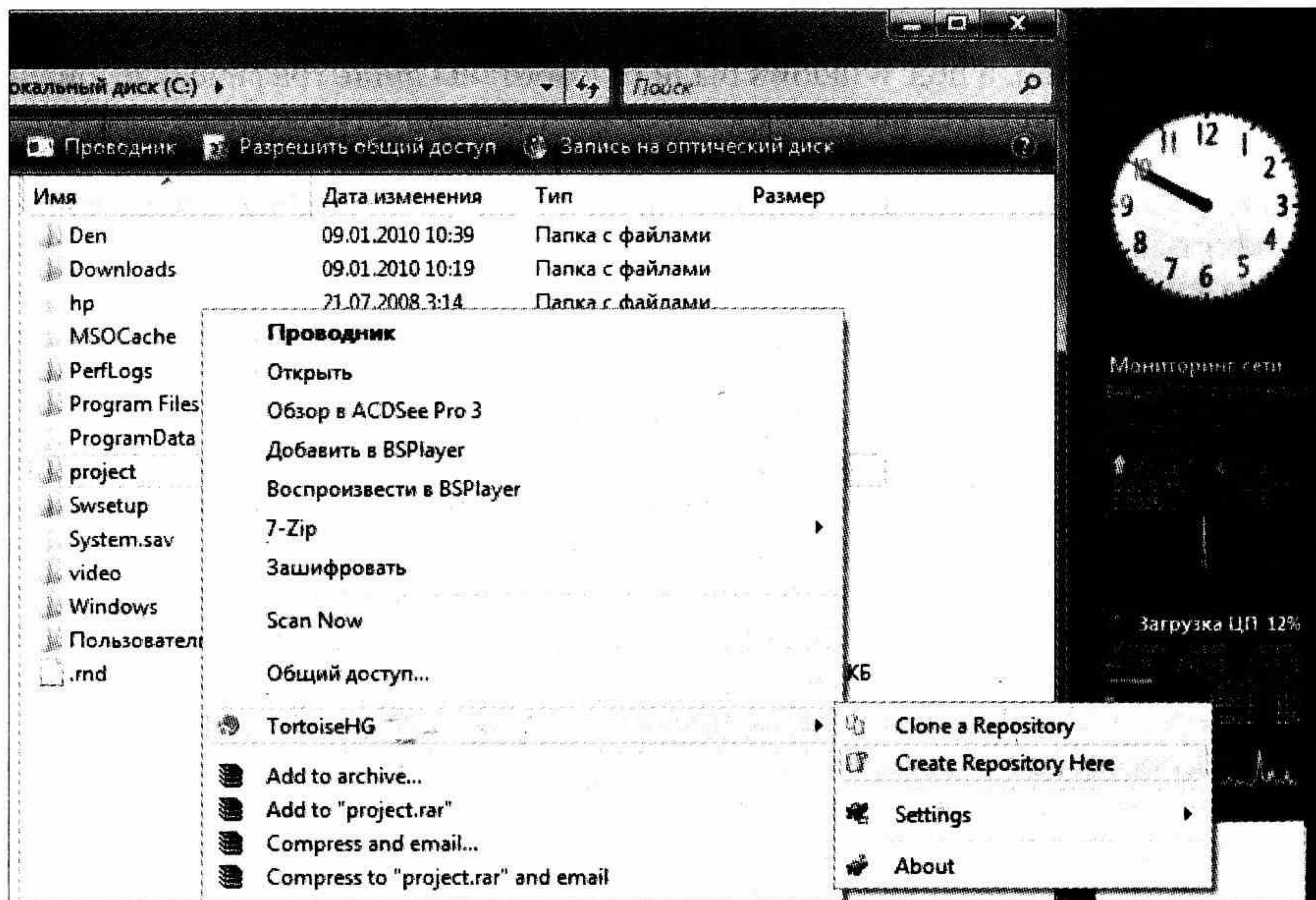


Рис. 27.1. Создание репозитория проекта

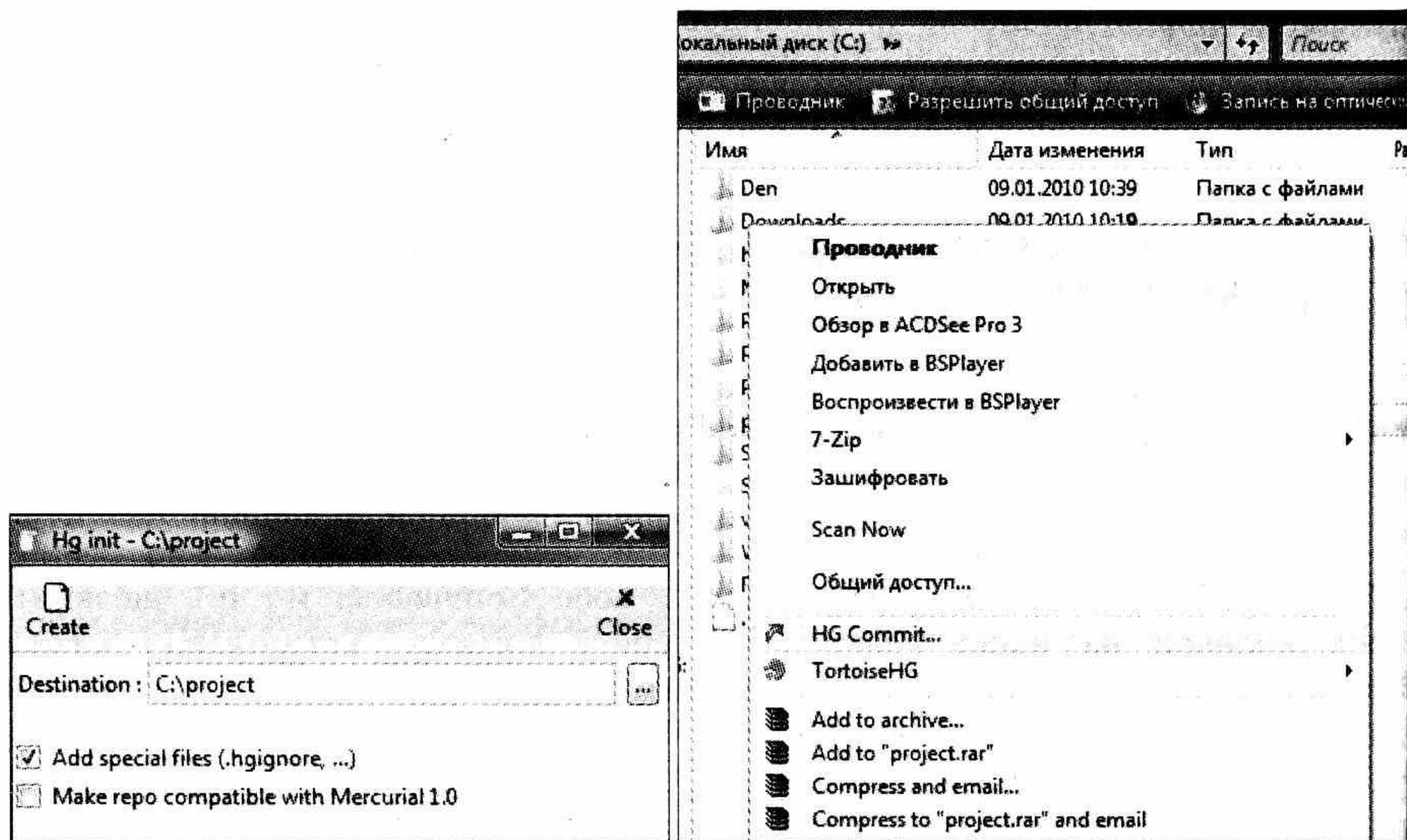


Рис. 27.2. Нажмите кнопку Create

Рис. 27.3. Выберите команду HG Commit

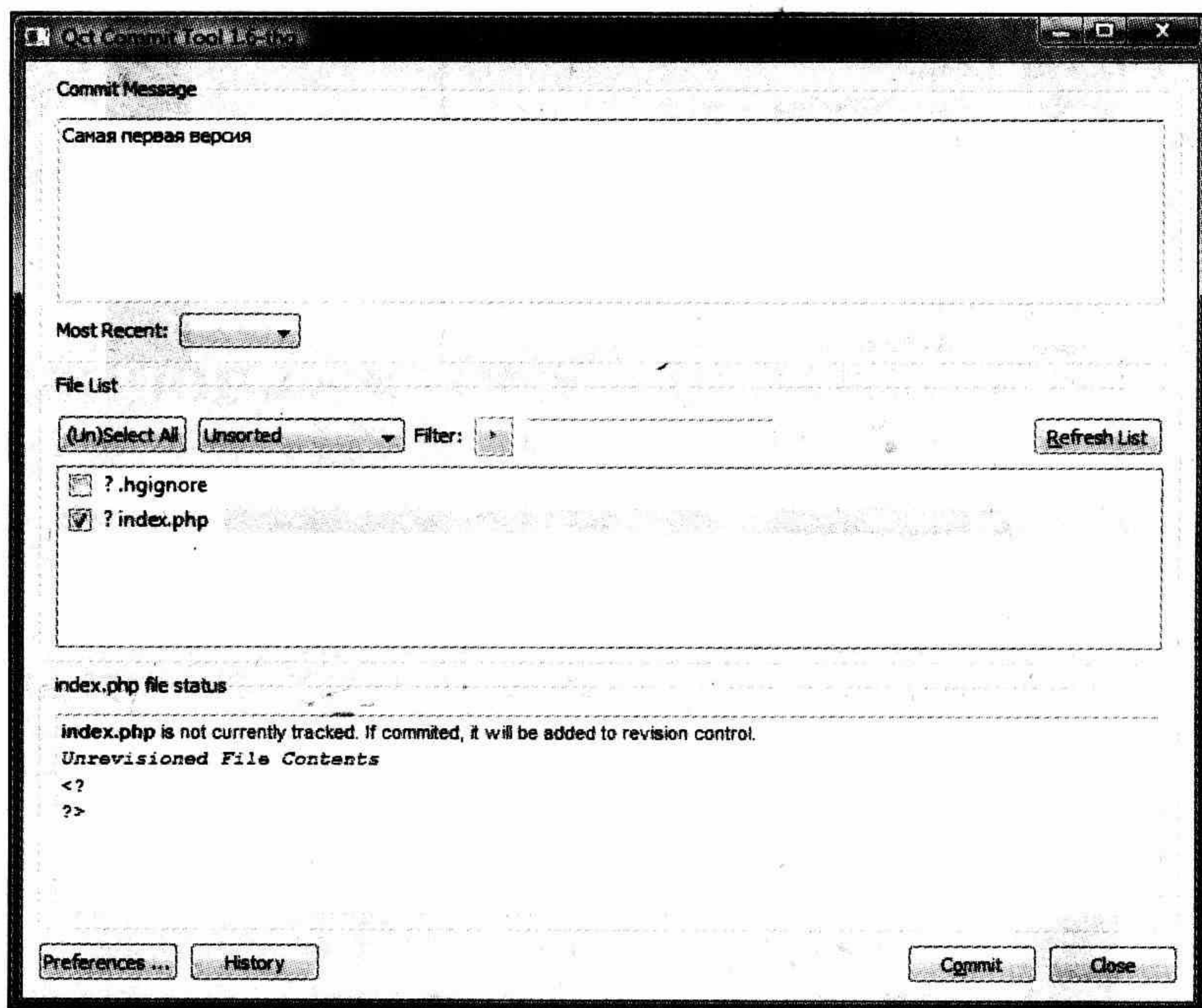


Рис. 27.4. Выберите файлы, по которым нужно отслеживать изменения

27.3. Просмотр внесенных изменений

Откройте любой текстовый редактор и внесите какие-либо изменения в файл `index.php`. После этого щелкните правой кнопкой мыши на каталоге `project` и выберите команду **TortoiseHG | Visual Diff** (рис. 27.5).

Открывшееся окно будет разделено на две части. В левой части окна — предыдущий вариант файла, в правой — измененный вариант (рис. 27.6).

Далее все просто:

- если изменения вас устраивают, щелкните правой кнопкой на каталоге `project` и выберите команду **HG Commit**. Введите описание новой версии проекта, выберите файлы, с изменениями в которых вы согласны, и нажмите кнопку **Commit**;
- если изменения вас не устраивают, выберите команду **TortoiseHG | Undo Changes**. В открывшемся окне выберите файлы, в которых нужно отменить изменения.

Для добавления новых файлов в проект служит команда **TortoiseHG | Add Files**, а для удаления файлов — **TortoiseHG | Remove Files**.

Как видите, система контроля версий Mercurial очень проста в использовании, а ее возможностей вполне достаточно для рядового PHP-программиста.

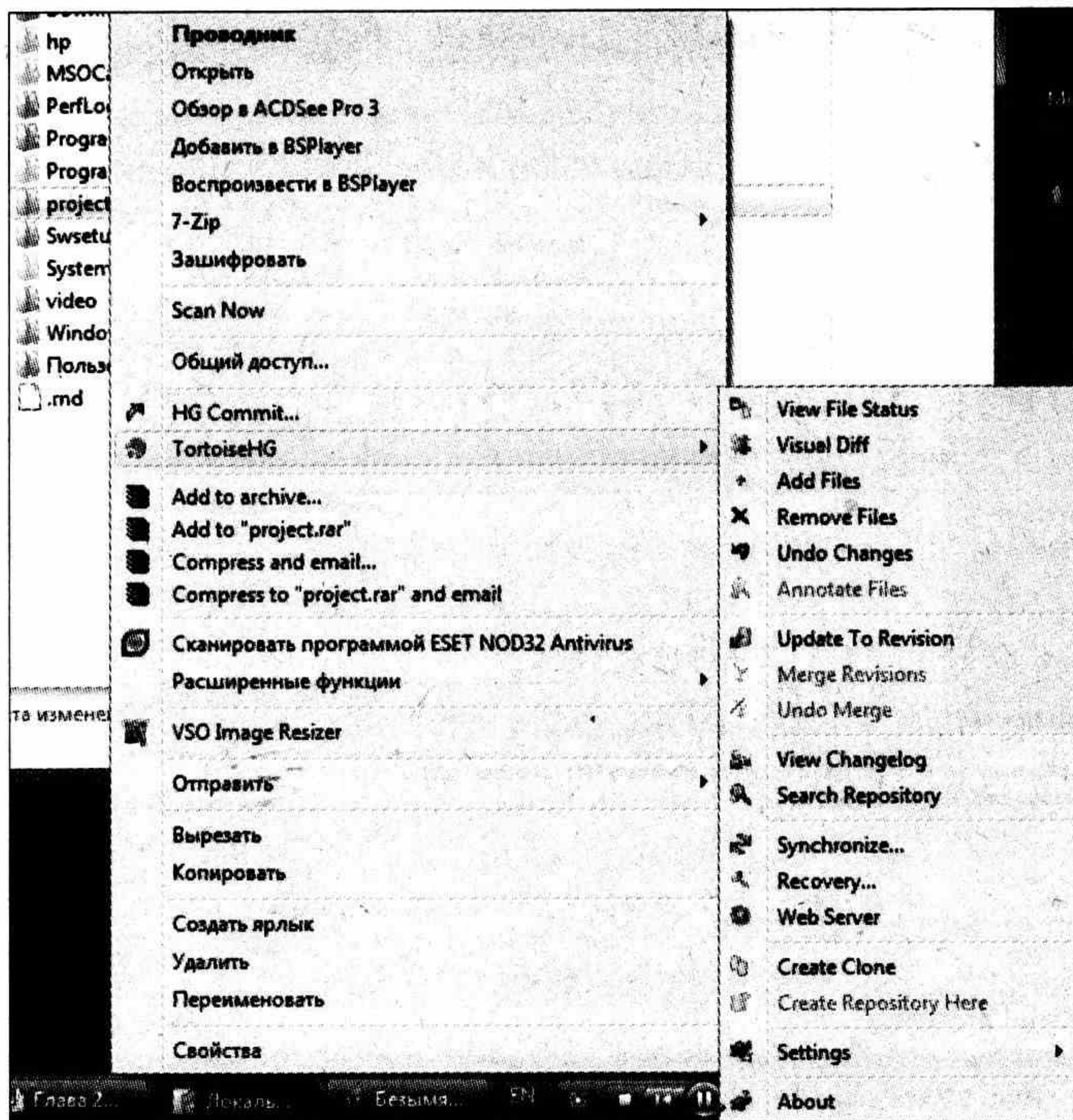


Рис. 27.5. Контекстное меню TortoiseHG

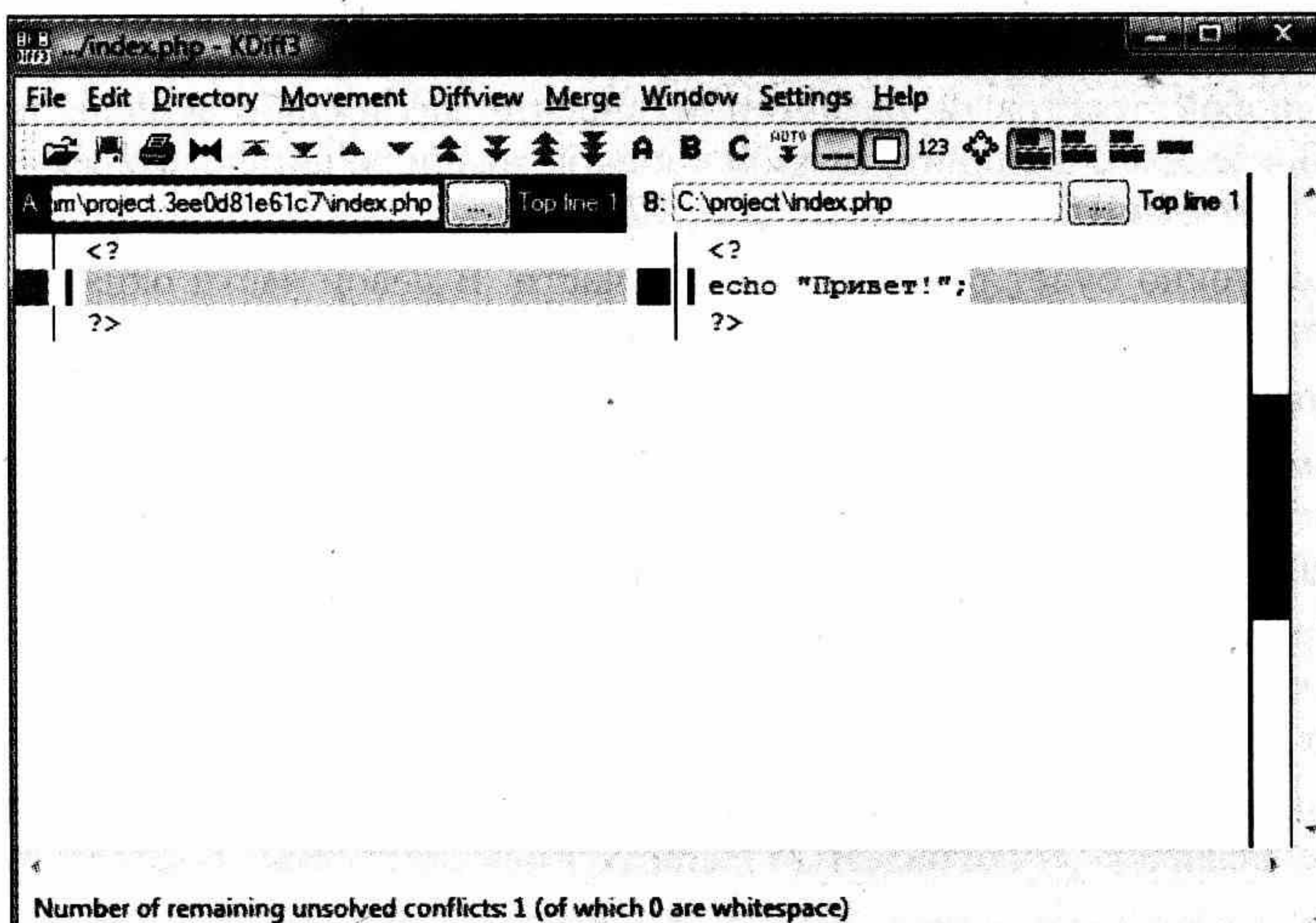
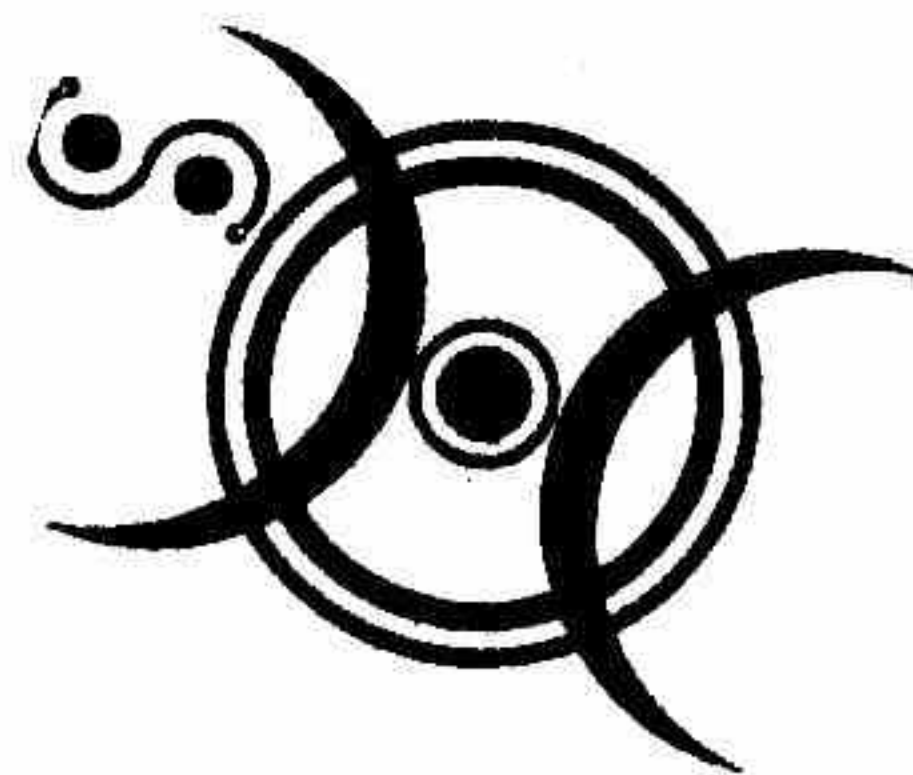


Рис. 27.6. Просмотр изменений в файле

ГЛАВА 28



Тестирование RНР-сценариев

28.1. Программа работает, но не так, как нам нужно

Знакома ли вам такая ситуация? Вроде бы все правильно, программа запускается, ошибок нет, что-то она делает, но явно не то, чего мы от нее ожидаем. Возникает вопрос — в чем же причина? Возможно, какая-то функция вернула неправильный результат, возможно, мы передали функции вследствие логической ошибки неверное значение. Да, именно логические ошибки — самые коварные. Ведь с точки зрения синтаксиса все написано правильно. Синтаксические ошибки выявить просто, поскольку их помогает устранить интерпретатор. А вот ошибки в логике программы интерпретатор устранить не в состоянии, поскольку он пока еще не умеет читать наши мысли.

Если вы ранее программировали на C++, Object Pascal или Java, то вам будет особенно сложно. Ведь вы привыкли к строгому коду, да и компиляторы этих языков программирования выявляют очень много различных ошибок, причем не только синтаксических, но иногда помогают устранять и логические ошибки. Еще раз повторюсь, что работать с RНР будет намного труднее. Во-первых, RНР не требует обязательной инициализации переменных, как в случае с другими языками программирования. Во-вторых, само понятие "тип переменной" здесь довольно размыто. Вот небольшой пример:

```
$a = 'test string';  
$a = 10-5;  
$b = '5';  
$c = $a + $b;
```

Сначала переменная `$a` у нас является строкой (в том же Object Pascal это тип `string`). Затем `$a` присваивается целое значение (5) — выходит, она теперь стала переменной целого типа. Переменная `$b` должна быть строкой, поскольку ее значение заключено в апострофы, но, как видите, компилятор "ни слова не скажет", когда вы попытаетесь сложить переменные двух разных типов: целого (переменная

\$a) и строкового (\$b). Переменная \$c, как ни в чем не бывало, будет содержать целое значение 10.

Попробуйте реализовать такое в C++. Без функций преобразования типов у вас ничего не выйдет, а в PHP — пожалуйста, типы переменных преобразуются "на лету". С одной стороны, это хорошо, с другой — не очень, поскольку строгий код C++ (Pascal, Java) дисциплинирует программиста. Да и такое преобразование может стать источником проблем. Допустим, у нас есть функция, которая ожидает параметр определенного типа — например, строку. А вы по ошибке передаете ей переменную, содержащую целое значение. Интерпретатор автоматически преобразует тип переменной, но, понятно, что функция будет работать неправильно. Скажем, функция ожидала имя файла, а вы передали ей какое-то число.

Кроме того, наряду с преобразованием типов переменных "на лету" и отсутствием объявлений переменных непривычным является символ доллара, с которого в PHP начинается имя каждой переменной. Почему это так, думаю понятно. В том же C++ или Pascal нужно сначала объявить переменные, т. е. указать имя переменной и ее тип (можно сразу задать и ее значение). А в PHP, как уже было отмечено, тип не является обязательным атрибутом переменной. Конечно, можно было бы в начале каждого сценария указывать имена переменных и их значения, но это неудобно. Поэтому, чтобы выделить переменные из прочей "серой массы" кода, приходится использовать знак доллара. Программист, не привыкший к значку \$, будет делать много ошибок (особенно поначалу). Вот небольшой пример:

```
for ($i = 0; $i < N; $i++) echo $x[$i];
```

А вот еще один:

```
for ($i = 0; $i < 100; $i++) echo $x[i];
```

Почему вывод пуст? Ведь массив содержит данные. А потому что в первом случае забыли указать знак доллара для переменной \$N, а во втором — для \$i. А ведь интерпретатор при этом не сказал ни слова, он просто молча "проглотил" код.

За свободу, как видите, приходится расплачиваться большим количеством ошибок в коде, которые очень непросто выявить. К тому же некоторые наиболее коварные ошибки проявляются только при определенных условиях, а в остальных случаях все работает отлично.

28.2. "Самодельные" точки останова

Если вы когда-нибудь программировали на C++ или Pascal, то знаете, что в любом отладчике есть возможность установки точки останова (breakpoint). Вы выбирали произвольную (нужную вам) строчку кода и устанавливали точку останова. Отладчик выполнял программу до заданной точки останова, и выполнение программы приостанавливалось. А вы во время "рекламной паузы" могли просмотреть значения любой переменной (предварительно, переменные нужно было внести в Watch-список).

Для PHP отладчика, подобного описанному, пока нет (точнее, есть, но за него нужно выложить приличную сумму — Zend Studio стоит 400 долларов в год). Можно, конечно, использовать среду разработки Eclipse, но это, во-первых, не всегда возможно, а, во-вторых, зачастую не дает желаемых результатов. Поэтому попробуем организовать отладчик своими руками.

Роль точки останова будет выполнять функция `die()`, завершающая работу сценария. В нужном месте вы вызываете эту функцию, а в качестве ее параметра передаете все переменные, которые хотите вывести. Если выводить значения переменных через функцию `die()` вам неудобно, можно сначала вывести значения переменных, а затем вызвать `die()`.

Вот небольшой пример:

```
$a = 10; $b= 'test';  
...  
die('a = '.$a. ' b = '.$b);
```

У данного решения всего один недостаток — точка останова может быть только одна, и мы не сможем продолжить выполнение после вызова функции `die()`. Но это все же лучше, чем вообще ничего.

28.3. Система автоматического тестирования

Немного облегчить отладку PHP-приложений поможет система автоматического тестирования SimpleTest, являющаяся развитием систем JUnit для Java, точнее, ее аналогом для PHP. Скачать систему можно по адресу: <http://sourceforge.net/projects/simpletest/files/>. Здесь доступна как сама система, так и модуль для системы Eclipse, если вы ею пользуетесь. В каталоге Glava_28 сопровождающего книгу электронного архива (см. приложение 4) вы найдете файл `simpletest_1.0.1.tar.gz` с версией 1.0.1 этого пакета.

Распакуйте архив системы в каталог `simpletest`. В каталоге `simpletest/docs` содержится подробное описание системы, а в каталоге `simpletest/test` — много различных примеров.

Разберемся, как работает система, на простейшем примере ее использования. Создайте файл `ex.php` и поместите его в каталог `simpletest`. Код этого файла приведен в листинге 28.1. Внимательно читайте комментарии, чтобы понять, что и как происходит. Можно, конечно, было бы описать методы отдельно, но вместе с реальным кодом комментарии воспринимаются проще.

Листинг 28.1. Пример использования системы автоматического тестирования

```
<?php  
  
// подключаем необходимые файлы  
require_once ("simpletest/unit_tester.php");  
require_once ("simpletest/reporter.php");
```



```
// пример тестирования кода PHP с помощью SimpleTest

// создаем класс PHPSimpleTest, который расширяет стандартный
// класс UnitTestCase
class PHPSimpleTest extends UnitTestCase {

function setUp() {

// здесь нужно задать действия, которые должны быть выполнены
// до вызова функции тестирования testCode1()

}

function tearDown() {
// здесь нужно задать действия, которые должны быть выполнены
// после вызова функции тестирования testCode1()

}

// а вот сама функция тестирования PHP-кода
function testCode1()
{

// определяем переменную $a
$a=2*5;

// проверка на равенство
$this->assertEqual($a,10);

// определяем массив
$ARR=array($a,10);

// вывод значения $ARR и $a
$this->dump($ARR);
$this->dump($a);

// производим проверку на равенство значения и соответствие типа
$this->assertIdentical($ARR,array(10,10));

// проверяем значение на истинность
$this->assertTrue($a==10);

// проверяем значение на ложность
$this->assertFalse($a!=10);
}
}
```



```
// вызываем тестирование
$test = new PHPSimpleTest();
$reporter = new HTMLReporter();

$test->run($reporter);

?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *ex.php*), представленный в каталоге *Glava_28* сопровождающего книгу электронного архива (см. приложение 4).

Запустите сценарий *ex.php* в окне браузера. Результат его выполнения представлен на рис. 28.1.

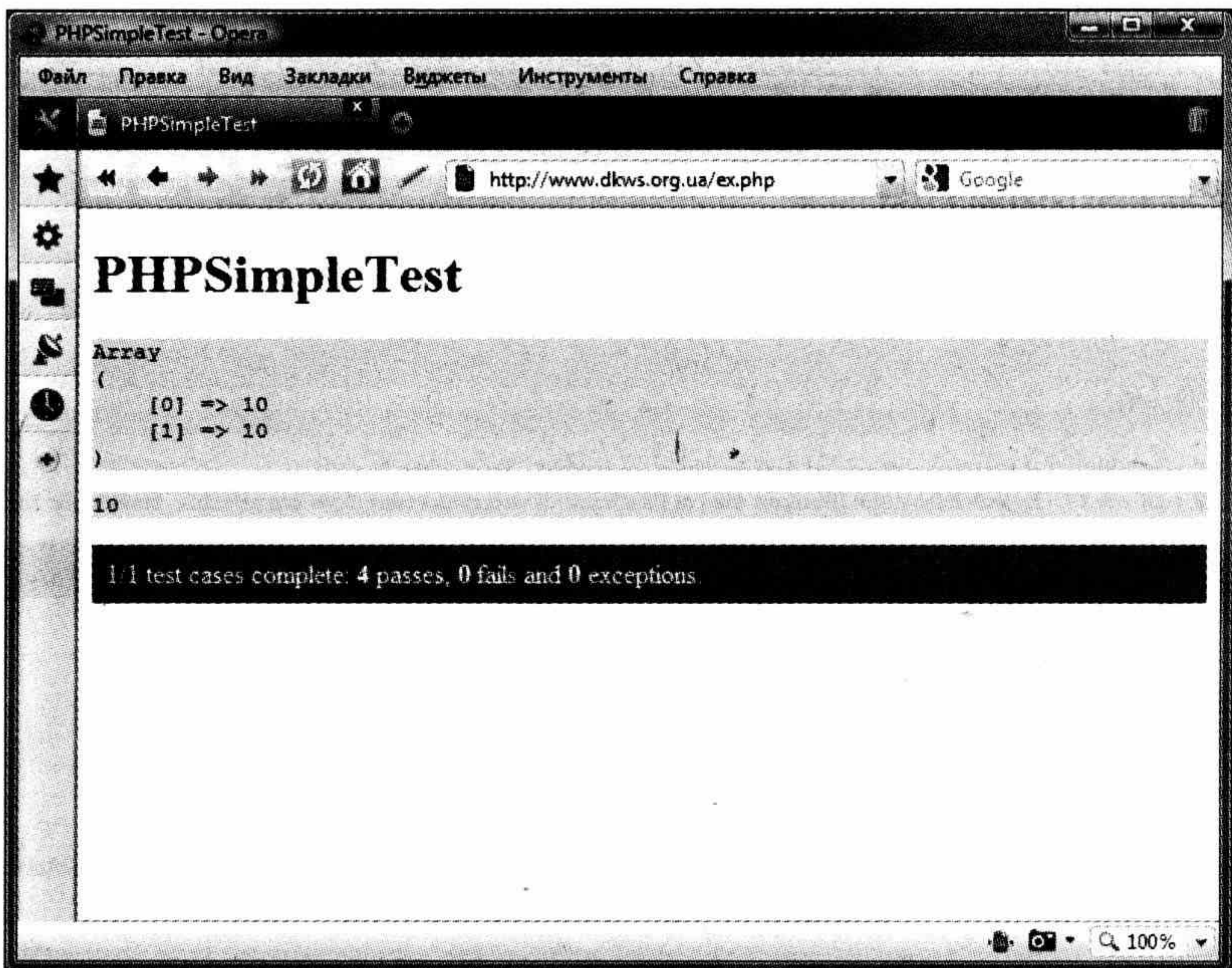


Рис. 28.1. Результат проверки кода

Как видно из рис. 28.1, сначала выведены значения массива *\$ARR* и переменной *\$a*, а затем выводится общий отчет о тестировании — проверена одна тест-функция, которая прошла все четыре теста:

```
// проверка на равенство
$this->assertEqual($a,10);
// производим проверку на равенство значения и соответствие типа
$this->assertIdentical($arr,array(10,10));
```



```
// проверяем значение на истинность
$this->assertTrue($a==10);
// проверяем значение на ложность
$this->assertFalse($a!=10);
```

Для лучшего понимания сделайте следующий эксперимент — перед строкой

```
$this->assertTrue($a==10);
```

добавьте строку

```
$a = 9;
```

Этим мы сделали все возможное, чтобы наша функция "провалила" тест. Запустите еще раз сценарий и посмотрите на результат (рис. 28.2).

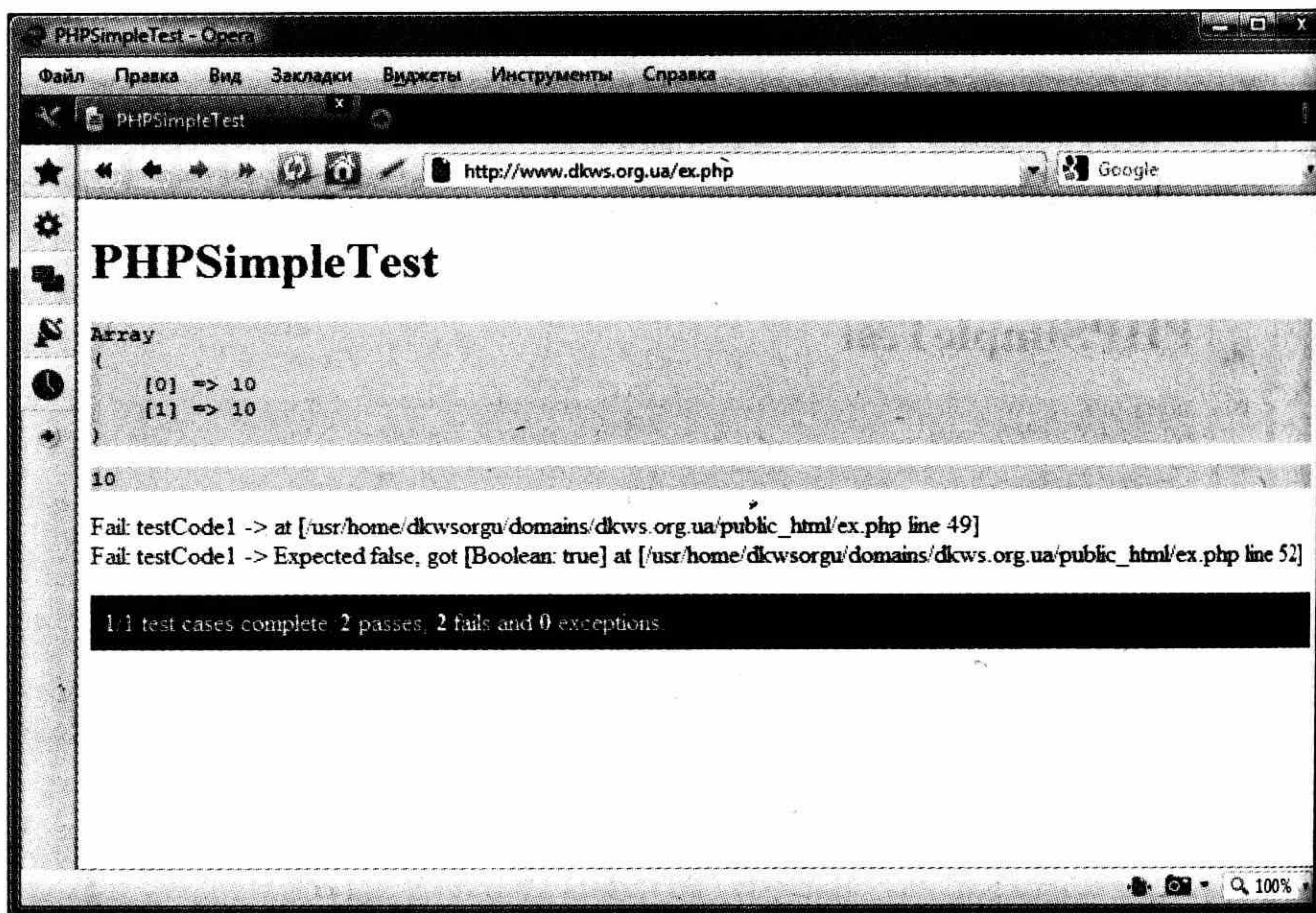


Рис. 28.2. Функция "провалила" два теста

Поскольку мы присвоили переменной `$a` значение 9, то два последних теста оказались невыполненными:

- ☐ проверка на истинность условия, что переменная `$a` равна 10;
- ☐ проверка на ложность того, что `$a` не равна 10 (поскольку `$a = 9`, то результатом логического выражения `$a!=10` будет истина, а мы проверяли выражение на ложность).

Если одной функции для проверки кода вам мало, можно добавить еще одну. Функция должна называться `testCodeN()`, где `N` — номер функции. Например:


```
function testCode2()  
{  
    $B=12*15;  
    $this->dump($B);  
}
```

28.4. Директива *error_reporting*

При отладке сценариев не нужно забывать об очень полезной директиве: `error_reporting`. Откройте ваш файл `php.ini`. В Windows он находится или в основном каталоге Windows (`C:\Windows`), или в одном с PHP каталоге. В Linux этот файл может находиться в каталоге `/etc` или в `/etc/php5/cli` (зависит от вашего дистрибутива). В FreeBSD файл `php.ini` может находиться в каталоге `/usr/local/lib` или любом другом каталоге, указанном при сборке PHP.

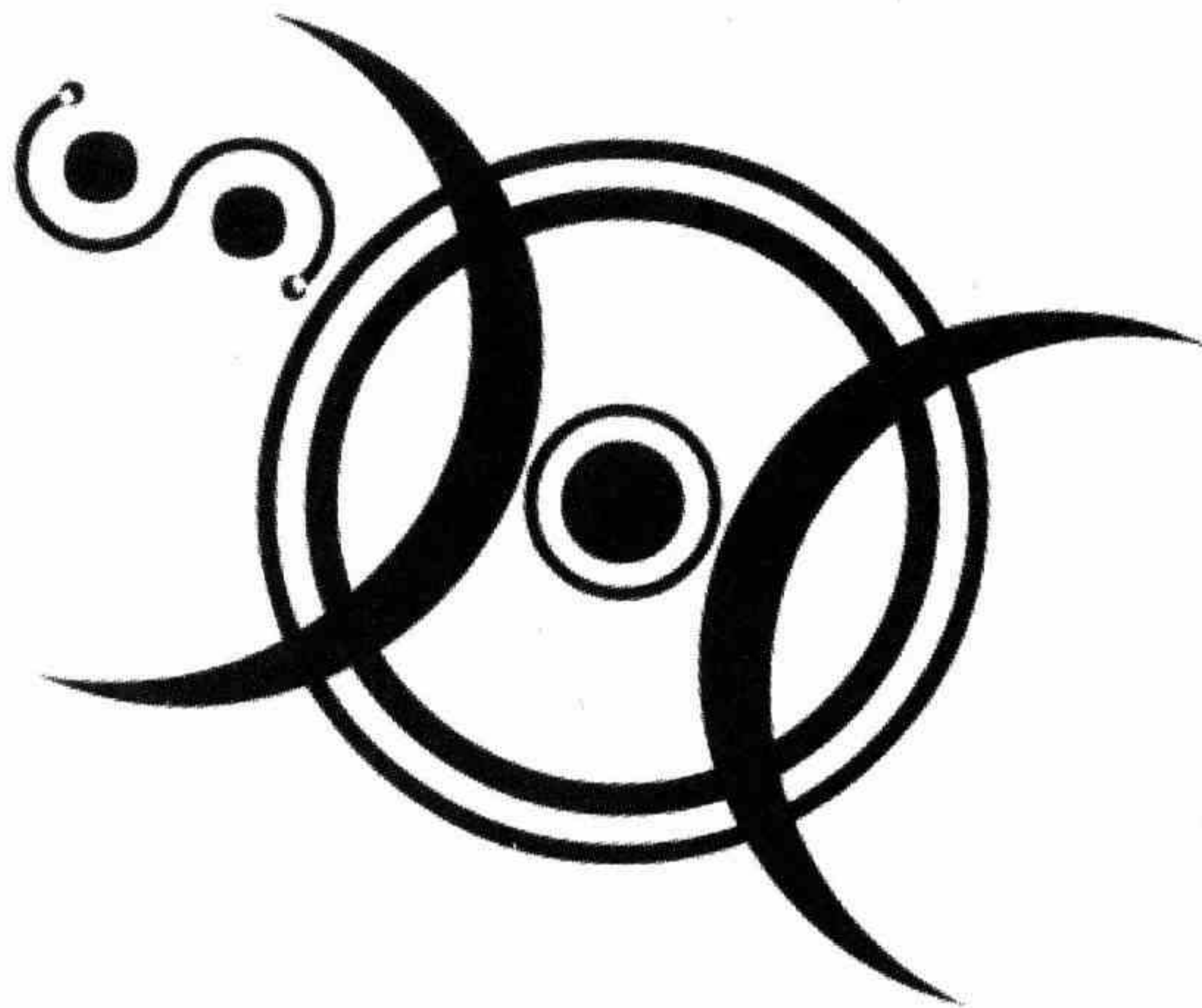
Найдите директиву `error_reporting` и установите для нее значение `E_ALL`:

```
error_reporting = E_ALL
```

Можно также установить и другие значения (они описаны в файле `php.ini`), но на этапе отладки лучше, чтобы PHP выводил все предупреждения и сообщения об ошибках.

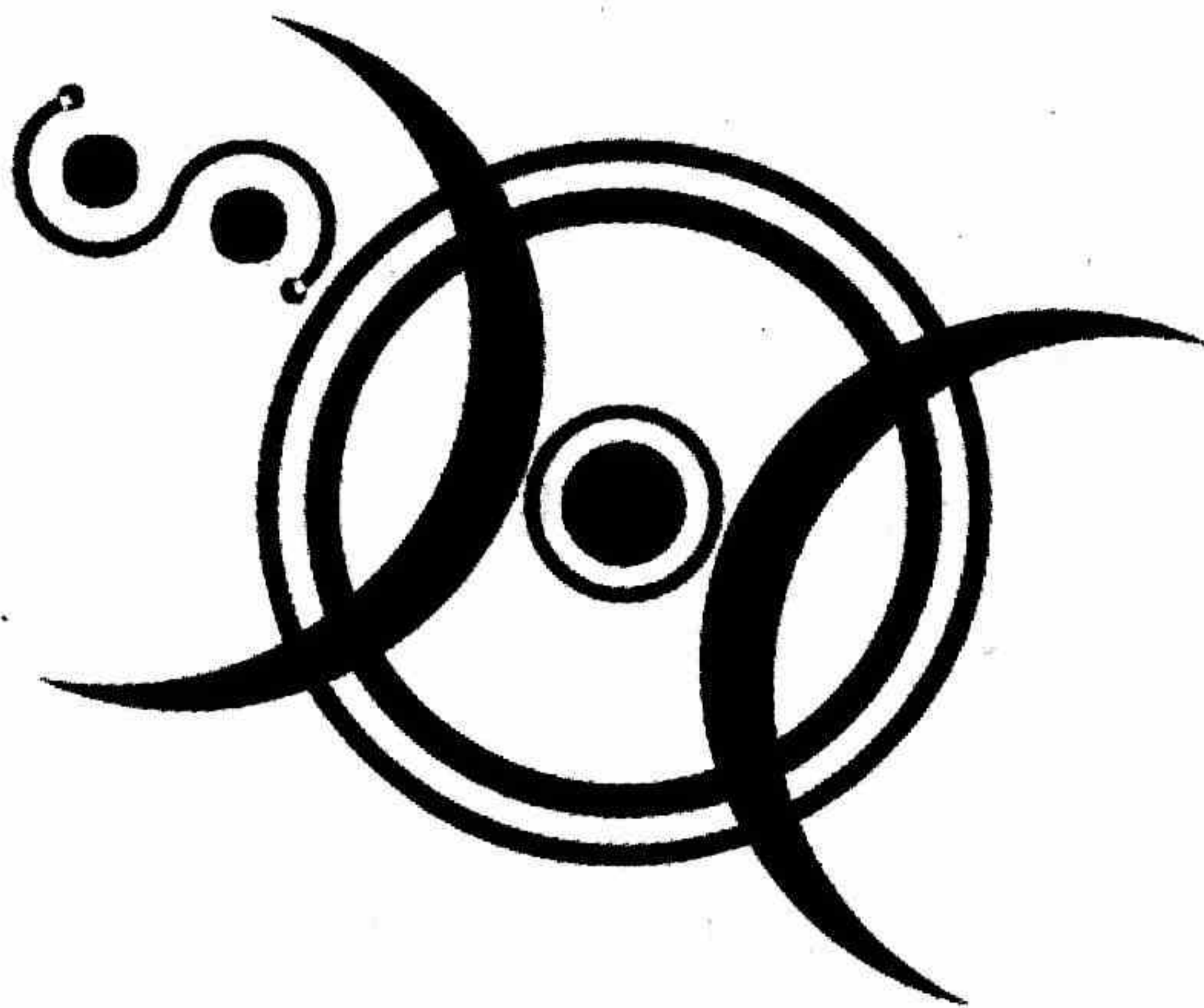
Установить значение данной директивы можно и во время работы сценария так:

```
error_reporting( 'E_ALL' );
```

ЧАСТЬ II

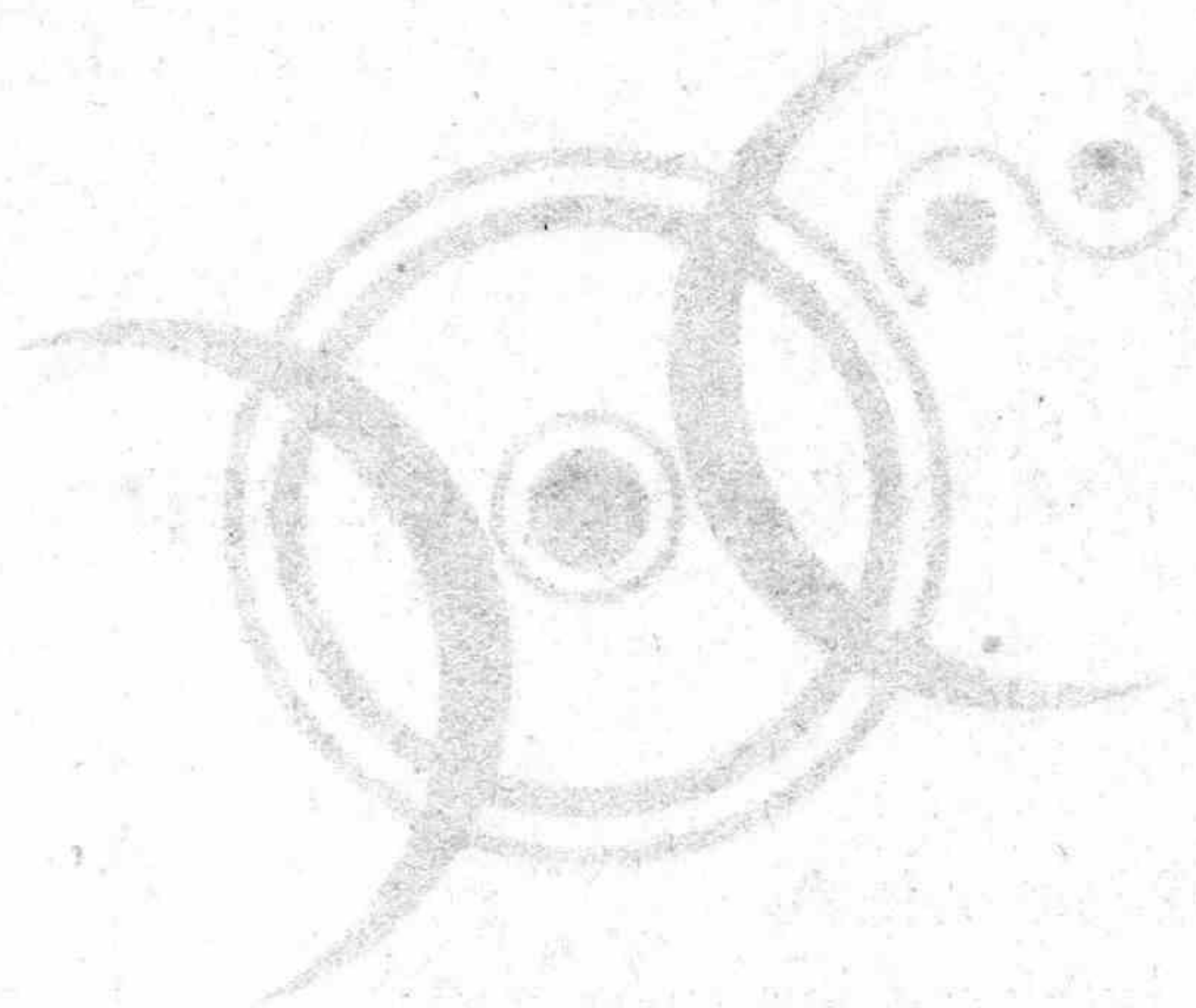
ПРАКТИКА



II. РАЗДЕЛ 7

Разработка основных элементов сайта

Глава 29.	Загрузка файлов на сервер
Глава 30.	Использование FTP-функций
Глава 31.	Отправка и прием почты
Глава 32.	Работа с RSS: получаем новости на сайт
Глава 33.	Поиск с использованием регулярных выражений
Глава 34.	Работаем с MP3
Глава 35.	Технология AJAX



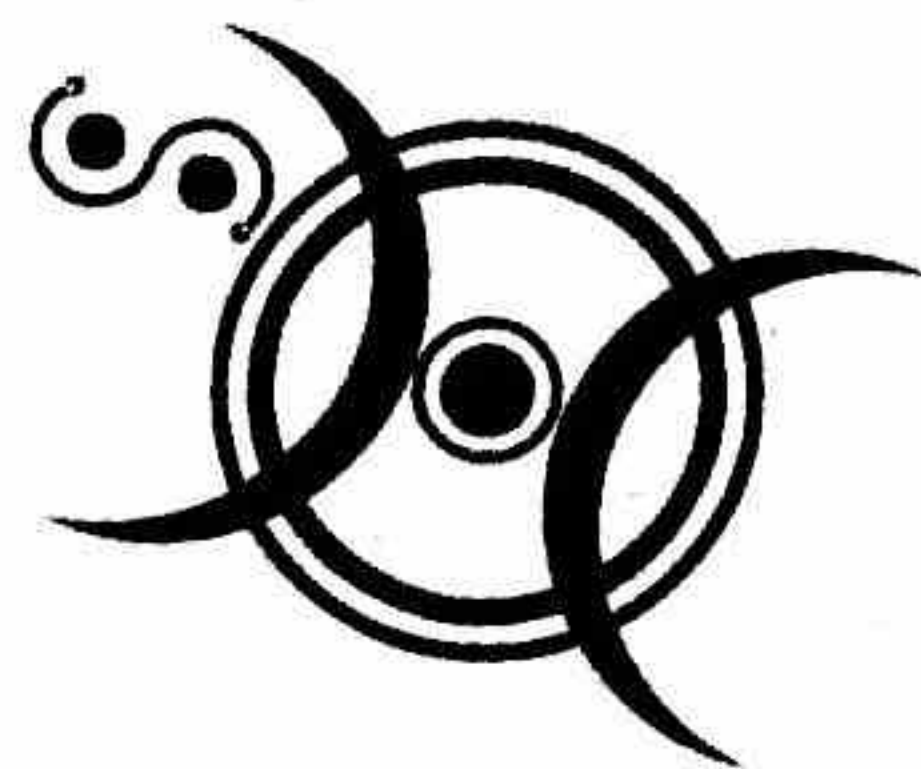
II. РАЗДЕЛ

РАЗРАБОТКА

ОСНОВНЫХ ЭЛЕМЕНТОВ САНКТ

1. Общие сведения	2. Основные элементы
3. Технические характеристики	4. Методы измерения
5. Результаты измерений	6. Анализ данных
7. Выводы	8. Заключение
9. Приложение	10. Литература
11. Справочные данные	12. Дополнительные сведения
13. Заключение	14. Литература
15. Приложение	16. Литература
17. Справочные данные	18. Дополнительные сведения
19. Заключение	20. Литература
21. Приложение	22. Литература
23. Справочные данные	24. Дополнительные сведения
25. Заключение	26. Литература
27. Приложение	28. Литература
29. Справочные данные	30. Дополнительные сведения
31. Заключение	32. Литература
33. Приложение	34. Литература
35. Справочные данные	36. Дополнительные сведения
37. Заключение	38. Литература
39. Приложение	40. Литература
41. Справочные данные	42. Дополнительные сведения
43. Заключение	44. Литература
45. Приложение	46. Литература
47. Справочные данные	48. Дополнительные сведения
49. Заключение	50. Литература
51. Приложение	52. Литература
53. Справочные данные	54. Дополнительные сведения
55. Заключение	56. Литература
57. Приложение	58. Литература
59. Справочные данные	60. Дополнительные сведения
61. Заключение	62. Литература
63. Приложение	64. Литература
65. Справочные данные	66. Дополнительные сведения
67. Заключение	68. Литература
69. Приложение	70. Литература
71. Справочные данные	72. Дополнительные сведения
73. Заключение	74. Литература
75. Приложение	76. Литература
77. Справочные данные	78. Дополнительные сведения
79. Заключение	80. Литература
81. Приложение	82. Литература
83. Справочные данные	84. Дополнительные сведения
85. Заключение	86. Литература
87. Приложение	88. Литература
89. Справочные данные	90. Дополнительные сведения
91. Заключение	92. Литература
93. Приложение	94. Литература
95. Справочные данные	96. Дополнительные сведения
97. Заключение	98. Литература
99. Приложение	100. Литература

ГЛАВА 29



Загрузка файлов на сервер

29.1. Что нужно знать о загрузке файлов на сервер?

Прежде всего следует выяснить, зачем нам это нужно. Оказывается, многие серьезные проекты требуют загрузки файлов на сервер (в англоязычной литературе данный процесс называется *upload*). Вот несколько примеров:

- форумы и чаты — загрузка аватар пользователей;
- фотогалерея — загрузка фотографий;
- интернет-магазин — загрузка фотографий товаров;
- веб-интерфейс почтовой службы — сначала нужно загрузить файл на сервер, а только после этого прикрепить его к письму;
- веб-интерфейс системы управления контентом сайта — в этом случае без загрузки файлов на сервер тоже никак не обойтись.

Как видите, загрузка файлов на сервер встречается очень часто. На самом деле эта операция довольно простая. И здесь мы поговорим о том, как реализовать загрузку своими силами, не прибегая для этого к использованию сторонних классов.

Для загрузки файлов на сервер существуют специальные формы загрузки — так называемые *multipart*-формы (листинг 29.1).

Листинг 29.1. *Multipart*-форма для загрузки файлов на сервер

```
<html>
<body>
<FORM ENCTYPE="multipart/form-data" ACTION="upload.php" METHOD=POST>
Выберите файл: <INPUT NAME="upfile" TYPE="file">

<INPUT TYPE="submit" VALUE="Загрузить">
```



```
</FORM>
</body>
</html>
```

ПРИМЕЧАНИЕ

Коды формы загрузки (файл *upload.html*) и сценария загрузки (файл *upload.php*), приведенного в листинге 29.3, вы найдете в каталоге *Glava_29* сопровождающего книгу электронного архива (см. приложение 4).

Multipart-форма отличается от обычной прежде всего наличием параметра `ENCTYPE="multipart/form-data"`. Но это еще не все. Вторым обязательным атрибутом такой формы — наличие поля для выбора файла:

```
<INPUT NAME="upfile" TYPE="file">
```

Все остальное реализуется как обычно: задается сценарий, который будет обрабатывать форму (параметр `ACTION`), метод передачи файла (`METHOD`) и кнопка передачи информации (параметр `submit`). В окне браузера наша multipart-форма выглядит так, как показано на рис. 29.1.

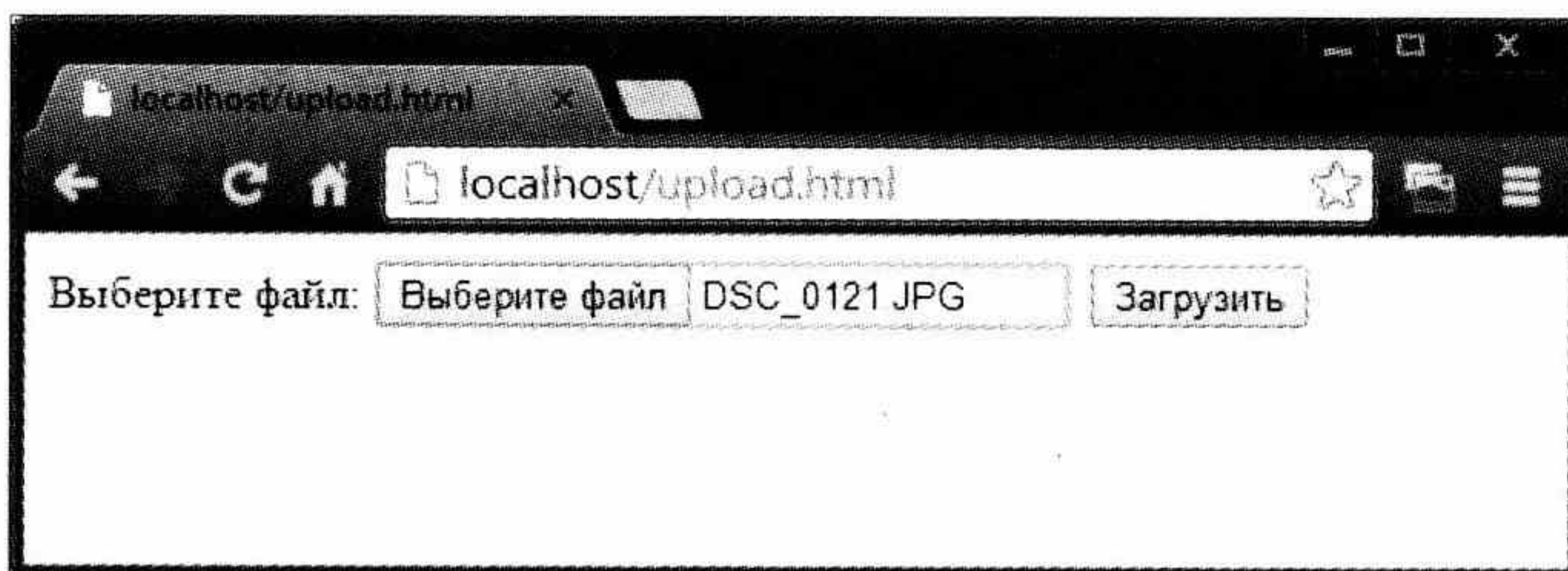


Рис. 29.1. Multipart-форма в окне браузера

Multipart-форма, безусловно, необходима. Но ее наличие еще совсем не гарантирует саму загрузку файлов на сервер. Для включения загрузки файлов нужно отредактировать файл конфигурации `php.ini`. Найдите в нем секцию `file uploads` и измените в ней следующие параметры:

- ☐ `file_uploads = On` — разрешает загрузку файлов на сервер;
- ☐ `upload_max_filesize = 2M` — устанавливает максимальный объем загружаемого файла;
- ☐ `upload_tmp_dir = /tmp` — создает временный каталог для загрузки файлов.

Для того чтобы изменения вступили в силу, нужно перезагрузить веб-сервер Apache.

Следует отметить, что редактирование файла конфигурации может понадобиться только на вашем компьютере для тестирования сценария, поскольку на сервере хостинг-провайдера обычно данные параметры уже установлены (это понятно, ведь хостинг-провайдер старается предоставить максимальное качество обслуживания).

Если вы хотите убедиться в том, что ваш хостинг-провайдер разрешил загрузку файлов на сервер, создайте файл `test.php` (листинг 29.2), загрузите его на сервер и запустите в окне браузера. Найдите параметр `file_uploads`. Если он установлен в значение `On`, значит, загрузка файлов на сервер разрешена (рис. 29.2). Там же вы найдете имя временного каталога и максимальный размер загружаемого файла.

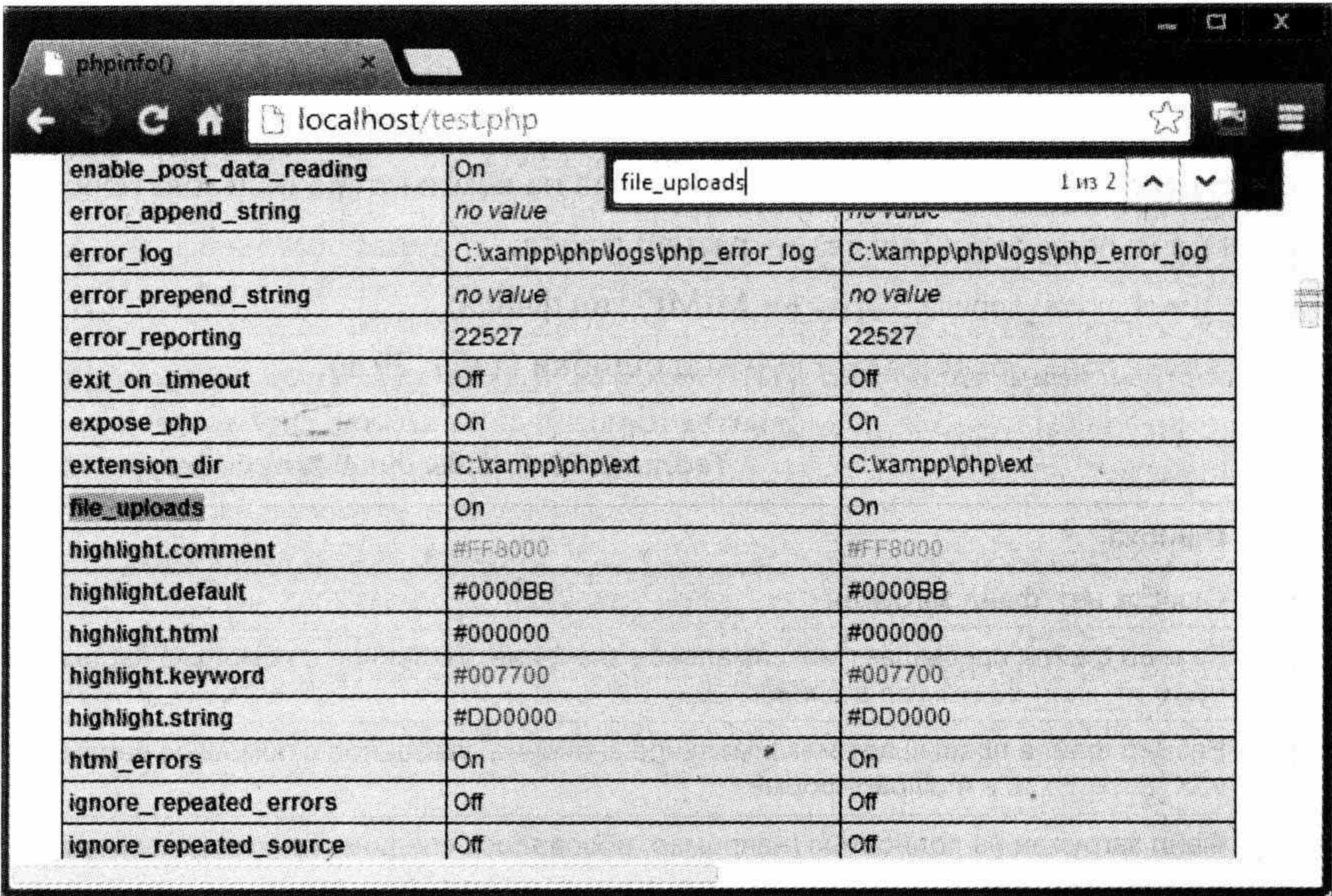


Рис. 29.2. Загрузка файлов включена

Листинг 29.2. Файл `test.php`

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

ПРИМЕЧАНИЕ

Код этого теста вы найдете в файле `29-2.php` каталога `Глава_29` сопровождающего книгу электронного архива (см. приложение 4).

Теперь, когда мы знаем, что загрузка файлов на сервер разрешена, и у нас есть `multipart`-форма, можно приступить непосредственно к самой загрузке файла.

29.2. Реализация загрузки файла

Задача multipart-формы заключается в передаче содержимого файла нашему сценарию. Должен вас обрадовать — PHP все сделает за вас, вам необходимо только обеспечить копирование файла в требуемый каталог.

В суперглобальном массиве `$_FILES` содержится информация о загружаемых файлах. Структура этого массива следующая:

- ❑ `$_FILES[имя_поля_input][tmp_name]` — PHP сохраняет принятые файлы во временном каталоге, в этом поле массива хранится имя временного файла;
- ❑ `$_FILES[имя_поля_input][name]` — имя файла на компьютере пользователя;
- ❑ `$_FILES[имя_поля_input][size]` — размер файла;
- ❑ `$_FILES[имя_поля_input][type]` — MIME-тип файла;
- ❑ `$_FILES[имя_поля_input][error]` — код ошибки (табл. 29.1).

Таблица 29.1. Коды ошибок при загрузке файлов

Код	Ошибка
0	Ошибок нет, файл загружен
1	Размер файла превышает максимальное значение, указанное с помощью параметра <code>upload_max_filesize</code> в <code>php.ini</code>
2	Размер файла превышает максимальное значение, указанное с помощью параметра <code>MAX_FILE_SIZE</code> в multipart-форме
3	Файл загружен не полностью (например, оборвалось соединение между сервером и клиентом)
4	Файл не загружен

Рассмотрим сценарий, реализующий загрузку файлов на сервер (листинг 29.3).

Листинг 29.3. Сценарий загрузки файла на сервер `upload.php`

```
<?php

// каталог для загрузки файлов
$dir = './upload/';

// в multipart-форме мы определили имя input-поля upfile
// это имя нужно использовать при работе с массивом $_FILES
if(isset($_FILES["upfile"]))
{
    $upfile      = $_FILES["upfile"]["tmp_name"];
    $upfile_name = $_FILES["upfile"]["name"];
    $upfile_size = $_FILES["upfile"]["size"];
```



```
$upfile_type = $_FILES["upfile"]["type"];
$error_code  = $_FILES["upfile"]["error"];

// если ошибок нет
if($error_code == 0)
{
    // выводим информацию о принятом файле
    echo "Имя файла на сервере: ".$upfile."<br>";
    echo "Имя файла на компьютере пользователя: ".$upfile_name."<br>";
    echo "MIME-тип файла: ".$upfile_type."<br>";
    echo "Размер файла: ".$upfile_size."<br><br>";

    // дополняем имя файла
    $upfile_name = $dir . $upfile_name;

    // копируем временный файл в каталог $dir, имя файла будет исходное,
    // т. е. как на компьютере у пользователя
    // первый параметр — источник
    // второй параметр — получатель
    copy($upfile, $upfile_name);

    // можно использовать функцию move_uploaded_file()
    //move_uploaded_file($upfile, $upfile_name);
}
}

?>
```

ПРИМЕЧАНИЕ

Напомню, что этот код представлен в файле *upload.php* каталога *Glava_29* сопровождающего книгу электронного архива (см. приложение 4).

Теперь проанализируем сценарий. Он будет копировать с помощью функции `copy()` принятый файл из временного каталога в каталог, заданный переменной `$dir`. Нужно позаботиться о том, чтобы этот каталог существовал, и установить для него права доступа 777:

```
chmod 777 ./upload/
```

Функция копирования `copy()`, в случае если файл-получатель существует, перезапишет его. Поэтому будьте осторожны — в реальном сценарии придется еще добавить функцию проверки существования файла-получателя и, если файл с заданным именем существует, к его имени дописывать произвольный символ.

Вместо `copy()` можно указать функцию `move_uploaded_file()`, которая не копирует, а перемещает файл. Особого смысла в этом я не вижу, поскольку серверы провайдера обычно настроены на автоматическую очистку временных каталогов.

29.3. Загрузка нескольких файлов

Возможно, вам потребуется реализовать автоматическую загрузку нескольких файлов. Особых проблем с этим нет. Просто измените multipart-форму так, как показано в листинге 29.4.

Листинг 29.4. Форма для одновременной загрузки трех файлов

```
<FORM ENCTYPE="multipart/form-data" ACTION="upload.php" METHOD=POST>
```

Выберите файлы:

```
<p><INPUT NAME="upfile1" TYPE="file">
```

```
<p><INPUT NAME="upfile2" TYPE="file">
```

```
<p><INPUT NAME="upfile3" TYPE="file">
```

```
<INPUT TYPE="submit" VALUE="Загрузить">
```

```
</FORM>
```

ПРИМЕЧАНИЕ

Код этой формы вы найдете в файле *29-4.html* каталога *Glava_29* сопровождающей книгу электронного архива (см. приложение 4).

При этом массив `$_FILES` будет выглядеть так:

- ❑ `$_FILES["upfile1"]["tmp_name"]` — имя первого временного файла;
- ❑ `$_FILES["upfile1"]["name"]` — исходное имя первого файла;
- ❑ `$_FILES["upfile1"]["size"]` — размер первого файла;
- ❑ `$_FILES["upfile1"]["type"]` — MIME-тип первого файла;
- ❑ `$_FILES["upfile1"]["error"]` — код ошибки для первого файла;
- ❑ `$_FILES["upfile2"]["tmp_name"]` — имя второго временного файла;
- ❑ `$_FILES["upfile2"]["name"]` — исходное имя второго файла;
- ❑ `$_FILES["upfile2"]["size"]` — размер второго файла;
- ❑ `$_FILES["upfile2"]["type"]` — MIME-тип второго файла;
- ❑ `$_FILES["upfile2"]["error"]` — код ошибки для второго файла;
- ❑ `$_FILES["upfile3"]["tmp_name"]` — имя третьего временного файла;
- ❑ `$_FILES["upfile3"]["name"]` — исходное имя второго файла;
- ❑ `$_FILES["upfile3"]["size"]` — размер третьего файла;
- ❑ `$_FILES["upfile3"]["type"]` — MIME-тип третьего файла;
- ❑ `$_FILES["upfile3"]["error"]` — код ошибки для третьего файла.

Конечно, в программе придется анализировать, передал пользователь файл или нет. Ведь форма рассчитана на передачу трех файлов, а пользователь мог передать

только два или вообще один файл. Проверить факт передачи файла можно или по имени временного файла, или по коду ошибки (листинг 29.5).

Листинг 29.5. Проверка факта передачи файла

```
...
if(isset($_FILES["upfile1"]))
{
    if ($_FILES["upfile1"]["error"] == 0) {

        // первый файл загружен, копируем
    }

}
...
```

ПРИМЕЧАНИЕ

Код этого фрагмента вы найдете в файле *29-5.txt* каталога *Glava_29* сопровождающего книгу электронного архива (см. приложение 4).

Также нужно решить, под каким именем сохранять файл. Можно использовать имя временного файла, которое автоматически присваивает загруженному файлу интерпретатор PHP. В этом случае имя файла будет иметь примерно такой вид: *php05hhGi*. Первые три символа — это *php*, а остальные будут сгенерированы случайным образом. Вот пример кода, реализующего такую схему именования файлов (пример загрузки фотографий в галерею):

```
if(isset($_FILES["upfile"]))
{
    // имя временного файла
    $upfile = $_FILES["upfile"]["tmp_name"];

    // если ошибок нет
    if($error_code == 0)
    {
        // каталог для загрузки фото
        $d = './fotos/';
        // дополняем имя файла
        $upfile_name = $d . basename($upfile) . ".jpg";

        // перемещаем файл
        move_uploaded_file($upfile, $upfile_name);

        echo "<p>Загруженное фото:";
        echo "<img border=0 width=640
src=$upfile_name>";
    }
}
```


Но, как показывает практика, когда количество картинок большое, имена файлов начинают повторяться намного раньше, чем нам бы этого хотелось. В итоге одна картинка перезаписывает другую, чего делать нельзя. Можно проверять существование временного файла и, если такой файл уже существует, пытаться добавить к имени нового файла какой-то символ, сгенерированный случайным образом, — например, число от 1 до 99. Но это не очень хороший вариант — дополнительные проверки, операторы, все это увеличит время выполнения PHP-сценария, которое и так ограничено. Нужно сразу сгенерировать случайное имя, которое будет уникальным с вероятностью 99 %.

Я предлагаю добавлять к имени временного PHP-файла временную метку (timestamp). В результате получится, что в имени файла скомбинируются случайно сгенерированная последовательность символов (ее генерирует PHP) и временная метка. Вероятность того, что будут сгенерированы две одинаковые последовательности символов в один и тот же момент времени, практически исключена. Вот пример генерирования уникального имени файла:

```
$tm = time();  
$upfile_name = $dir . basename($upfile) . $tm . ".jpg";
```

29.4. Индикатор загрузки файла на PHP 5.4/5.5

29.4.1. Некоторые теоретические предпосылки

В PHP 5.4 появился индикатор загрузки файла, и здесь будет показано, как его использовать. Ранее такой возможности не было и приходилось обращаться к сторонним решениям — например, к модулям `uploadprogress`¹ или `APM`². Задача была не из легких и требовала даже модификации конфигурационного файла Apache (нужно было прописать в нем дополнительные модули). На своем сервере это сделать несложно, а вот уговорить хостинг-провайдера подключить дополнительные модули к своему веб-серверу удавалось не всегда.

С выходом PHP 5.4 ситуация с индикатором загрузки стала лучше. Да, в PHP наконец-то появился механизм, способный контролировать процесс загрузки файлов, и он стал частью механизма сессий. Однако создание рабочего индикатора загрузки (прогресс-бара) все равно остается далеко не легкой задачей, и одного только PHP здесь мало — нужно как минимум подключать к работе и AJAX. Но хоть не нужно изменять конфигурацию веб-сервера — и на том спасибо!

Прежде, чем приступить к реализации индикатора загрузки, рассмотрим табл. 29.2, в которой приведены конфигурационные параметры механизма контроля процесса загрузки.

¹ <http://pecl.php.net/package/uploadprogress>.

² <http://habrahabr.ru/post/17620/>.

Таблица 29.2. Необходимые параметры

Параметр	Описание
<code>session.upload_progress.enabled</code>	Включает/выключает механизм контроля процесса загрузки. Значение по умолчанию: 1 (механизм включен)
<code>session.upload_progress.cleanup</code>	Включает/выключает очистку данных в сессии после загрузки файла. Значение по умолчанию: 1 (очистка включена)
<code>session.upload_progress.prefix</code>	Задаёт префикс переменной в сессии. Значение по умолчанию: <code>upload_progress_</code>
<code>session.upload_progress.name</code>	Название элемента массива <code>\$_POST</code> , в котором передается название переменной сессии, использующейся для хранения информации о загружаемых файлах. Если такой переменной в <code>\$_POST</code> не будет, механизм контроля процесса загрузки окажется недоступен! Значение по умолчанию: <code>PHP_SESSION_UPLOAD_PROGRESS</code>
<code>session.upload_progress.freq</code>	Частота обновления данных в сессии. Можно задать значения в байтах, а можно в процентах от общего размера загружаемого файла. По умолчанию частота обновления: 1 %
<code>session.upload_progress.min_freq</code>	Минимальная задержка в секундах между обновлениями данных. По умолчанию: 1 секунда

Если вы внимательно изучили табл. 29.2, то вам будет понятно, что механизм контроля процесса загрузки файла станет доступен, когда выполняются следующие условия:

- включен параметр `session.upload_progress.enabled`;
- массив `$_POST` содержит элемент с названием `session.upload_progress.name`;
- значение только что упомянутого элемента массива `$_POST` не пустое.

Для выполнения этих условий в текущей сессии создается переменная, название которой формируется из значения параметра `session.upload_progress.prefix` и значения элемента с названием `session.upload_progress.name` массива `$_POST`, а именно:

```
$_SESSION[ini_get(session.upload_progress.prefix).$_POST[ini_get(session.upload_progress.name)]]
```

В эту переменную в виде ассоциативного массива сохраняется информация о процессе загрузки и о самих загружаемых файлах. Структура массива приведена в листинге 29.6.

Листинг 29.6. Структура массива переменной с данными о загружаемых файлах

```
$_SESSION[ini_get(session.upload_progress.prefix).$_POST[ini_get(session.upload_progress.name)]] = array(
    'start_time'           => дата начала загрузки,
    'content_length'       => размер загружаемых данных,
```



```

'bytes_processed'      => кол-во принятых и обработанных байтов,
'done'                 => флаг завершения загрузки (true/false)
// Информация о загружаемых файлах
'files' => array(
    // Информация о первом файле
    0 => array(
        'field_name'      => название поля в форме,
        'name'            => имя файла,
        'tmp_name'         => имя файла во временном каталоге,
        'error'            => код ошибки,
        'done'             => загружен?(true/false),
        'start_time'       => время начала загрузки этого файла,
        'bytes_processed'  => кол-во принятых и обработанных байтов,
    ),
    // Информация о втором файле и т.д.
    1 => array(
        ...
    ),
    n => array(
        ...
    ),
)
);

```

ПРИМЕЧАНИЕ

Код этого листинга вы найдете в файле *29-6.txt* каталога *Glava_29* сопровождающего книгу электронного архива (см. приложение 4).

Представим, что у нас есть форма загрузки двух файлов, код которой приведен в листинге 29.7.

Листинг 29.7. Форма загрузки двух файлов

```

<form action="" method="POST" enctype="multipart/form-data">
    <input type="hidden" name="<?php echo
ini_get("session.upload_progress.name"); ?>" value="pbar" />
    <label for="file1">Файл 1: </label><input type="file" name="file1" />
    <label for="file2">Файл 2: </label><input type="file" name="file2" />
    <input type="submit" value="Загрузить" />
</form>

```

ПРИМЕЧАНИЕ

Код этого листинга вы найдете в файле *29-7.html* каталога *Glava_29* сопровождающего книгу электронного архива (см. приложение 4).

Форма как форма. Очень похожа на обычную форму загрузки файлов, но обратите внимание на скрытое поле, имя которого формирует PHP-оператор `echo ini_get("session.upload_progress.name")`.

Согласно нашей форме при загрузке файлов в сессии будет сохранена следующая информация:

```
$_SESSION['upload_progress_test'] = array(5) {
  "start_time" => 1391929280,
  "content_length" => 4294304,
  "bytes_processed" => 4294304,
  "done" => true,
  "files" => array(2) {
    [0] => array(7) {
      "field_name" => "file1",
      "name" => "DSC_0101.JPG",
      "tmp_name" => "/tmp/phpky7Mqc",
      "error" => 0,
      "done" => true,
      "start_time" => 1391929280,
      "bytes_processed" => 2097152
    },
    [1] => array(7) {
      "field_name" => "file2",
      "name" => "DSC_0102.JPG",
      "tmp_name" => "/tmp/phpDZvS31",
      "error" => 0,
      "done" => true,
      "start_time" => 1391929280,
      "bytes_processed" => 2197152
    },
  }
}
```

Из этого массива становится понятным, что загружалось два (количество элементов в массиве `files`) файла (судя по именам — фотографии) общим размером чуть больше 4 Мбайт, оба файла успешно загружены, о чем свидетельствует общий флаг `done` и флаги `done` каждого из файлов. Прежде, чем перейти к практической части, нельзя не упомянуть следующую переменную:

```
$_SESSION['upload_progress_test']['cancel_upload']
```

Если ей присвоить `true`, процесс загрузки файлов будет прерван. Эту возможность можно использовать для отмены загрузки файлов.

29.4.2. Пример практической реализации

Как уже было упомянуто мною ранее, к сожалению, реализовать индикатор загрузки только средствами PHP не получится, поэтому мы воспользуемся плагином `form`¹

¹<http://jquery.malsup.com/form>.

для библиотеки jQuery¹. Да, без AJAX и JavaScript не обойдется, поэтому я надеюсь, что у читателя есть хотя бы небольшой опыт программирования на JavaScript. Что же касается AJAX, то если вы никогда не работали с этой технологией и не знаете, для чего она применяется, тогда настоятельно рекомендую прямо сейчас обратиться к *главе 35*, в которой описаны ее основы. Конечно, AJAX и jQuery, прочитав *главу 35*, вы не изучите, но хотя бы начнете понимать, что происходит далее в этой главе.

Кроме плагина form и библиотеки jQuery нам также понадобится библиотека jQuery-UI². Что касается jQuery, то я выбрал версию 1.x, поскольку она лучше работает с текущей версией jQuery UI, а также поддерживает даже старые браузеры, — например, IE версий 6/7/8, поддержка которых удалена из ветки 2.x.

Для тестирования и отладки сценариев использовался локальный веб-сервер, созданный с помощью XAMPP (версия PHP 5.4.19). От версии 5.5 я здесь сознательно отказался, поскольку хотелось проверить работу этого функционала именно в версии 5.4 (в которой он только появился). На хостинге конфигурацию проверить не удалось, поскольку хостинг-провайдер до сих пор использует версию 5.3, что делает невозможным включение механизма контроля процесса загрузки файлов.

Создайте на своем сервере каталог upload (в корневом каталоге документов, в случае с XAMPP — это каталог htdocs). В какой-нибудь другой каталог распакуйте jQuery, form и jQuery-UI. В каталог upload нужно поместить следующие файлы:

- ☐ jquery-1.x.x.min.js — библиотека jQuery;
- ☐ jquery-ui-1.x.x.custom.min.js — библиотека jQuery UI;
- ☐ jquery-ui-1.8.14.custom.css — таблица стилей jQuery-UI;
- ☐ jquery.form.js — плагин form.

Далее в каталоге upload создайте подкаталог images. В него поместите GIF-файл, который будет использоваться для заполнения индикатора загрузки. Для тестирования можете взять пока вот этот: <http://dkws.org.ua/files/progress.gif>.

Наш сценарий состоит из трех следующих файлов. Основной файл — index.php — выводит форму загрузки и отображает процесс загрузки. Ход загрузки файла будет получен от сценария progress.php. Третий файл — cancel.php — отменяет загрузку файлов.

Этот сценарий будет загружать файлы только во временный каталог и отображать ход загрузки. Вам нужно будет доработать его и скопировать все загруженные файлы в нужный каталог. Это можно сделать с помощью функции `copy()`, как было показано ранее в этой главе.

Большая часть файла index.php (листинг 29.8) состоит из JavaScript и HTML-кода. PHP-кода там совсем мало. Перед использованием сценария убедитесь, что вы прописали ваши номера версий jQuery и jQuery-UI.

¹ <http://jquery.com/>.

² <http://blog.jqueryui.com/2011/06/jquery-ui-1-8-14/>.

Листинг 29.8. Файл index.php

```
<?php
// Вот и весь PHP-код. Просто запускаем сессию
session_start();
?>
<html>
<head>
<script type="text/javascript" src="jquery-1.x.x.min.js"></script>
<script type="text/javascript" src="jquery-ui-1.x.x.custom.min.js"></script>
<script type="text/javascript" src="jquery.form.js"></script>
<link href="jquery-ui-1.8.14.custom.css" rel="stylesheet" type="text/css" />
<link href="pbar.css" rel="stylesheet" type="text/css" />
<script>
var t;
progress = function(){
    $.ajax({
        url: 'progress.php',
        dataType: 'json',
        success: function(data){
            if(data.percent) {
                $("#bar").progressbar({
                    value: Math.ceil(data.percent),
                });
                $('.ui-progressbar-value').text(data.percent+'%');
            }
        }
    });
}

$(document).ready(function() {
    $('#form').ajaxForm(
    {
        type: 'POST',
        success: function() {
            clearTimeout(t);
            $('#progress').html('<b>Готово!</b>');
        },
        beforeSend: function() {
            $('#upload_form').hide();
            $('#progress').show();
            // Интервал 100 мс
            t = setInterval("progress()", 100);
        }
    });
    $('#cancel-form').ajaxForm(
    {
        success: function() {
            clearTimeout(t);
        }
    });
});
```



```

        $('#progress').html('<b>Вы отменили загрузку!</b>');
    }
}
);
});
</script>
</head>
<body>
<div id="upload">
<div id="upload_form">
<form id="form" action="progress.php" method="POST" enctype="multipart/form-
data">
<input type="hidden" name="<?php echo ini_get("session.upload_progress.name");
?>" value="pbar" />
<label for="file1">Файл 1:&nbsp;</label>
<input type="file" name="file1" /><br /><br />
<label for="file2">Файл 2:&nbsp;</label>
<input type="file" name="file2" /><br /><br />
<input type="submit" value="Загрузить" />
</form>
</div>

<div id="progress">
Загрузка файлов<br /><br /><div id="bar"></div><br />
<form id="cancel-form" action="cancel.php" method="POST"
enctype="multipart/form-data">
<input type="submit" value="Отмена" />
</form>
</div>
<div id="result"></div>
</div>
</body>
</html>

```

ПРИМЕЧАНИЕ

Код этого сценария вы найдете в файле *.29-8.php* каталога *Glava_29* сопровождающего книгу электронного архива (см. приложение 4).

Теперь разберемся, что здесь что. Самая главная функция — функция `progress`. Она вызывает файл `progress.php` и получает от него информацию о ходе загрузки файла. Данные отправляются в формате JSON.

После JavaScript-кода начинается HTML-код, в котором вы видите две формы: одна форма загрузки двух файлов, вторая — форма отмены загрузки. Также есть три компонента `<DIV>`: `upload_form` (форма загрузки), `progress` (будет содержать индикатор загрузки) и `result` (результат загрузки). Код файла `progress.php` представлен в листинге 29.9.

Листинг 29.9. Файл progress.php

```
<?php
session_start();
$p = 0; // Процент загрузки файлов
$data = array();

if(isset($_SESSION['upload_progress_pbar']) and
is_array($_SESSION['upload_progress_pbar'])) {
// Вычисляем процент загрузки
$p = ( $_SESSION['upload_progress_pbar']['bytes_processed'] * 100 ) /
$_SESSION['upload_progress_pbar']['content_length'];

$p = round($p,2); // Округляем процент

// Формируем массив данных
$data = array(
    'percent' => $p,
    'content_length' => $_SESSION['upload_progress_pbar']['content_length'],
    'bytes_processed' => $_SESSION['upload_progress_pbar']['bytes_processed']);
}

// Отправляем данные в JSON-формате
echo json_encode($data);
?>
```

ПРИМЕЧАНИЕ

Код этого листинга вы найдете в файле *29-9.php* каталога *Glava_29* сопровождающего книгу электронного архива (см. приложение 4).

Думаю, тут все понятно. Сценарий просто получает данные о загрузке из сессии, вычисляет процент загрузки и передает эту информацию в JSON-формате вызывающему JavaScript-сценарию.

Код файла *cancel.php* приведен в листинге 29.10. Этот файл — самый простой.

Листинг 29.10. Файл cancel.php

```
<?php
session_start();
$_SESSION['upload_progress_pbar']['cancel_upload'] = true;
?>
```

ПРИМЕЧАНИЕ

Код этого листинга вы найдете в файле *29-10.php* каталога *Glava_29* сопровождающего книгу электронного архива (см. приложение 4).

Осталось рассмотреть еще один файл — таблицу стилей *pbar.css*, определяющую внешний вид индикатора загрузки (листинг 29.11).

Листинг 29.11. Таблица стилей pbar.css

```
.ui-progressbar-value {  
    background-image:  
    url(images/progress.gif);  
    padding-left:10px;  
    font-weight:normal;  
}  
  
#upload_form {  
    display:block;  
}  
  
#progress {  
    display: none;  
}  
  
#progress #bar {  
    height: 25px;  
    width: 350px;  
}
```

ПРИМЕЧАНИЕ

Код этой таблицы вы найдете в файле 29-11.css каталога Glava_29 сопровождающего книгу электронного архива (см. приложение 4).

Теперь посмотрим, как все это работает. Откройте браузер, запустите сценарий index.php, выберите файлы и нажмите кнопку **Загрузить** (рис. 29.3) — вы увидите индикатор загрузки файла, вот только приготовьтесь к тому, что при загрузке небольших файлов на локальный сервер все происходит достаточно быстро, настолько быстро, что этот индикатор вы можете и не заметить. На рис. 29.4 изображен индикатор загрузки файла в процессе работы.

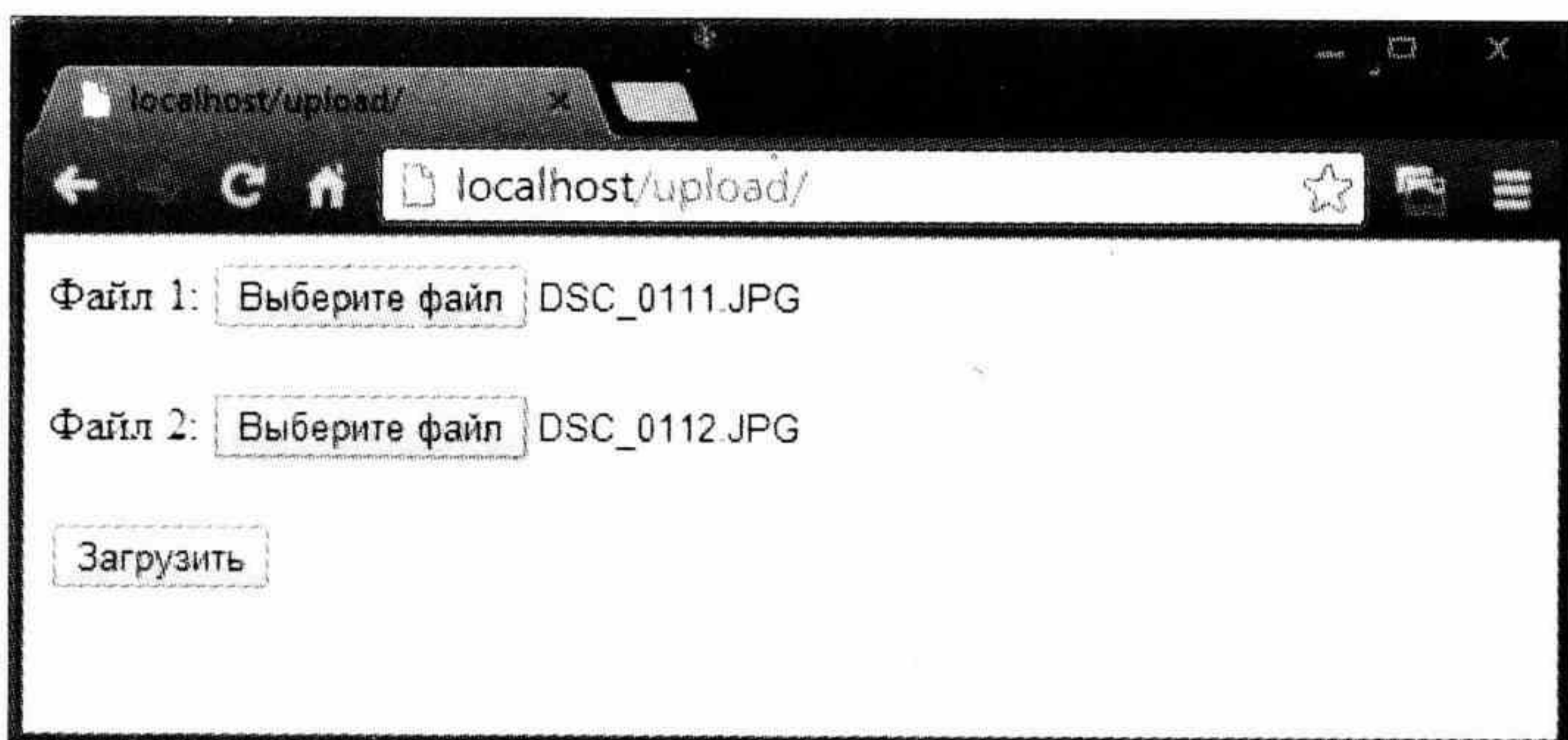


Рис. 29.3. Форма загрузки файлов

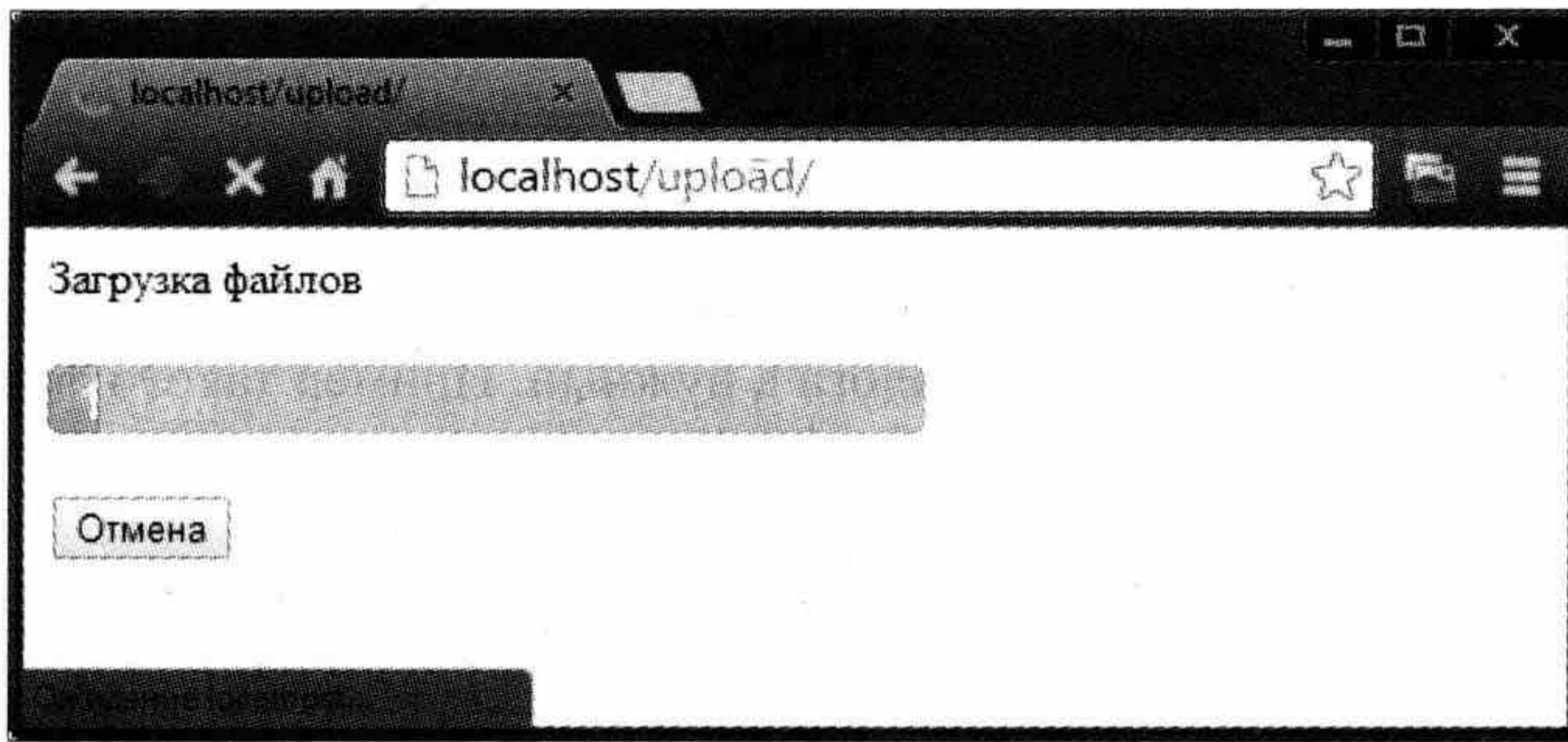


Рис. 29.4. Индикатор процесса загрузки файлов

Для удобства отладки давайте произведем некоторые изменения в нашей программе (чтобы вы убедились, что все работает). Для начала в файле `pbac.css` изменим описание стиля `#progress` так:

```
#progress {
}
```

Другими словами, мы не будем скрывать форму загрузки. Далее в файле `index.php` закомментируйте строчку `$('#upload_form').hide();`:

```
//$('#upload_form').hide();
```

Это даст возможность не скрывать форму загрузки в процессе. Далее заменим строку:

```
$('#progress').html('<b>Готово!</b>');
```

следующей строкой:

```
$('#result').html('<b>Готово!</b>');
```

Исходная строка перезаписывала область индикатора процесса загрузки по окончании процесса загрузки, новая строка перенаправляет вывод строки **Готово** в область результата, не перезаписывая индикатора загрузки. В результате страничка загрузки файла будет выглядеть так, как показано на рис. 29.5. На ней видны все три области: форма загрузки, индикатор загрузки и область результата.

Теперь расскажу, как организовать копирование файлов в нужный вам каталог (сейчас они загружаются во временный каталог и удаляются по завершении работы сценария).

Файл `progress.php` должен использоваться только для отображения процесса загрузки и должен быть определен только в JavaScript-функции:

```
progress = function(){
    $.ajax({
        url: 'progress.php',
```

В форме загрузки вместо файла `progress.php` следует использовать другой файл — например, `upload.php`:


```
<form id="form" action="upload.php" method="POST" enctype="multipart/form-
data">
<input type="hidden" name="<?php echo ini_get("session.upload_progress.name");
?>" value="pbar" />
```

Далее — дело техники. Вам нужно написать сценарий `upload.php`, копирующий загруженные файлы из временного каталога в нужный. Пример такого файла уже был приведен ранее в этой главе (см. листинг 29.3).

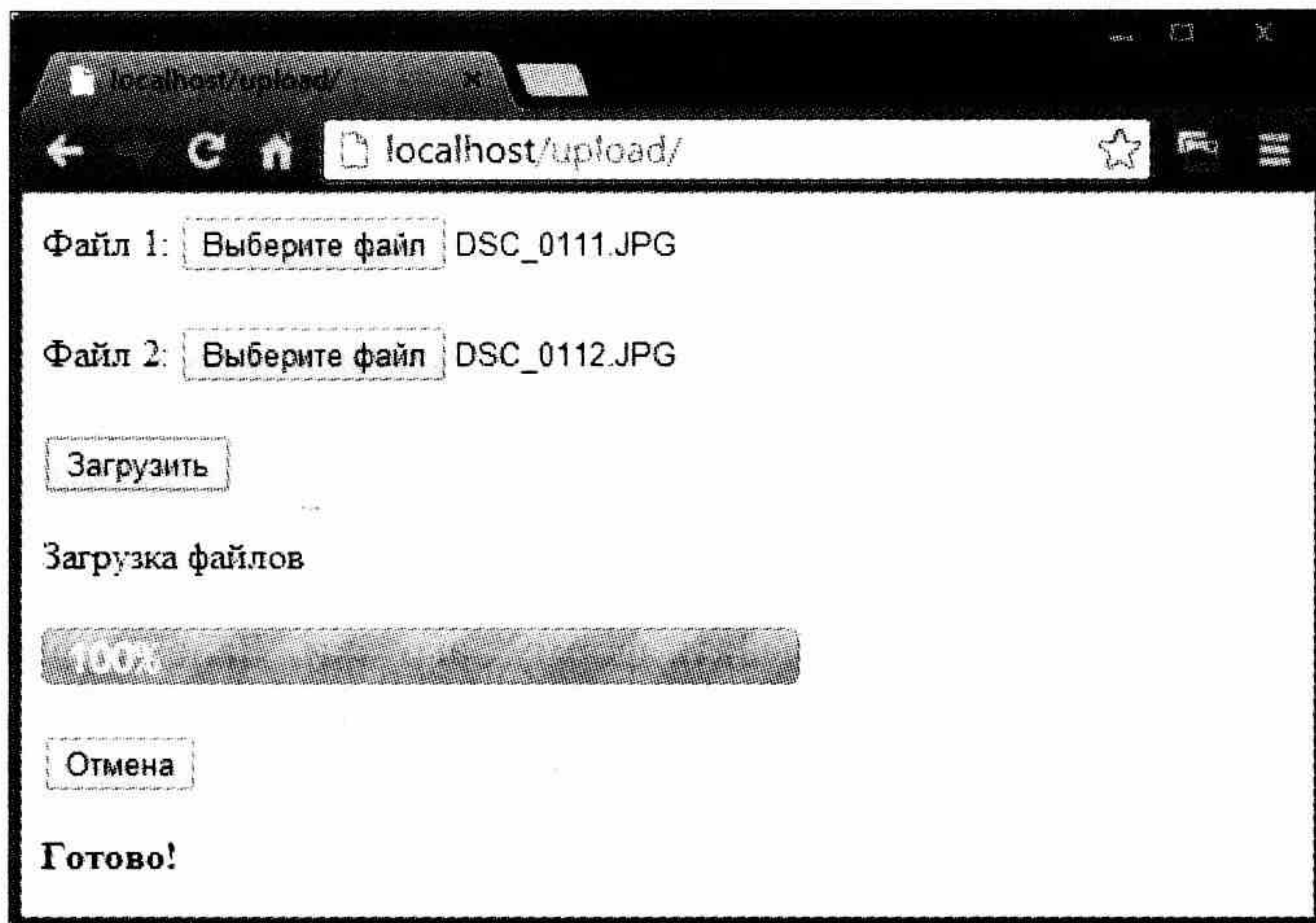


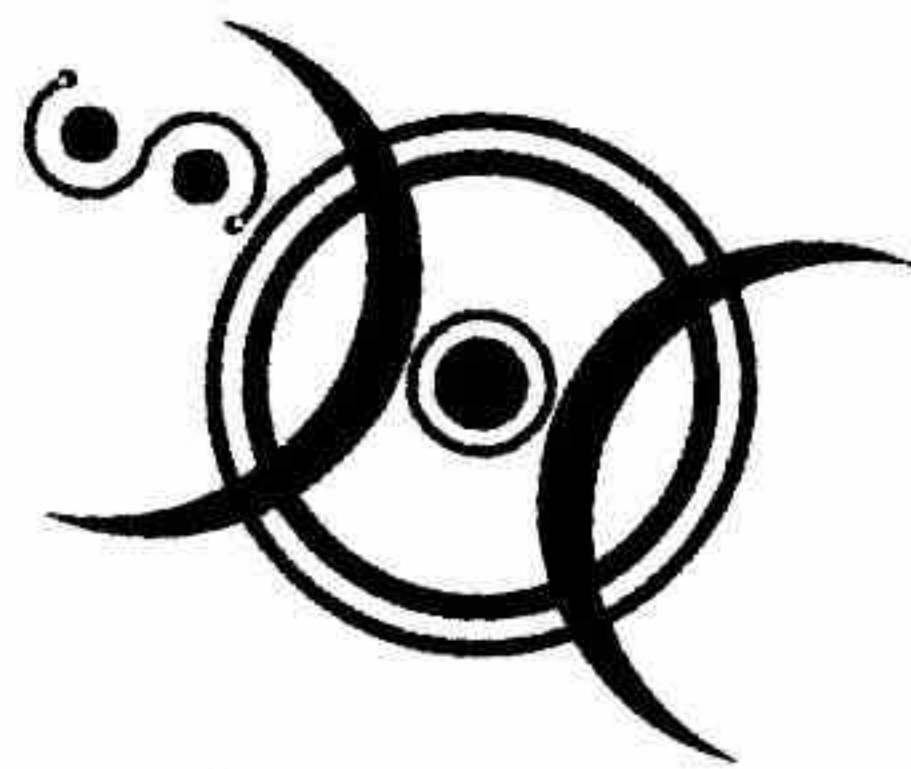
Рис. 29.5. Страница загрузки файлов (режим отладки)

29.5. Проблемы при загрузке файлов

Если вы сделали все, как здесь было описано, трудностей с загрузкой файлов у вас быть не должно. Но все же могут возникнуть две проблемы:

- ❑ файл загружается на сервер, его можно прочитать (ведь у нас есть имя временного файла) и вывести в браузер, но невозможно скопировать в заданный каталог. Скорее всего, это проблема с правами доступа — нужно просто установить права доступа 777 для каталога, в который вы хотите скопировать файл. Это можно сделать с помощью любого FTP-клиента или файлового менеджера панели управления хостингом;
- ❑ файл загружается поврежденным — в этом случае ошибка связана с тем, что Apache пытается перекодировать бинарные файлы, — например, символ с кодом 0x00 он заменяет на символ 0x20 (пробел). Для решения этой проблемы нужно выключить параметр `CharsetRecodeMultipartForms` в файле конфигурации Apache `httpd.conf`:

```
CharsetRecodeMultipartForms Off
```

ГЛАВА 30

Использование FTP-функций

30.1. Функции для работы с FTP

Непонятно почему, но большинство программистов предпочитают использовать для работы с FTP-сервером сокеты, хотя PHP обладает встроенными функциями, позволяющими работать с FTP. Может быть, причина в том, что немногие знают об этих функциях, поскольку во многих самоучителях они не упомянуты, зато описаны сокеты. Возможно, есть и какая-то другая причина такого положения. Если вы также никогда не слышали о FTP-функциях, это легко поправить.

Для подключения к FTP-серверу предусмотрена функция `ftp_connect()`:

```
resource ftp_connect (string host [, int port [, int timeout]])
```

Параметр `host` задает имя сервера, остальные два параметра можно не указывать. Второй параметр — это номер порта, по умолчанию установлено значение 21, а третий — тайм-аут в секундах (по умолчанию — 90).

После подключения к FTP-серверу на нем необходимо зарегистрироваться, т. е. передать имя пользователя и пароль. Для этого служит функция `ftp_login()`:

```
bool ftp_login (resource ftp_stream,  
               string username,  
               string password)
```

Этой функции нужно передать три параметра: идентификатор соединения (его возвращает функция `ftp_connect()`), имя пользователя и пароль.

После успешной регистрации можно осуществлять операции над файлами и каталогами.

Начнем с каталогов. Для изменения каталога предназначена функция `ftp_chdir()`:

```
bool ftp_chdir (resource ftp_stream, string directory)
```

Первый параметр задает идентификатор соединения, а второй — нужный нам каталог. Функция возвращает `true`, если каталог изменен успешно, `false` — если перейти в каталог не удалось. Помните, что не в каждый каталог можно перейти. Для перехода в каталог вы должны обладать правами доступа к нему.

Имя текущего каталога можно узнать с помощью функции `ftp_pwd()`:

```
string ftp_pwd (resource ftp_stream)
```

Для создания каталога служит функция `ftp_mkdir()`, которой передается идентификатор соединения и имя каталога:

```
string ftp_mkdir (resource ftp_stream, string directory)
```

Для создания каталога вы должны обладать необходимыми правами доступа. В случае успеха функция возвращает `true`.

Удалить каталог можно функцией `ftp_rmdir()` с такими же параметрами:

```
bool ftp_rmdir (resource ftp_stream, string directory)
```

Для просмотра содержимого каталога предусмотрена функция `ftp_nlist()`:

```
array ftp_nlist (resource ftp_stream, string directory)
```

Эта функция возвращает массив, содержащий имена файлов и каталогов в указанном каталоге.

Теперь перейдем к функциям для работы с файлами удаленного FTP-сервера. Чаще всего вам понадобятся функции `ftp_fget()` и `ftp_fput()`:

```
bool ftp_fget (resource ftp_stream,  
               resource fp,  
               string remote_file,  
               int mode)
```

```
bool ftp_fput (resource ftp_stream,  
               string remote_file,  
               resource fp,  
               int mode)
```

Первая функция загружает файл с сервера, вторая закидывает файл на сервер. Первый параметр обеих функций — это идентификатор соединения.

Последний параметр — это режим передачи файла:

- ☐ `FTP_ASCII` — текстовый;
- ☐ `FTP_BINARY` — двоичный.

Двоичный режим рекомендуется устанавливать даже для передачи текстовых файлов.

Параметр `fp` функции `ftp_fget()` задает указатель файла, в который должен записаться принимаемый файл. Файловый указатель возвращается функцией `fopen()` при открытии/создании файла. Параметр `remote_file` задает имя файла на удаленном сервере.

Аналогично, параметр `ftp` функции `ftp_fput()` задает указатель файла, который будет прочитан и передан на сервер. Имя файла, в который будут записаны прочитанные из `fp` данные, определяется параметром `remote_file` функции `ftp_put()`.

Функции `ftp_fget()` и `ftp_fput()`, конечно, хороши, но несколько неудобны. Ведь нужно открывать файл, а перед этим проверять его существование, передавать

файловый указатель... Намного проще указать имена локального и удаленного файлов. Это позволяют создать функции `ftp_get()` и `ftp_put()`, существенно упрощающие обмен файлами между компьютерами по FTP.

```
bool ftp_get (resource ftp_stream,
             string local_file,
             string remote_file,
             int mode)

bool ftp_put (resource ftp_stream,
             string remote_file,
             string local_file,
             int mode)
```

Тут все просто: первая функция получает файл с FTP-сервера, а вторая — закачивает файл на FTP-сервер. Параметр `local_file` задает имя локального файла, а `remote_file` — удаленного. Параметр `mode` определяет режим передачи файла. Как видите, все намного проще — нам не нужны больше файловые указатели.

Для переименования файла на FTP-сервере служит функция `ftp_rename()`:

```
bool ftp_rename (resource ftp_stream, string from, string to)
```

Параметр `from` задает старое имя файла, а `to` — новое.

Чтобы удалить файл, нужно вызвать функцию `ftp_delete()`, которой следует передать идентификатор FTP-соединения и имя файла:

```
bool ftp_delete (resource ftp_stream, string path)
```

Функция `file_size()` возвращает размер файла на FTP-сервере (полезно узнать размер перед скачиванием):

```
int ftp_size (resource ftp_stream, string remote_file)
```

Мы рассмотрели практически все необходимые FTP-функции. Упомянем еще три. Довольно полезной является функция `ftp_exec()`, позволяющая выполнять команды на FTP-сервере:

```
bool ftp_exec (resource ftp_stream, string command)
```

Функция возвращает `true`, если команду удалось выполнить:

```
if (ftp_exec($conn, 'chmod 777 images'))
    echo "Права доступа изменены успешно";
else
    echo "Ошибка при изменении прав доступа";
```

Протокол FTP предусматривает два режима работы: активный и пассивный. Разница между ними заключается в том, что в пассивном режиме клиент всегда инициирует соединение, а в активном инициатором соединения может быть FTP-сервер.

В активном режиме FTP-соединение выглядит так: клиент подключается к серверу и передает ему имя пользователя, пароль и номер порта (из динамического диапазона 1024–65 535). FTP-сервер подключается к указанному клиентом порту, а со своей стороны он использует TCP-порт 20.

В пассивном режиме сервер назначает номер порта (из диапазона 1024–65 535), к которому должен подключиться клиент для передачи данных. Таким образом, не сервер подключается к клиенту, а клиент к серверу. Пассивный режим полезен, когда брандмауэр блокирует входящие соединения.

Для перехода в пассивный режим служит функция `ftp_pasv()`:

```
bool ftp_pasv (resource ftp_stream, bool pasv)
```

Этой функции нужно передать идентификатор соединения и параметр `pasv`, определяющий режим работы FTP:

❑ `true` — пассивный;

❑ `false` — активный.

Осталось рассмотреть еще одну функцию — `ftp_close()`, которая закрывает FTP-соединение. Вообще, FTP-соединение будет закрыто автоматически при завершении работы сценария, но вызова этой функции требует культура программирования на PHP:

```
void ftp_close (resource ftp_stream)
```

При работе с FTP помните о максимальном времени выполнения сценария, которое задается в файле `php.ini`. Предположим, что это время равно 30 с (по умолчанию), а файл, который нужно передать, "весит" 1 Мбайт. Следовательно, между вами и FTP-сервером должно быть соединение со скоростью не менее 35 Кбайт/с, чтобы уложиться в отведенные 30 с. А учитывая то, что на установку соединения с сервером также требуется время, то скорость передачи данных должна быть не менее 36–40 Кбайт/с.

30.2. Примеры использования FTP-функций

Рассмотрим два примера. Первый пример (листинг 30.1) будет подключаться к FTP-серверу, регистрироваться и выводить содержимое текущего каталога (рис. 30.1). Второй пример будет загружать файл с сервера (листинг 30.2).

Листинг 30.1. Вывод содержимого каталога

```
<?php

// устанавливаем соединение с сервером ftp.server.ru
$conn = ftp_connect("ftp.server.ru");

// передаем имя пользователя и пароль
$login = ftp_login($conn, "user", "dfh45h2");

// подключаемся к серверу
if ((!$conn) || (!$login))
    die("Ошибка подключения к FTP-серверу");
```



```
else
    echo "Успешно подключились к серверу!";

echo "<p>Выводим содержимое текущего каталога";
// получаем список файлов и каталогов FTP-сервера
$A = ftp_nlist($conn, ftp_pwd($conn));

// выводим список файлов и каталогов
foreach($A as $a) echo "<br>$a";

// отключаемся
ftp_close($conn);

?>
```

Листинг 30.2 иллюстрирует пример загрузки файла с сервера.

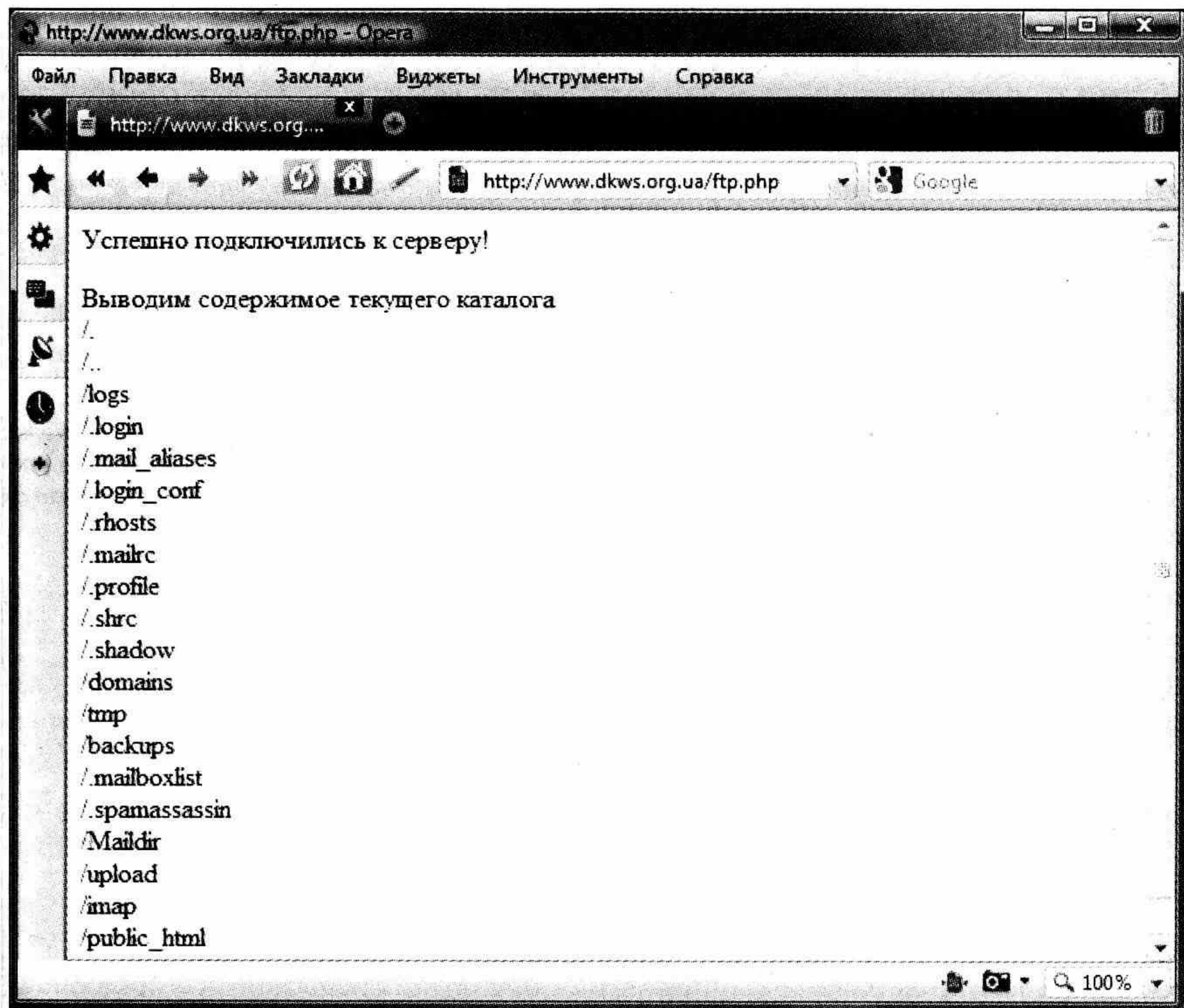


Рис. 30.1. Результат работы сценария (листинг 30.1)

Листинг 30.2. Пример загрузки файла с FTP-сервера

```
<?php

// устанавливаем соединение с сервером ftp.server.ru
$conn = ftp_connect("ftp.server.ru");

// передаем имя пользователя и пароль
$login = ftp_login($conn, "anonymous", "user@mail.ru");

// подключаемся к серверу
if ((!$conn) || (!$login))
    die("Ошибка подключения к FTP-серверу");
else
    echo "Успешно подключились к серверу!";

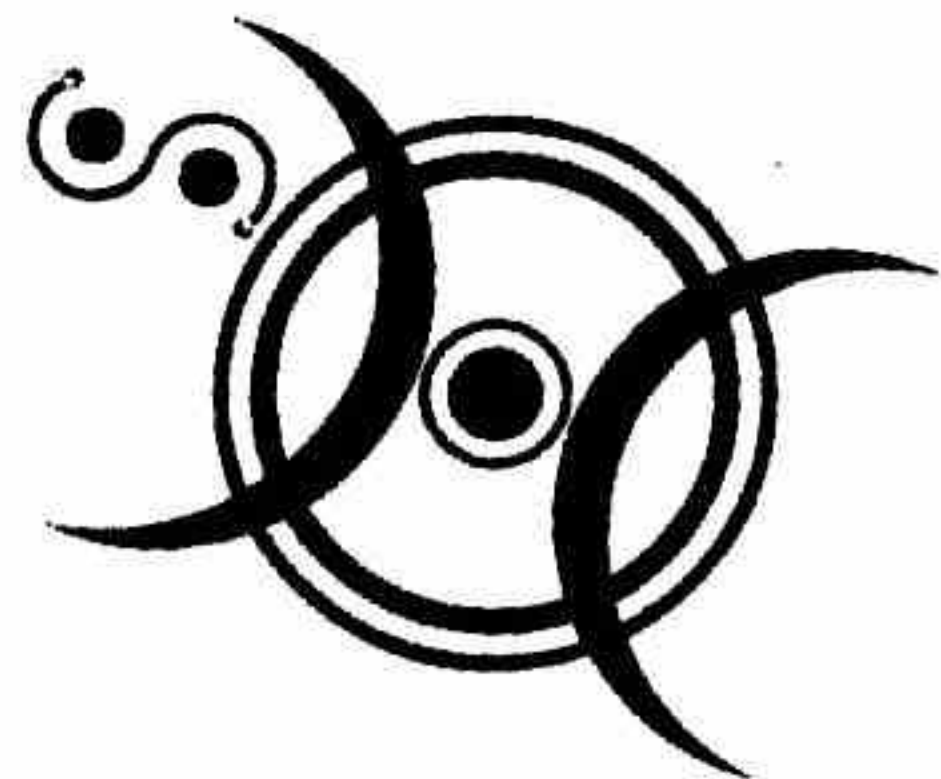
// загружаем удаленный файл ftp://ftp.server.ru/pub/soft.rar
if (ftp_get($conn, "soft.rar", "/pub/soft.rar", FTP_BINARY))
    echo "Файл загружен успешно";
else
    echo "Произошла ошибка";

// отключаемся
ftp_close($conn);

?>
```

ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода листингов 30.1 и 30.2, настоятельно рекомендую использовать уже готовые варианты (файлы *30-1.php* и *30-2.php* соответственно), представленные в каталоге *Glava_30* сопровождающего книгу электронного архива (см. приложение 4).



ГЛАВА 31

Отправка и прием почты

31.1. Отправка почты средствами PHP — функция *mail()*

Отослать подтверждение заказа, подтвердить забытый пароль, задать вопрос администрации сайта или поздравить пользователя с днем рождения — для всего этого (и не только) потребуется функция отправки электронных сообщений. В PHP для этой цели есть простая и удобная функция `mail()`:

```
mail(string email, string subject, string message [,string headers]);
```

Первый параметр функции — это электронный адрес получателя, второй — тема сообщения, третий — само сообщение, а четвертый — заголовки сообщения. По умолчанию заголовки указывать не обязательно, но иногда без них не обойтись. Вот самые полезные заголовки:

- ❑ `From` — содержит адрес отправителя (нужно же получателю знать, от кого он получил письмо?);
- ❑ `Reply-To` — позволяет указать адрес для ответа на сообщение;
- ❑ `Content-Type` — позволяет указать тип содержимого, например `text/plain`;
- ❑ `Content-Transfer-Encoding` — указывает, сколько битов используется для передачи символа. Для отправки сообщений на русском языке необходимо 8 битов.

Наиболее важны заголовки `Content-Type` и `Content-Transfer-Encoding`. Как уже было отмечено, первый позволяет определить тип содержимого, а второй — определить, как будут перекодироваться символы.

Для нормальной отправки сообщений на русском языке в кодировке 1251 нужно передать два следующих заголовка:

```
Content-Type: text/plain; charset=Windows-1251  
Content-Transfer-Encoding: 8bit
```

Можно также использовать механизм конвертации `Quoted-Printable`, гарантирующий целостность вашего содержимого при прохождении через различные почтовые шлюзы, производящие языковую перекодировку символов:


```
Content-Type: text/plain; charset=Windows-1251
```

```
Content-Transfer-Encoding: quoted-printable
```

Пример:

```
mail('user@host.ru', 'Привет', 'Привет! Как дела!' ,  
'Content-Type: text/plain; charset=Windows-1251\n'  
Content-Transfer-Encoding: quoted-printable\n');
```

После каждого заголовка не забываем указывать символ новой строки — `\n`.

Если вы тестируете свой сценарий под Windows, то функция `mail()`, скорее всего, работать у вас не будет. Для того чтобы она все-таки заработала, следует отредактировать файл `php.ini`. Откройте его и найдите секцию `mail function`. Параметру SMTP нужно присвоить имя (или IP-адрес) SMTP-сервера, а параметру `smtp_port` — номер порта почтового сервера (обычно 25):

```
[mail function]  
; For Win32 only.  
SMTP = localhost  
smtp_port = 25
```

С помощью функции `mail()` можно отправлять сообщения в формате HTML, для этого просто нужно указать тип содержимого `Content-Type: text/html`:

```
$mailto = 'user@host.ru';  
$message = '<html><body><h1>Привет, Вася!</h1></body></html>';  
$header = 'Content-Type: text/html; charset=Windows-1251\n';  
$header .= ' Content-Transfer-Encoding: quoted-printable\n';  
  
mail($mailto, $message, $header);
```

Обратите внимание, как присваивается значение второго заголовка! Не потеряйте точку!

Если вникнуть в принципы MIME-кодирования, то с помощью функции `mail()` даже можно отправлять электронные письма с вложениями. Но тратить время и силы на это не хочется, особенно когда существуют готовые решения. Далее мы рассмотрим класс `HtmlMimeMail`, позволяющий отправлять сообщения в формате HTML с вложениями, закодированными с помощью MIME.

31.2. Отправка писем с вложениями — класс *HtmlMimeMail*

31.2.1. Отправка сообщения

Класс `HtmlMimeMail` позволяет облегчить отправку электронных писем с вложениями средствами PHP. Этот класс, а также все остальные классы и листинги из этой главы, можно найти в каталоге `Glava_31` сопровождающего книгу электронного архива (см. приложение 4). Последняя версия класса `HtmlMimeMail` также всегда доступна по адресу: <http://www.phpguru.org/downloads/html.mime.mail/>.

Вместо последней версии `HtmlMimeMail5` — для ее работы нужен PHP 5/6 (архив с `HtmlMimeMail5` вы также найдете в каталоге `Glava_31` сопровождающего книгу электронного архива) — мы будем использовать тщательно проверенную временем и мною версию `HtmlMimeMail` — файл `htmlMimeMail-2.5.2.zip` из каталога `Glava_31` сопровождающего книгу электронного архива. К тому же, эта версия мною немного переработана и усовершенствована. В частности, в ней исправлено перекодирование русских символов. Архив с исправленной версией называется `mime.zip` и также размещен в каталоге `Glava_31` сопровождающего книгу электронного архива. В файле `mime.zip` вы найдете и несколько вспомогательных файлов для демонстрации примера:

- ❑ `message.html` — текст сообщения в формате HTML;
- ❑ `message.txt` — текст сообщения (`text/plain`);
- ❑ `bg.gif` — фоновая картинка для HTML;
- ❑ `empty.zip` — пустой архив (для демонстрации вложения файлов);
- ❑ `test.php` — пример, демонстрирующий отправку сообщения в формате HTML с вложением `empty.zip`.

Рассмотрим код файла `test.php` (листинг 31.1).

Листинг 31.1. Отправка сообщения (`test.php`)

```
<?
// на время отладки показываем все ошибки
error_reporting(E_ALL);

// подключаем класс отправки сообщения
include "htmlMimeMail.php";

// создаем экземпляр класса
$mail = new htmlMimeMail();

// читаем картинку (будет использоваться в качестве фона)
$image = $mail->getFile('bg.gif');

// читаем файл, который нужно "прикрепить к письму"
$attach = $mail->getFile('empty.zip');

// читаем обычный текст
$text = $mail->getFile('message.txt');

// читаем текст в формате HTML
$html = $mail->getFile('message.html');

// устанавливаем текст в формате HTML и для почтовых клиентов,
// не поддерживающих HTML, указываем обычный текст
$mail->setHtml($html, $text);
```



```
// связываем картинку с HTML
// правильно указывайте тип картинки:
// image/gif — для GIF, image/jpeg — для JPEG
$mail->addHtmlImage($image, 'bg.gif', 'image/gif');

// добавляем вложение. Обратите внимание на тип файла:
// если тип будет другой, его нужно будет подкорректировать,
// например: image/gif для GIF-картинки, image/jpeg — для JPEG-картинки
$mail->addAttachment($attach, 'empty.zip', 'application/zip');

// адрес отправителя
// нужно указать правильный адрес, иначе вы получите сообщение об ошибке
// и письмо не будет отправлено — многие почтовые серверы проверяют
// правильность адресов отправителя и получателя
$mail->setReturnPath('user@romb.net');

// заголовок "От" (содержит адрес отправителя)
$mail->setFrom('user@romb.net');

// тема сообщения
$mail->setSubject('Привет');

// отправляем письмо адресату user2@romb.net
$result = $mail->send(array('user2@romb.net'), 'smtp');

// если нет ошибок, выводим сообщение "Отправлено"
if (!$result) echo $mail->errors;
else
echo 'Отправлено';

?>
```

Разберемся, какие здесь используются методы класса `HtmlMimeMail`. Я специально привел пример отправки сообщения в формате HTML с вложением, чтобы сразу продемонстрировать все возможности класса.

Метод `getFile()` необходим для чтения любого файла, будь то файл сообщения или файл вложения.

Метод `setHtml()` служит для установки HTML-текста сообщения. Если вам нужно отправить обычное текстовое сообщение (без HTML), то следует указать метод `setText()`:

```
$mail->setText($text);
```

Метод `addHtmlImage()` предназначен для связки картинки с HTML, т. е. с помощью этого метода вы указываете, что данная картинка относится к HTML, а не является просто вложением.

С помощью метода `addAttachment()` можно добавить к письму вложение. Первый параметр — это файл, прочитанный методом `getFile()`, второй — имя файла, которое будет отображаться в почтовом клиенте, а третий — MIME-тип файла. MIME-тип используется для правильной обработки вложения почтовым клиентом. О MIME-типах мы поговорим отдельно.

Адрес отправителя можно задать или с помощью метода `setReturnPath()`, или методом `setFrom()`. Если вызваны оба метода, то первый имеет больший приоритет, чем второй. Моя показательная ошибка, связанная с использованием этого класса, как раз касается первого метода — я указал неправильный адрес отправителя (не исправил демонстрационное значение), а сервер не отправлял сообщение, хотя поле `From` было указано правильно (с помощью метода `setFrom()`).

Метод `setSubject()` служит для установки темы сообщения.

А саму отправку сообщения обеспечивает метод `send()`. Первый параметр этого метода задает список получателей письма (в нашем случае получатель всего один), а второй параметр указывает способ отправки. По умолчанию (если второй параметр не указывать) будет вызываться функция `mail()`, но, как уже было отмечено, она не всегда работает. Поэтому для большей универсальности поддерживается отправка с использованием протокола SMTP (значение второго параметра `'smtp'`).

Запустите тестовый сценарий в окне браузера. Вы увидите предупреждения PHP (рис. 31.1), но пока вы не протестируете свой сценарий, не отключайте функцию `error_reporting()`.

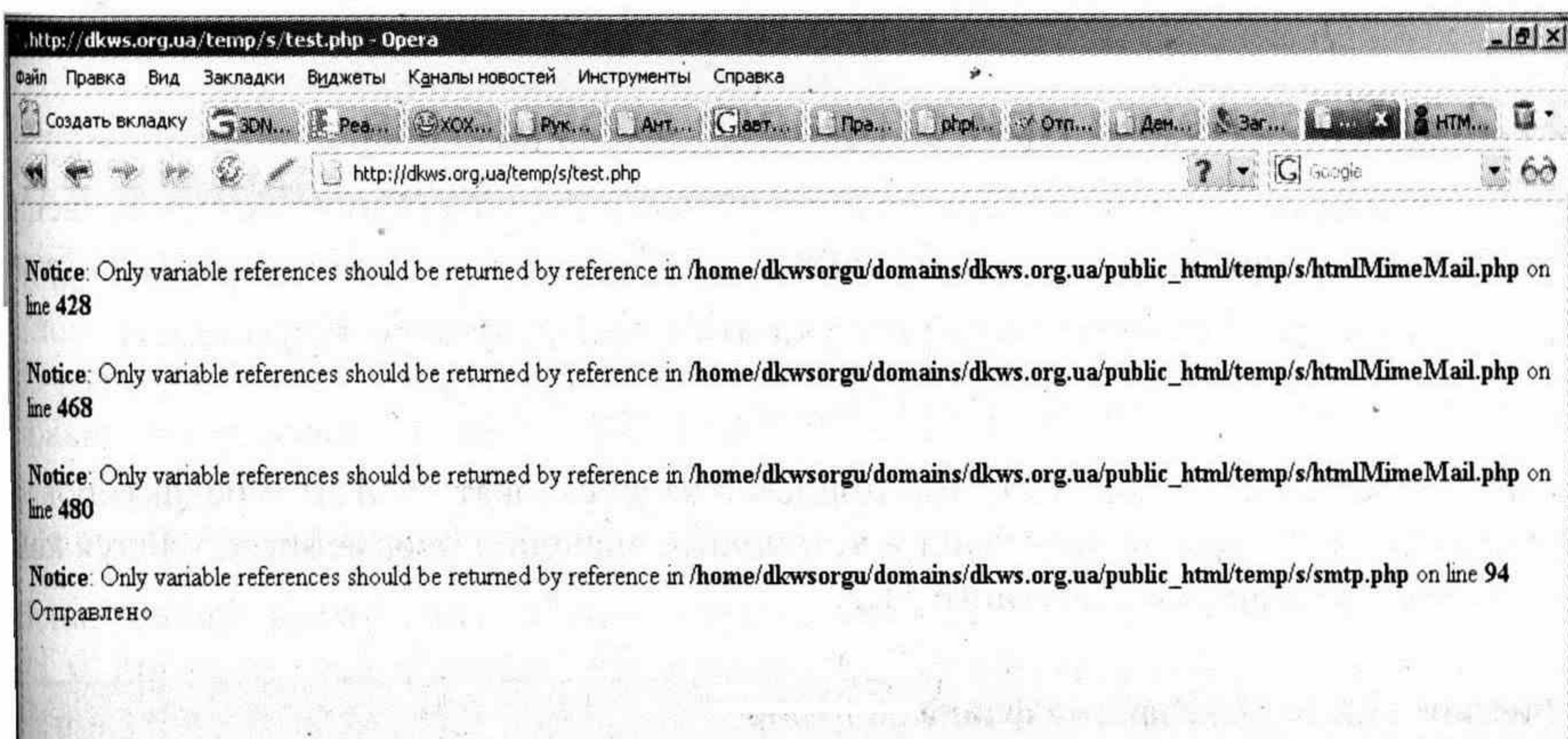


Рис. 31.1. Отправка сообщения

Если вы все сделали правильно, то через пару секунд вы должны получить подтверждение, что сообщение отправлено (рис. 31.2). Как видите, в окне почтового агента The Bat! наше сообщение отображается корректно.

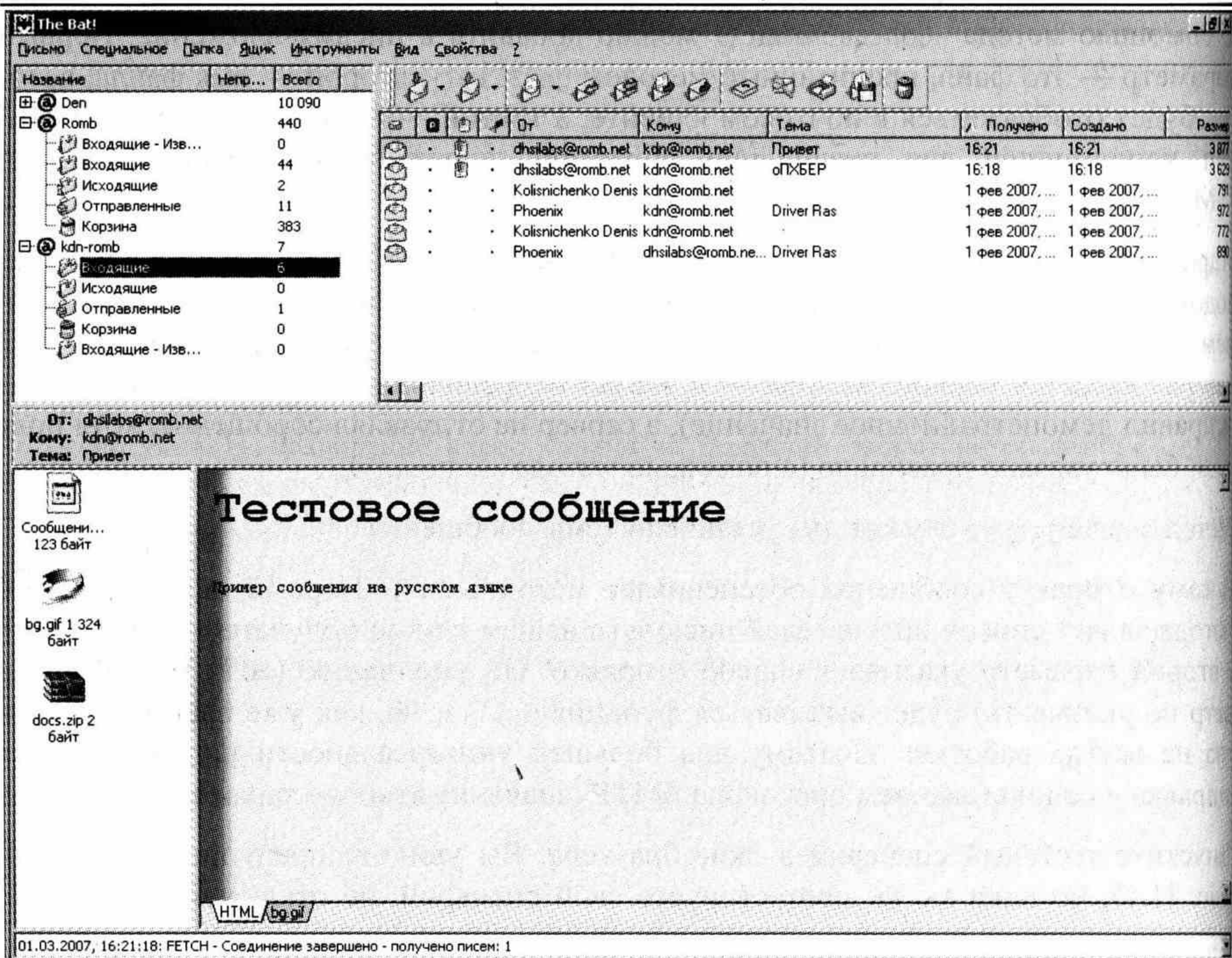


Рис. 31.2. Отправленное сообщение

31.2.2. Проблемы при отправке сообщения

При отправке сообщения вы можете столкнуться с двумя трудностями: собственно с проблемой отправки самого сообщения и с проблемой отображения русских букв.

Начнем с первой. Сначала попробуйте указать стандартную функцию `mail()`:

```
$result = $mail->send(array('user@romb.net'),'mail');
```

Если это решение не поможет, вам придется задействовать SMTP и редактировать файл `smtp.php`. Откройте этот файл и установите значения переменных, следуя комментариям, указанным в листинге 31.2.

Листинг 31.2. Корректировка файла `smtp.php`

```
// измените на TRUE, если требуется аутентификация SMTP
$this->authenticated = FALSE;
// тайм-аут (обычно изменять не нужно)
$this->timeout = 5;
$this->status = SMTP_STATUS_NOT_CONNECTED;
```



```
// имя или IP-адрес SMTP-сервера
$this->host    = 'localhost';
// порт (обычно 25)
$this->port    = 25;
$this->helo    = 'localhost';
// измените на TRUE, если требуется аутентификация SMTP
$this->auth    = FALSE;
// имя пользователя и его пароль (для аутентификации)
$this->user    = '';
$this->pass    = '';
```

Другими словами, в файле `smtp.php` нужно указать параметры SMTP-сервера. Эти параметры вы можете уточнить у своего провайдера.

Теперь поговорим о второй проблеме. Класс из файла `mime.zip` я настроил для работы с кодировкой `windows-1251`. Если у вас установлена другая кодировка, то ваши сообщения будут отображаться неправильно. Для изменения кодировки символов нужно открыть файл `HtmlMimeMail.php` и найти следующие строки:

```
$this->build_params['html_encoding'] = 'quoted-printable';
$this->build_params['text_encoding'] = '8bit';
$this->build_params['html_charset']  = 'windows-1251';
$this->build_params['text_charset']  = 'windows-1251';
$this->build_params['head_charset']  = 'windows-1251';
$this->build_params['text_wrap']     = 998;
```

Измените кодировку `windows-1251` на нужную вам.

31.2.3. MIME-типы

Аббревиатура MIME расшифровывается как **M**ultipurpose **I**nternet **M**ail **E**xtension — многоцелевые расширения почты Интернета. Правда, сегодня MIME применяется не только в системе электронной почты. Прежде MIME-типы предназначались для правильной обработки присоединенных к электронным письмам различных файлов. По MIME-типу почтовый клиент мог определить, какую программу вызвать для просмотра прикрепленного к письму файла.

В настоящее время типы MIME используются не только почтовыми клиентами, но и другими программами для правильного отображения файлов разных форматов.

MIME-тип записывается так:

"базовый тип/подтип"

Базовые типы MIME и часто встречающиеся подтипы представлены в табл. 31.1.

Таблица 31.1. Базовые типы MIME

Базовый тип	Описание	Примеры подтипов
application	Двоичные данные, например application/zip	application/mspowerpoint application/msword application/pdf application/x-bzip2 application/x-excel application/x-gtar application/x-gzip application/zip
audio	Аудиоданные	audio/midi audio/mpeg audio/x-aiff audio/x-wav
image	Изображения	image/gif image/jpeg image/png image/tiff
message	Сообщения электронной почты, внедренные как вложения (сейчас практически этот тип не используется)	message/news message/partial message/rfc822
model	Модели — различные структуры, состоящие из объектов (используется редко)	
multipart	Составные файлы, которые состоят из набора файлов разных типов	multipart/alternative multipart/mixed
text	Текст	text/html
video	Видеофайлы	video/mpeg video/quicktime video/x-msvideo

31.3. Класс *PHPMailer*. Разработка сценария автоматической рассылки прайс-листа

Поддержка классов `HtmlMimeMail` и `HtmlMimeMail5` на данный момент остановлена т. е. разработчики больше не станут выпускать новые версии этих классов. Вместо них рекомендуют использовать класс `Rmail`. С этим классом я не работал — скажу честно — поэтому что-либо сказать (хорошее или плохое) относительно него не могу.

Отсутствие поддержки вовсе не означает, что указанные классы больше не будут работать. PHP-код этих классов абсолютно рабочий. Но исправление возможных

ошибок и добавление новых функций в классы `HtmlMimeMail` и `HtmlMimeMail5` более производиться не будет.

В некоторых своих проектах я с успехом использовал класс `PHPMailer`, который также служит для отправки почты. Последняя на момент написания этих строк версия класса `PHPMailer` (файл `PHPMailer_v5.1.zip`) находится в каталоге `Глава_31` сопровождающего книгу электронного архива. В дальнейшем свежие версии этого класса вы всегда сможете скачать по адресу: <http://phpmailer.worxware.com/>.

Класс `PHPMailer`, как и класс `HtmlMimeMail`, поддерживает SMTP-аутентификацию и отправку сообщений с вложениями. Перед началом работы с `PHPMailer` нужно отредактировать файл `class.phpmailer.php`. Как минимум, следует указать имя SMTP-сервера и его порт:

```
var $Host          = 'smtp.isp.ru';
var $Port          = 25;
```

В некоторых случаях (когда передача данных осуществляется с шифрованием SSL) вместо порта 25 используется порт 465.

Тип шифрования задается переменной `$SMTPSecure`, для SSL и TLS (соответственно) нужно установить ее значение так:

```
var $SMTPSecure = "ssl";
var $SMTPSecure = "tls";
```

Чтобы включить SMTP-аутентификацию, нужно установить следующие переменные так:

```
var $SMTPAuth      = true;
var $Username      = 'имя_пользователя';
var $Password      = 'пароль';
```

Кодировка писем устанавливается так:

```
public $CharSet      = 'windows-1251';
```

Теперь приступим к написанию сценария, использующего `PHPMailer`. Подключаем класс:

```
require_once("class.phpmailer.php");
$mailer = new PHPMailer;
```

Следующий код устанавливает получателя, добавляет вложение и отправляет сообщение:

```
// очищаем список получателей
$mailer->ClearAddresses();
// добавляем получателя, вы можете указать
// несколько получателей
$mailer->AddAddress('root@host.ru');
// адрес отправителя
$mailer->From = 'den@host.ru';
// имя отправителя
$mailer->FromName = 'Home';
```



```
// тема сообщения
$mailer->Subject = 'Привет';
// текст сообщения
$mailer->Body = "Пример отправки сообщения с вложением";
// удаляем все вложения
$mailer->ClearAttachments();
// добавляем вложение. Первый параметр -
// имя файла на диске, второй - имя файла во вложении
$mailer->AddAttachment('mailer/test.zip', 'test.zip');
// отправляем письмо
if ($mailer->Send()) {
    echo "<p>Сообщение успешно отправлено";
} else {
    // выводим сообщение об ошибке
    echo $mailer->ErrorInfo;
}
```

Работать с классом `PHPMailer` так же просто, как и с `HtmlMimeMail`. Какой класс использовать — решать только вам.

Рассмотрим небольшой практический проект. Представим, что отделу сбыта нужно рассылать прайс-лист клиентам вашей фирмы. Нам нужно организовать автоматическую рассылку прайс-листа по указанному списку адресов. Адреса клиентов, как и многую другую информацию, которую можно представить в виде таблицы, отечественные менеджеры почему-то привыкли хранить в формате `Excel`. Не будем создавать систему управления таблицей клиентов на базе `MySQL` и переучивать менеджеров — пусть себе ведут свой список в `Excel`. Нужно только экспортировать его в формат `CSV`, затем прочитать этот файл, выделить из него `e-mail` и отправить каждому клиенту копию прайс-листа.

Чтобы не усложнять наш сценарий, будем считать, что:

- ❑ архив с прайс-листом размещен в каталоге `mail` подкаталога, в котором находится сценарий, и называется он `price.zip`. Вы можете добавить форму загрузки файла и сделать так, чтобы менеджер мог каждый раз загружать прайс по протоколу `HTTP`. Это не сложно — в своем проекте я так и сделал, но чтобы не усложнять код сейчас, возложу добавление такой возможности на ваши плечи. Текущая версия сценария предполагает, что файл с прайсом попал в каталог `mail` до запуска сценария, — например, вы его туда загрузили по `FTP`;
- ❑ текст сообщения хранится в файле `message_body.txt`. Как правило, текст сообщения всегда один, его достаточно один раз записать в такой файл и закачать по `FTP` на сервер;
- ❑ все `e-mail`-адреса в `CSV`-файле (который, кстати, называется `clients.csv`) указаны правильно. При желании можете добавить функцию проверки корректности `e-mail` (см. главу 6);
- ❑ в `CSV`-файле всего два поля: первое — имя получателя (или название фирмы), второе — `e-mail`.

Код нашего сценария представлен в листинге 31.3. Вкратце опишу его работу: сначала сценарий получает из CSV-файла адрес получателя, затем передает его функции `phpmailer()`, которая и осуществляет отправку.

Листинг 31.3. Сценарий `sender.php`

```
<?php

// подключаем PHPMailer
require_once("class.phpmailer.php");
$mailer = new PHPMailer;

$from = 'den@host.ru';
$from_name = 'Home';

// функция отправки сообщения с вложением
function phpmailer($email) {

global $mailer, $from, $from_name;

// получаем текст сообщения
$txt = join('', file('message_body.txt'));

$mailer->ClearAddresses();
$mailer->AddAddress($email);
// адрес отправителя
$mailer->From = $from;
// имя отправителя
$mailer->FromName = $from_name;
// тема сообщения
$mailer->Subject = 'Наш новый прайс-лист';
// текст сообщения
$mailer->Body = $txt;
// удаляем все вложения
$mailer->ClearAttachments();
// добавляем вложение
$mailer->AddAttachment('mailer/price.zip', 'price.zip');
// отправление письма
if ($mailer->Send()) {
echo "<p>Сообщение по адресу $email успешно отправлено <br>";
} else {
// выводим сообщение об ошибке
echo $mailer->ErrorInfo;
}

} // phpmailer()

// открываем CSV-файл
$f = fopen('clients.csv', 'r');
```



```
while($array=fgetcsv($f, 1024, ';')) {  
    // получаем имя получателя и его адрес  
    $name= $array[0];  
    $email = $array[1];  
  
    // отправляем сообщение  
    phpmailer($email);  
  
}
```

?>

Мы создали простой сценарий автоматической отправки прайс-листа, работающий без использования базы данных, что упрощает эксплуатацию сценария.

31.4. Получение писем по протоколу POP3

Для приема электронных сообщений используется протокол POP3 (Post Office Protocol v. 3). Письма поступают на сервер и хранятся там, пока мы их не получим (можно при получении сообщения не удалять, тогда они останутся на сервере и после получения). Если при отправке сообщений еще можно было написать свой универсальный класс, используя сокет, то при приеме сообщения по протоколу POP3 это будет сделать накладно — уж очень много нюансов. Гораздо проще применить готовые решения.

Здесь мы рассмотрим класс POP3 (файл pop3.zip из каталога Glava_31 сопровождающего книгу электронного архива). Как и HtmlMimeMail, класс POP3 берет на себя всю рутинную работу. Нам остается лишь запрограммировать действия POP3-клиента.

Разберемся, как нужно работать с классом POP3. Первым делом следует подключить сам класс:

```
include "pop3.php";
```

Сразу после этого необходимо создать экземпляр класса:

```
$pop3=new pop3_class;
```

Вот теперь можно подключаться к POP3-серверу:

```
// задаем имя POP-сервера  
$pop3->hostname="pop.mail.ru";  
// открываем соединение  
$error_message = $pop3->Open();
```

Метод `Open()`, как и многие другие методы класса POP3, возвращает строку. Если строка пуста, значит, операция выполнена успешно, в противном случае строка содержит сообщение об ошибке. Поэтому сразу после подключения, точнее, вызова метода `Open()`, мы должны проанализировать переменную `$error_message`. Если она не пустая, то выводим ее и завершаем работу сценария:

```
if ($error_message != "") die($error_message);
```


Если мы успешно подключились к серверу, нам потребуется сообщить ему имя пользователя и пароль, а также указать, будет ли использоваться метод аутентификации APOP (Authenticated Post Office Protocol) или нет (последний параметр метода Login):

```
$apop = 0; // APOP не используется
$error_message = $pop3->Login($user, $password, $apop);
```

После успешной аутентификации (об этом будет свидетельствовать "пустая" переменная `$error_message`) нужно получить статистику, а именно — сколько сообщений в почтовом ящике и какой размер они занимают:

```
$messages = 0; // число сообщений в ящике
$size = 0; // размер
$pop3->Statistics(&$messages, &$size);
```

Если в почтовом ящике есть сообщения, можно с помощью метода `ListMessages()` получить список сообщений или сразу методом `RetrieveMessage()` начать получать сами сообщения. Рассмотрим определение метода `ListMessages()`:

```
Function ListMessages (string message, int id)
```

Первый параметр — это число возвращаемых сообщений. Если задана пустая строка, возвращаются все сообщения (кстати, нумерация сообщений начинается с единицы, а не с нуля). Второй параметр задает, какую информацию о сообщении нужно сообщать: если `id = 1`, то возвращается уникальный идентификатор сообщения, если `id = 0`, то — размер сообщения. Список сообщений возвращается в виде массива. Вот как можно его вывести:

```
$result=$pop3->ListMessages("", 0);
for(Reset($result), $m=0; $m<count($result); Next($result), $m++)
echo '<P>Сообщение ' . Key($result) . ' '
    . $result[Key($result)] . ' байтов';
```

Метод `RetrieveMessage()` используется для получения сообщений:

```
Function RetrieveMessage($message, &$headers, &$body, $lines)
```

Первый параметр — это номер получаемого сообщения, второй параметр служит для получения заголовков сообщения, третий — для получения тела сообщения. Последний параметр регулирует, сколько строк тела сообщения необходимо получить. Если хотите получить все сообщение, установите отрицательное значение.

Для удаления сообщения с сервера служит метод `DeleteMessage()`:

```
Function DeleteMessage(int $N)
```

где `N` — это номер удаляемого сообщения.

Отключиться от сервера можно с помощью метода `Close()`.

Мы вплотную подошли к созданию собственного сценария получения сообщений, полная версия которого представлена в листинге 31.4.

Листинг 31.4. Сценарий получения сообщений

```
<?
include "pop3.php";

$user="test";
$password="12345678";
$apop=0;
$pop3=new pop3_class;

$pop3->hostname="pop.mail.ru";

$error_message = $pop3->Open();
if ($error_message != "") die($error_message);

$error_message = $pop3->Login($user,$password,$apop);
if ($error_message != "") die($error_message);

$messages = 0; // число сообщений в ящике
$size = 0;      // размер

$pop3->Statistics(&$messages,&$size);

echo "<p>Messages: $messages Size: $size";

if ($messages>0) {

    for($M=1; $M<=$messages; $M++)
    {
        // получаем сообщение с номером $M
        if(($error=$pop3->RetrieveMessage($M,&$headers,&$body,-1)) == "")
        {
            echo "<p><b>Message $M :</b>
                <p><b>Headers</b>";

            // выводим заголовки сообщения
            for($line=0;$line<count($headers);$line++)
                echo "<PRE>", htmlspecialchars($headers[$line]),
                    "</PRE>\n";

            // выводим тело сообщения
            echo "<p><b>Body</b>:<PRE>";
            for($line=0;$line<count($body);$line++)
                echo htmlspecialchars($body[$line]). "\n";
        }
    }

}

$pop3->Close();

?>
```


ПРИМЕЧАНИЕ

Напомню, что коды листингов 31.1–31.4 приведены в файлах *31-1.php*–*31-4.php* соответственно каталога *Glava_31* сопровождающего книгу электронного архива (см. приложение 4).

Результат работы сценария представлен на рис. 31.3.

Надеюсь, что материал этой главы поможет вам в разработке собственного почтового интерфейса.

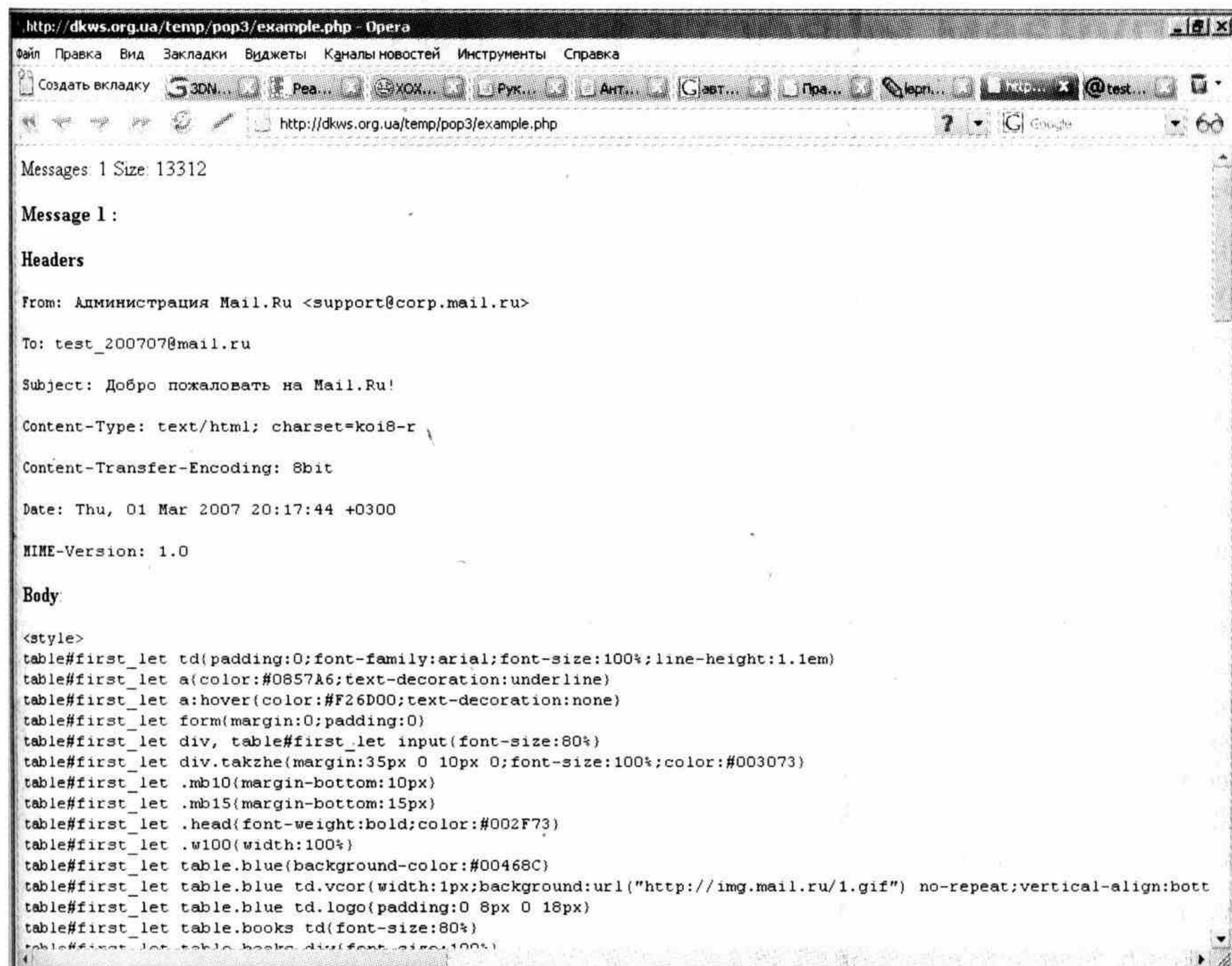


Рис. 31.3. Получение сообщений с помощью класса POP3



Работа с RSS: получаем новости на сайт

32.1. Краткие сведения о RSS

Формат RSS предназначен для публикации информации на сайтах новостей. Популярность этого формата огромна — он встречается повсеместно, начиная от передовых информационных сайтов и заканчивая обычными сетевыми дневниками (блогами).

RSS позволяет публиковать не только новости, но и любую другую информацию, которую можно разделить на части: статьи, обзоры и т. д. Ограничений на публикуемое содержимое нет.

Программы, которые "понимают" формат RSS, называются *агрегаторами*. Подобные программы довольно популярны, ведь если ваш любимый новостной сайт поддерживает RSS (в большинстве случаев так оно и есть), вам не придется заходить на сам сайт — агрегатор, который к тому же часто бывает встроен в браузер, сам подключится к серверу и загрузит последние новости. Вам останется лишь просмотреть их. Агрегаторы очень удобны, поскольку с их помощью можно получить все нужные вам новости сразу (в настройках агрегаторов можно указать несколько новостных сайтов), соответственно отпадает необходимость тратить время на посещение новостных сайтов.

Аббревиатура RSS расшифровывается по-разному, в зависимости от стандарта RSS:

- ☐ Really Simple Syndication (RSS 2.0);
- ☐ Rich Site Summary (RSS 0.91, RSS 1.0);
- ☐ RDF Site Summary (RSS 0.9 и 1.0).

В последнее время практически везде используется формат RSS версии 2.0, но все же иногда может встретиться и более ранняя версия. В этой главе мы будем рассматривать версию 2.0.

32.2. Формат RSS-файла

Прежде чем написать сценарий импорта новостей, познакомимся с форматом RSS 2.0. В листинге 32.1 представлен типичный RSS-файл, точнее, его фрагмент, поскольку приводить RSS-файл полностью нет необходимости.

Листинг 32.1. Фрагмент RSS-файла

```
<?xml version="1.0" encoding="windows-1251" ?>
<rss version="2.0">
<channel>
<title>ITnews: Софт</title>
<link>http://itnews.com.ua/</link>
<description>
  Новости Информационных Технологий. Софт
</description>
<language>ru</language>
<copyright>ITnews</copyright>
<image>
<url>http://itnews.com.ua/images/logos/88x31.gif</url>
<title>ITnews</title>
<link>http://itnews.com.ua/</link>
<width>88</width>
<height>31</height>
<description>Новости Информационных Технологий</description>
</image>
<lastBuildDate>Tue, 13 Feb 2007 16:42:00 +0200</lastBuildDate>
<ttl>30</ttl>
<textInput>
<title>Поиск</title>
<description>Поиск новостей</description>
<name>q</name>
<link>http://itnews.com.ua/s.cgi</link>
</textInput>
<item>
<title>GFI готовится к нашествию уток</title>
<link>http://itnews.com.ua/30507.html</link>
<pubDate>Tue, 13 Feb 2007 16:42:00 +0200</pubDate>
<description>Компания GFI подписала контракт с BLACK SHEEP STUDIO, получив
права на издание детской аркадной игры "Я и Гадкий Утенок"</description>
<guid isPermaLink="true">http://itnews.com.ua/30507.html</guid>
<enclosure url="http://i.proext.com/n/551203384.jpg" length="2197"
type="image/jpeg" />
</item>
<item>
<title>NVIDIA собирается выпускать драйверы ежемесячно</title>
<link>http://itnews.com.ua/30505.html</link>
<pubDate>Tue, 13 Feb 2007 15:45:00 +0200</pubDate>
<description>После волны критики, которая обрушилась на NVIDIA после выхода
Windows Vista, компания сделала ободряющее заявление</description>
<guid isPermaLink="true">http://itnews.com.ua/30505.html</guid>
<enclosure url="http://i.proext.com/n/a23b746d9.jpg" length="3246"
type="image/jpeg" />
```



```

</item>
...
</channel>
</rss>

```

RSS-файл, по сути, является XML-файлом, он написан с использованием расширенного языка разметки (XML, eXtended Markup Language). В теге XML обязательно указывается версия XML (1.0) и кодировка (windows-1251). Все это необходимо для правильного отображения документа:

```
<?xml version="1.0" encoding="windows-1251" ?>
```

Тег RSS содержит весь основной документ, параметр version задает версию RSS:

```

<rss version="2.0">
...
</rss>

```

В теге CHANNEL описывается все, что касается новостного канала: его общее описание и сами новости. Вот пример общего описания новостного канала:

```

<title>ITnews: Софт</title>
<link>http://itnews.com.ua/</link>
<description>Новости Информационных Технологий. Софт</description>
<language>ru</language>
<copyright>ITnews</copyright>

```

Тег TITLE задает название новостного канала, LINK — ссылку на новостной сайт, DESCRIPTION — подробное описание канала, LANGUAGE — язык документа, а COPYRIGHT содержит информацию об авторских правах.

Но нас больше интересует не столько информация о канале, сколько сами новости. Каждая статья новостей описывается тегом ITEM (листинг 32.2).

Листинг 32.2. Описание статьи новостей

```

<item>
<title>GFI готовится к нашествию уток</title>
<link>http://itnews.com.ua/30507.html</link>
<pubDate>Tue, 13 Feb 2007 16:42:00 +0200</pubDate>
<description>Компания GFI подписала контракт с BLACK SHEEP STUDIO, получив
права на издание детской аркадной игры "Я и Гадкий Утенок"</description>
<guid isPermaLink="true">http://itnews.com.ua/30507.html</guid>
<enclosure url="http://i.proext.com/n/551203384.jpg" length="2197"
type="image/jpeg" />
</item>

```

Назначение тегов описания новости, думаю, в особых комментариях не нуждается.

32.3. Написание сценария импорта новостей

Теперь, когда мы познакомились с форматом RSS-файла, можно приступить к написанию сценария парсинга RSS-файла. Но хочется ли вам этим заниматься? Мне — нет. Особенно, когда уже есть много готовых PHP-классов, выполняющих всю "грязную" работу. Все, что вам нужно, — это подключить класс к своему сценарию и вывести новости.

Один из таких классов — `lastRSS`, код которого вы найдете в файле `lastRSS.php` каталога `Glava_32` сопровождающего книгу электронного архива (см. приложение 4). Последняя версия этого класса доступна по адресу <http://lastrss.webdot.cz/>.

Перед тем как приступить к импорту новостей, нужно найти новостной канал, который соответствует тематике вашего сайта. По запросу новости экспорт RSS Google предоставил более миллиона ссылок, то есть дефицита RSS-каналов нет. Наоборот, есть проблема выбора. Уж слишком их много. В этой главе в качестве примера мы будем импортировать новости с сайта www.itnews.com.ua. Здесь на странице <http://itnews.com.ua/export/> имеется список доступных RSS-каналов (рис. 32.1).



Рис. 32.1. Список доступных RSS-каналов

Как видите, доступно несколько каналов, что позволяет более гибко управлять импортом. Вывод сразу всех новостей нежелателен — их много, и пользователю будет неудобно их читать. Лучше импортировать новости частями. Например, на вашем сайте есть раздел **Программы**. В него целесообразно вставить новости из категории **Софт**. А в раздел **Железо** — новости из одноименной категории.

Напишем сценарий `import.php`, который будет импортировать новости на сайт, используя RSS-класс `lastRSS` (листинг 32.3).

Листинг 32.3. Сценарий `import.php`

```
<?php

// подключаем класс lastRSS
include "../lastRSS.php";

// новый экземпляр класса
$rss = new lastRSS;

// список адресов RSS-файлов
$rss_import = array(
    'http://itnews.com.ua/export/soft.rss'
);

// выводим все RSS-файлы из списка
foreach ($rss_import as $url) {

    if ($chan = $rss->get($url)) {

        // выводим заголовок и ссылку канала
        echo "<h1><a href=$chan[link]>$chan[title]</a></h1>";

        // выводим описание канала
        echo "<p><h2>$chan[description]</h2>";

        // выводим авторские права
        echo "<p>&copy; $chan[copyright]";

        // выводим статьи
        foreach ($chan['items'] as $item)
        {
            echo "<p><b><a href=$item[link]>$item[title]</a></b><br>
                $item[description]";
        }
    }
    else {
        echo "Ошибка! Пустой канал или неправильный формат
            файла <br>$url\n<br>";
    }
}

?>
```


ПРИМЕЧАНИЕ

Код этого сценария вы найдете в файле *import.php* каталога *Glava_32* сопровождающего книгу электронного архива (см. приложение 4).

Наш сценарий позволяет указать не один, а несколько RSS-файлов — они будут выведены по очереди. Сначала мы выводим данные из заголовка RSS-канала: название канала, ссылку, описание канала и информацию об авторских правах. Затем — сами новости: заголовок, описание и ссылку, по которой можно перейти на сайт-источник новостей и прочитать интересующую новость подробнее. Зная формат RSS-файла (см. разд. 32.2), вы можете вывести дополнительные элементы заголовка или статьи, например дату публикации статьи (рис. 32.2).

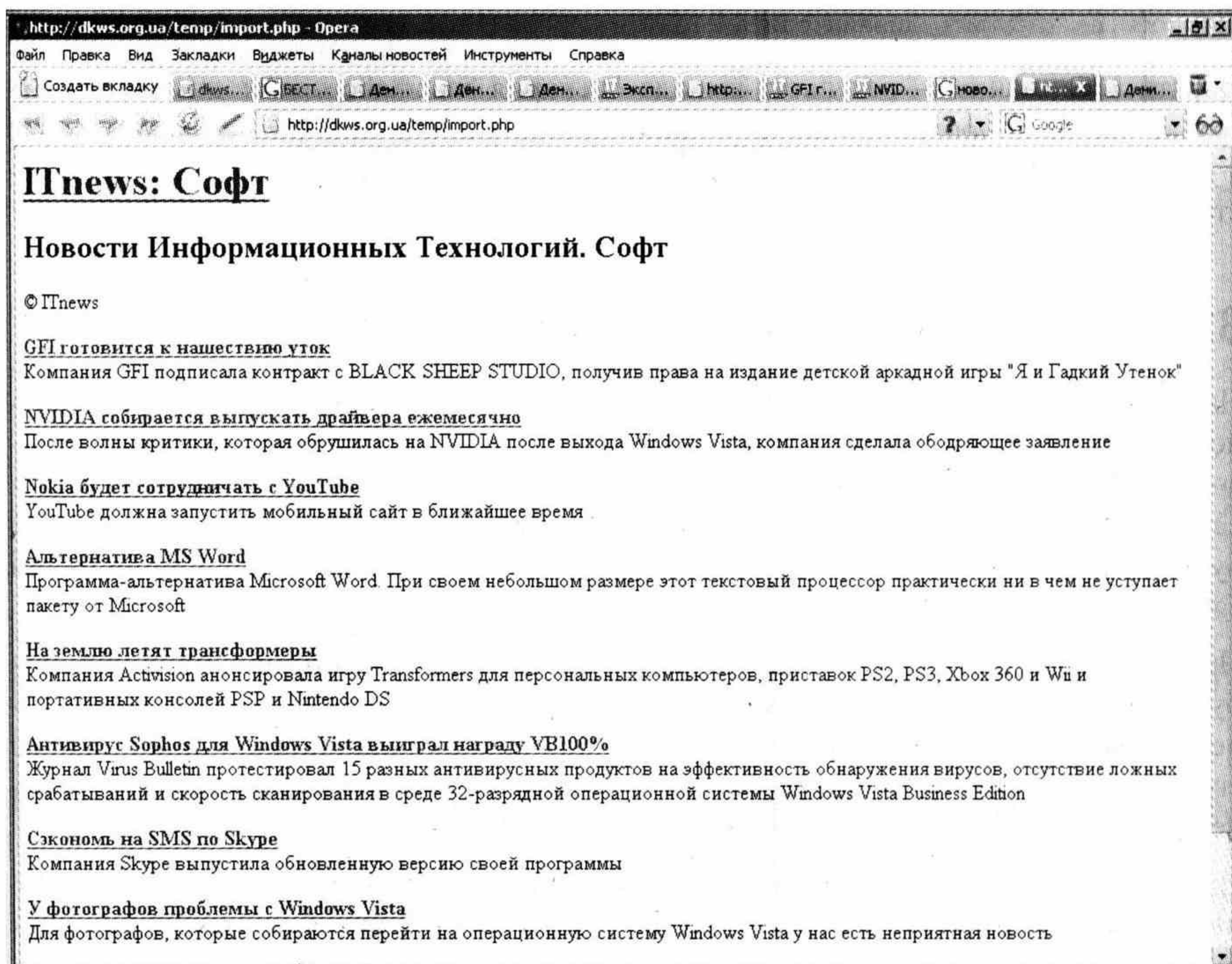


Рис. 32.2. Результат работы сценария *import.php*

32.4. Подключение файла *import.php* к сайту

Тут все зависит от вашего сайта и от его движка. Если вы используете CMS, то, возможно, вам придется преобразовать файл в модуль для вашей системы. Этот вариант мы рассматривать не будем, поскольку систем управления контентом очень много, и описать создание модуля для каждой системы просто невозможно.

Если же движок для сайта разрабатывали вы сами, тогда все намного проще. Вам просто потребуется в нужном месте подключить файл `import.php`:

```
include "import.php";
```

Если же вы написали сложную систему вывода страниц, и такое подключение файла может нарушить дизайн вашего сайта, то рекомендую поступить иначе. Сначала следует получить вывод сценария `import.php`, а затем вывести результат его работы в нужном месте:

```
// укажите ваше имя сервера  
$txt = join('',file('http://dkws.org.ua/import.php'));  
// переменную $txt следует вывести в нужном месте,  
// чтобы не нарушить дизайн сайта
```

Обратите внимание — результат вывода сценария `import.php` необходимо получать только согласно протоколу HTTP — если просто указать имя файла (`import.php`), то функция `file()` прочитает его с диска, а не по HTTP, и в результате вместо новостей будет выведен исходный код сценария.

32.5. Создание граббера новостей

Наш сценарий импорта новостей работает просто превосходно. Но ссылки на полное описание статей уведут пользователей с вашего сайта на сайты-источники новостей. В следующий раз пользователь просто зайдет на сайт-источник, и о вашем сайте забудут. Ведь так? Поэтому желательно написать граббер новостей, который будет подключаться к сайту-источнику, "забирать" новость и выводить ее на вашем сайте в вашем дизайне.

Приступим к написанию подобного сценария. Создание граббера я проиллюстрирую на примере сайта **itnews.com.ua**. Заметим, что HTML-код у других сайтов будет иным, поэтому с остальными сайтами этот вариант работать не станет. Но по его "образу и подобию", изменив пару строк кода, вы сможете адаптировать приведенный здесь пример для любого новостного сайта.

Идея заключается в следующем. Перейдите по любой ссылке, например: **http://itnews.com.ua/30519.html**. Откройте исходный код загруженной страницы. Вы увидите, что статья начинается тегом `h1`, а заканчивается тегами `</TD></TR></TABLE>`. Вы уже уловили идею? Нам остается всего лишь с помощью функции `file()` получить HTML-страницу с новостью и убрать весь лишний код до тега `h1` и после тегов `</TD></TR></TABLE>`. Конечно, также понадобится удалить все HTML-теги (кроме `P` и `IMG`) из текста статьи, поскольку они могут не вписываться в дизайн вашего сайта. Как видите, все очень просто.

Напишем такой сценарий (пусть он будет называться `news.php`) — листинг 32.4. Вызвать граббер можно по ссылке: **http://localhost/news.php?u=http://itnews.com.ua/30519.html**. Результат работы сценария представлен на рис. 32.3.

Листинг 32.4. Сценарий news.php — граббер новостей

```
<?

// получаем веб-страницу с новостью
// переменная $u содержит URL страницы
$f=join('<BR>', file($u));

// начало новости
$begin=strpos(strtolower($f), "<h1>", 0);

// конец новости
$end=strpos(strtolower($f), '</td></tr></table>', $begin);

// вырезаем то, что нам нужно — текст новости
$txt=substr($f, $begin, $end-$begin);
// некоторые сайты используют кодировку KOI8-R, поэтому
// данная строка может понадобиться для перекодирования из KOI8-R
// в windows-1251
// $txt = convert_cyr_string($txt, "k", "w");

// удаляем все теги, кроме <p> и <img>
$txt = strip_tags($txt, '<p><img>');

// выводим текст новости
echo $txt;

?>
```

ПРИМЕЧАНИЕ

Код этого сценария вы найдете в файле *news.php* каталога *Glava_32* сопровождающего книгу электронного архива (см. приложение 4).

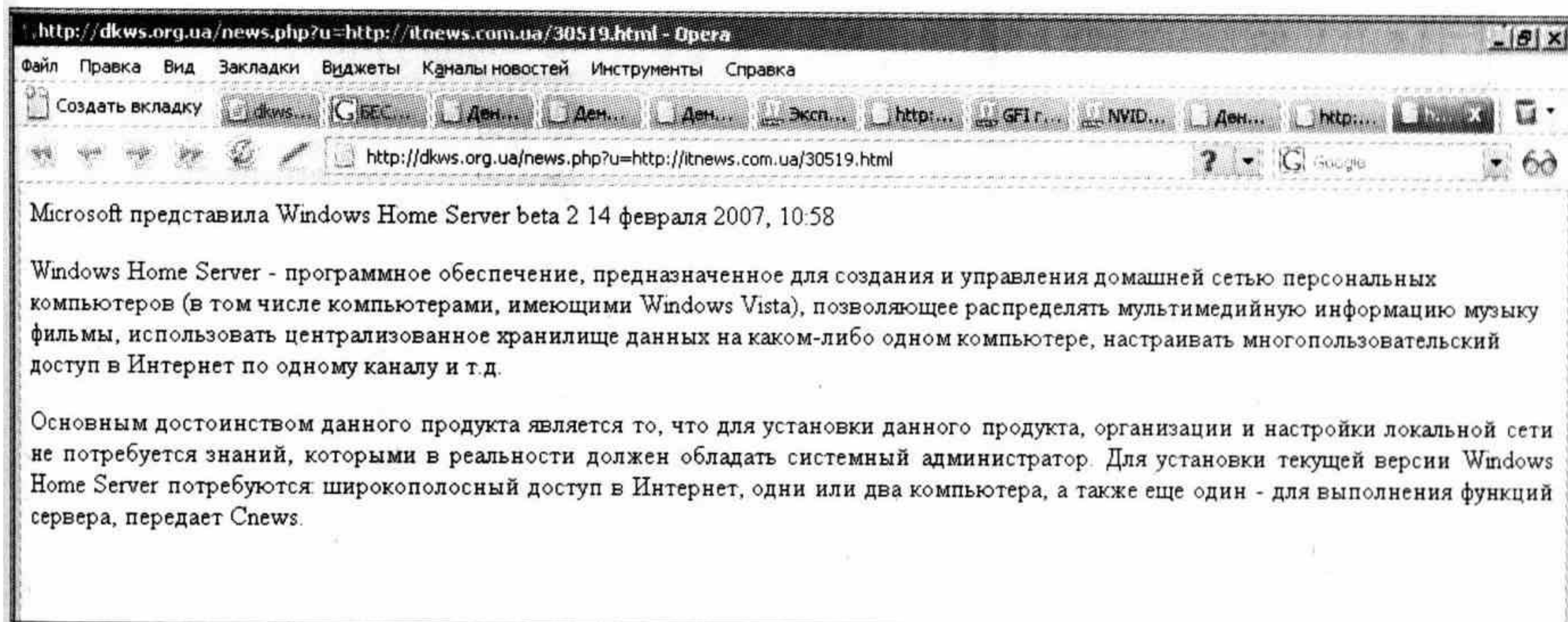


Рис. 32.3. Результат работы граббера новостей

Но это еще не все. Ведь на главной странице вашего сайта все равно выводятся ссылки на сайты-источники новостей, а не на сценарий news.php. Поэтому сейчас мы вернемся к тому месту, где подключали сценарий импорта новостей:

```
// укажите ваше имя сервера
$txt = join(' ', file('http://dkws.org.ua/import.php'));
// переменную $txt нужно вывести в нужном месте,
// чтобы не нарушить дизайн сайта
```

Переменная \$txt содержит ссылки на сайт-источник, которые нужно заменить ссылками на ваш файл:

```
// укажите свой адрес сайта
$txt = str_replace('http://',
    'http://localhost/news.php?u=http://', $txt);
```

Все, теперь наш граббер полностью готов к использованию. Вам остается импортировать его в дизайн своего сайта. Конечные результаты его работы приведены на рис. 32.4 и 32.5.



Рис. 32.4. Импорт списка новостей: конечный результат

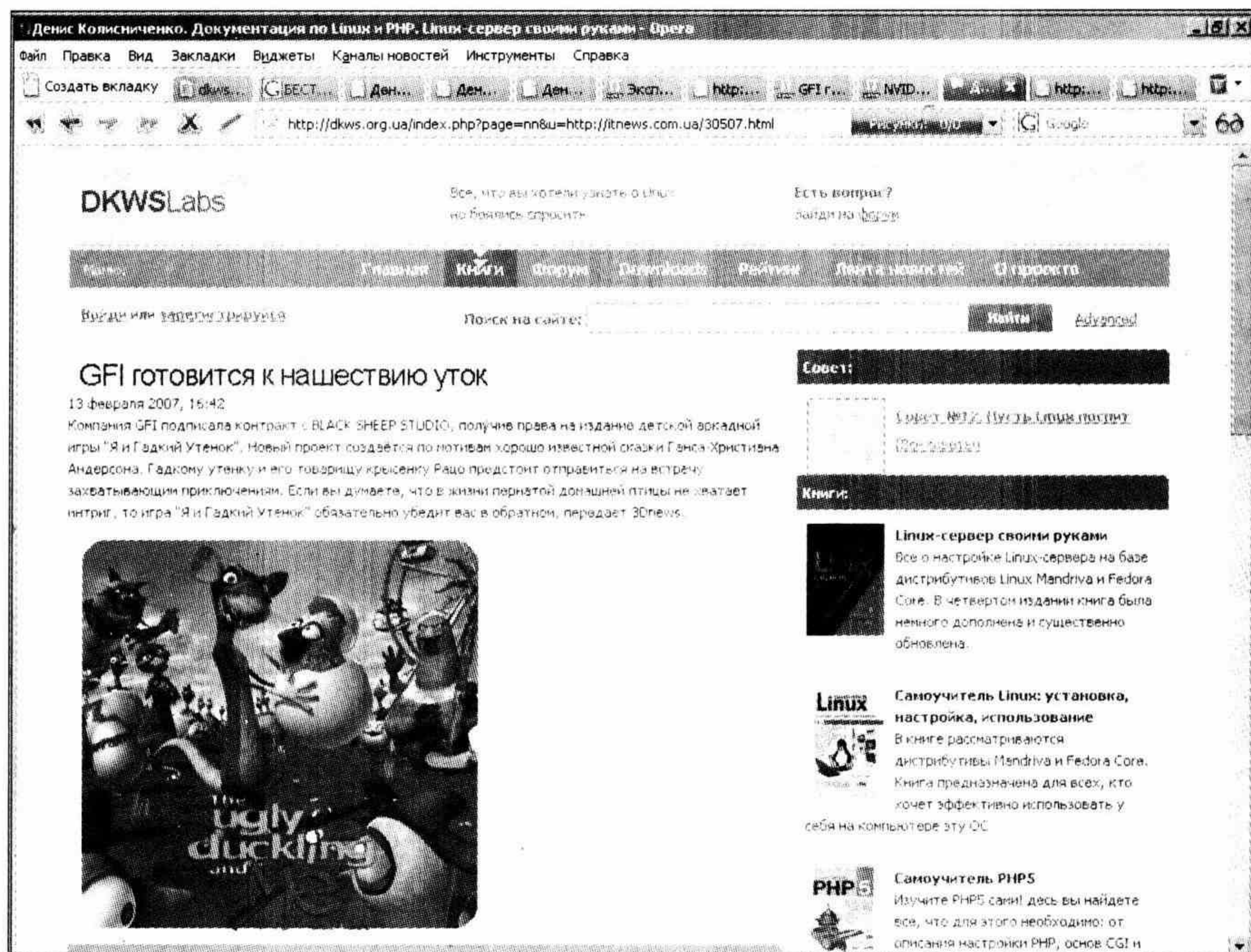
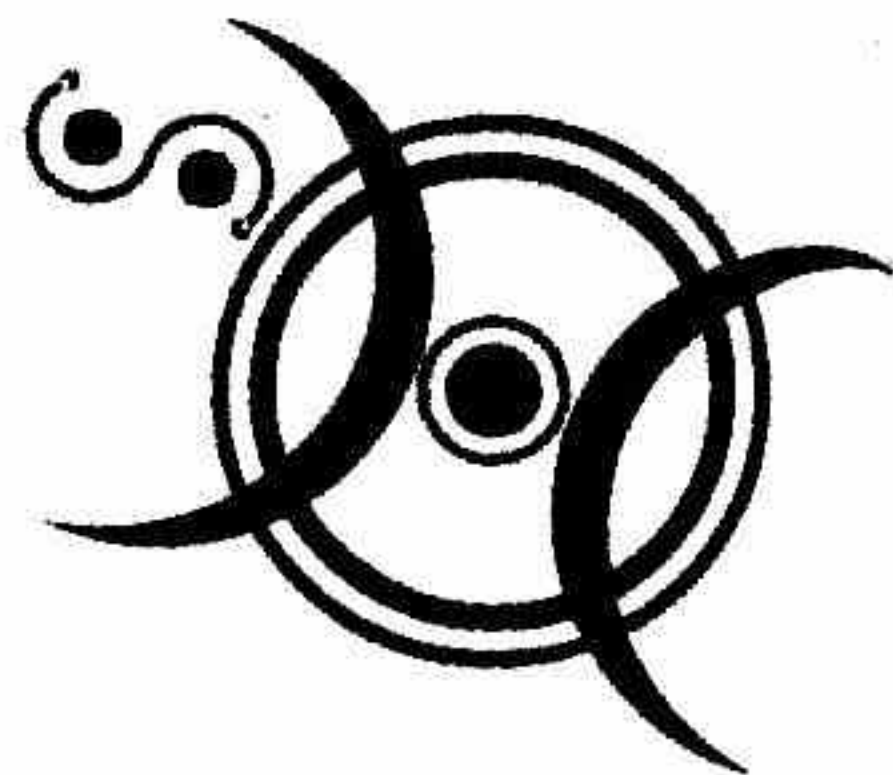


Рис. 32.5. Импорт новости

ПРИМЕЧАНИЕ

Наш граббер новостей никак не нарушает авторские права, поскольку мы не скрываем источник информации — мы честно отображаем источник новостей и не прячем оригинальные адреса новостных статей (их можно увидеть в строке адреса). Ведь можно было бы написать сценарий, который "сграбит" все новости в базу данных, а потом будет выводить информацию из нее. Но так поступать не следует, поскольку то, что мы сделали, считается перепечаткой с сохранением авторских прав, а второй вариант будет их нарушением.



Поиск с использованием регулярных выражений

33.1. Нужно что-то найти...

Иногда нам нужно найти информацию, которая соответствует определенным критериям, но что именно мы ищем, нам точно не известно. Вот небольшой пример. Предположим, что у нас есть HTML-файл. Нам необходимо найти в нем все интернет-адреса (URL). Заметьте, что нам заранее не известно, какие именно адреса есть в файле. Мы знаем только, что искомое является адресом URL.

Понятно, что обычные методы поиска здесь явно не подойдут. Да можно, конечно, вывести все строки, которые содержат подстроку 'http://'. Можно даже "обрезать" все лишнее, оставить только http:// и продолжение строки до первого пробела. Но и это не даст желаемого результата. Почему? Предположим, что в тексте есть просто подстрока 'http://' без адреса. Данная строка никак не является URL-адресом, но сценарий ее выведет. К тому же URL-адрес может начинаться не только на 'http://', кроме HTTP могут использоваться протоколы FTP и Gopher. Это тоже следует учитывать. Никто не сомневается, что вы, конечно, сможете написать сценарий, который правильно "выудит" все ссылки из HTML-файла. Но вряд ли такой сценарий будет компактным (об эффективности я вообще молчу).

А ведь РНР позволяет произвести такой поиск, вызвав всего лишь одну функцию. Этой функции нужно передать образец, т. е. "объяснить", что мы ищем. Таким образом является регулярное выражение. Поначалу все регулярные выражения будут казаться чем-то очень сложным, поэтому они и отпугивают начинающих программистов, которым кажется, что проще потратить день на создание сверхсложного сценария, чем 5 минут на написание регулярного выражения. Конечно, вам понадобится некоторое время на изучение регулярных выражений, но зато вы сэкономите очень много времени при написании сценариев. В этой главе вы убедитесь, что регулярные выражения просты, как все гениальное.

Давайте договоримся, что далее регулярные выражения для простоты будем называть *выражениями*. Чтобы понять излагаемый материал, вам нужно знать, что такое сопоставление и совпадение. *Сопоставление* — это проверка строки на соответствие регулярному выражению. Результат нашего поиска (если он есть) называется *совпадением*.

33.2. Язык регулярных выражений RegExr

PHP поддерживает два языка регулярных выражений: RegExr (классические регулярные выражения) и PCRE (Perl-Compatible Regular Expression, совместимые с Perl выражения). Но мы рассмотрим только RegExr. Во-первых, RegExr — это классика, а классику необходимо знать. Ведь в работе на других языках программирования вам тоже могут пригодиться регулярные выражения. И не думаю, что какой-то другой язык программирования, кроме Perl, поддерживает PCRE. Во-вторых, классические выражения более просты. Да и чтобы разобраться в PCRE, вам необходимо знать RegExr, поэтому, как ни крути, а с RegExr все равно пришлось бы познакомиться. В-третьих, выражения RegExr работают быстрее, чем выражения PCRE. Тем не менее, с другой стороны, по сравнению с классическими выражениями возможности PCRE шире. Как видите, в любом случае вам сначала потребуется познакомиться с RegExr, а затем приступить к изучению PCRE. Материала по PCRE в Интернете очень много, поэтому дефицита информации у вас не будет.

Теперь поговорим о самих выражениях. В каждом регулярном выражении есть как минимум одна управляющая конструкция. Таких конструкций может быть несколько, никто не запрещает создавать сложные выражения, но пока мы обойдемся самым простым вариантом, т. е. таким, когда управляющая конструкция одна.

Управляющая конструкция может содержать обычные символы, управляющие инструкции (о них отдельно), а также псевдосимволы — невидимые символы (конец строки, перевод каретки, табуляция и т. д.).

Обычные символы представляют сами себя. Если написана буква A, то перед вами именно буква A. Например, регулярное выражение `abcde` будет искать только строку `abcde`. Если в искомом тексте есть такая строка, то будет засчитано совпадение. К простым символам относятся только буквы и цифры, т. е. A–Z, a–z, 0–9.

Символы `.` `?` `*` `+` `{}` `[]` `|` `$` `^` называются управляющими. Если нужно найти строку, содержащую управляющие символы, то перед каждым управляющим символом надо добавить обратный слэш, например:

```
\[text\]
```

Такое выражение будет соответствовать строке `[text]`. Но, согласно синтаксису PHP, перед символом `\` требуется указывать еще один обратный слэш, поэтому получаем следующее выражение, которое нужно передать PHP-функции:

```
\\[text\\]
```

Помните об этом правиле, поскольку один пропущенный слэш может стать причиной того, что якобы правильно написанные регулярные выражения не заработают.

Теперь поговорим об управляющих символах. Начнем с точки. Точка заменяет один произвольный символ. Например, регулярному выражению `a.c` соответствуют строки:

```
aac, abc, acc ... azc, aAc ... aZc, a0c ... a9c.
```


Как видите, на месте точки может стоять любой символ. Если вы хотите уточнить, какой именно символ должен стоять на месте точки (например, цифра), тогда используется следующее выражение:

```
a[0123456789]c
```

То есть перечислить в квадратных скобках допустимые символы. Правда, запись `[0123456789]` не очень компактна, поэтому ее лучше заменить на `[0-9]`. Аналогично можно указать последовательность букв — например: `[a-e]`, `[a-z]`, `[A-Z]`. Задать все буквы (строчные и прописные) тоже просто: `[a-zA-Z]`. Всем алфавитно-цифровым символам соответствует выражение: `[a-zA-Z0-9_]`.

Внутри квадратных скобок, кроме символов и диапазонов символов, допустимы макросы, приведенные в табл. 33.1.

Таблица 33.1. Макросы RegExr

Макрос	Соответствие
<code>[:alpha:]</code>	Буква
<code>[:digit:]</code>	Цифра
<code>[:alnum:]</code>	Буква или цифра
<code>[:space:]</code>	Пробел или пробельный символ
<code>[:blank:]</code>	То же самое, что и <code>[:space:]</code> , но учитываются также символы с кодами 0 и 255
<code>[:cntrl:]</code>	Любой управляющий символ
<code>[:lower:]</code>	Строчная буква
<code>[:upper:]</code>	Прописная буква
<code>[:print:]</code>	Любой печатаемый символ
<code>[:punct:]</code>	Знак пунктуации (точка, запятая и т. д.)
<code>[:xdigit:]</code>	Цифра или прописная буква (A–Z)

ПРИМЕЧАНИЕ

Обратите внимание, что макрос может встречаться только внутри квадратных скобок. Например, выражение `[[:alpha:]]` — правильное, а `[:alpha:]` нет.

Если вы ищете все, кроме какого-то символа (или каких-то символов), тогда можно использовать операцию отрицания `^`:

```
[^0-9]
```

Это выражение означает — все, кроме цифр.

33.3. Управляющие конструкции

Здесь мы познакомимся с тремя управляющими конструкциями: квантификаторами, альтернативным оператором и скобками.

33.3.1. Квантификаторы

Квантификатор — это символ, служащий для уточнения значения символов, которые идут перед ним. В основном квантификатор применяется для задания повторения некоторых символов.

Вот небольшой пример квантификатора: 0^*7 . Ничего страшного в нем нет, но рассмотрим его подробнее. Звездочка $*$ означает "ноль или более символов". Вторая часть определения квантификатора (см. чуть ранее) гласит: "символов, которые идут перед ним". Перед символом $*$ в нашем случае стоит 0 . Следовательно, наше выражение будет соответствовать строке, в которой стоит ноль или более символов нуля и которая заканчивается символом 7 , то есть: 7 (ноль символов нуля), 07 , 007 , 0007 и т. д.

Квантификатор $+$ задает одно или более повторений (в отличие от квантификатора $*$, который задает ноль или более повторений).

Квантификатор $?$ задает ноль или одно повторение.

Квантификаторы $*$, $+$ и $?$ называются простыми. Сложные квантификаторы позволяют указать точное число повторов символов. Вот примеры сложных квантификаторов:

- $0\{n, k\}$ — символ 0 может встречаться от n до k раз;
- $0\{n\}$ — символ 0 должен встречаться n раз;
- $0\{n, \}$ — символ 0 может встречаться n или более раз.

ВНИМАНИЕ!

Числа n и k принадлежат диапазону $0-255$.

33.3.2. Альтернативный оператор $|$

Альтернативный оператор $|$ позволяет задать альтернативные группы символов. Рассмотрим несколько примеров (как говорится, лучше один раз увидеть):

- $A|b|c$ — эквивалентно выражению $[Abc]$;
- $joker|alex$ — данному выражению соответствуют строки, содержащие подстроку $joker$ или подстроку $alex$.

33.3.3. Скобки

Предположим, что нам нужно найти подстроки $Film1$, $Film2$ или $Film3$. Это можно сделать с помощью альтернативного оператора: $Film1|Film2|Film3$. Выражение можно немного сократить, а именно — вынести общую часть за скобки: $Film(1|2|3)$. Ведь так компактнее?

33.4. Псевдосимволы

Псевдосимволы служат для обозначения определенной позиции в строке:

- `^` — начало строки;
- `$` — конец строки;
- `[[:<:]]` — начало слова;
- `[[:>:]]` — конец слова.

Примеры:

- `^a` — все строки, которые начинаются с буквы `a`;
- `ddd$` — все строки, которые заканчиваются строкой `ddd`.

33.5. Практическое использование RegExr-функций

Самой простой функцией для работы с регулярными выражениями является `ereg()`:

```
bool ereg(string $expression, string $str [, list &$Matches])
```

Функция сопоставляет регулярное выражение, заданное параметром `$expression`, строке `$str`. Если строка соответствует выражению (есть совпадения), тогда функция возвращает `true`, в противном случае — `false`. В список `Matches`, если он задан, заносятся отдельные совпадения, если они есть.

Как видите, функция `ereg()` служит для поиска совпадений. Но намного чаще приходится выполнять замену совпадения на другую строку. Если бы не было других PHP-функций, то для замены всех совпадений пришлось бы использовать следующий алгоритм:

1. Найти все совпадения (функция `ereg()` сформировала массив `Matches`).
2. "Пройтись" по всем совпадениям и выполнить замену.

Предположим, что мы ищем подстроки, соответствующие выражению `1[a-z]2`. В нашей строке три совпадения: `1a2`, `1b2`, `1c3`. Эти совпадения будут занесены в массив `Matches`. Когда мы знаем, что ищем (ведь до этого мы знали только характеристику искомого, а теперь все существующие подстроки, соответствующие выражению, нам известны), мы можем произвести замену.

Понятно, что наш код в случае замены не был бы компактен: сначала вызов `ereg()`, затем цикл, в котором вызывается функция замены. Намного проще задействовать уже готовые функции.

Функция `ereg_replace()` позволяет сразу же заменить найденную строку:

```
string ereg_replace(string $expression,  
                    string $replacement,  
                    string $str)
```


Тут все просто: первый параметр — выражение, третий — исходная строка, в которой будет осуществляться поиск, второй параметр задает замену для найденных подстрок.

Рассмотрим небольшой пример поиска и замены в строке. Выведем РНР-код программы, выделив имена переменных полужирным. Другими словами, нашей программе требуется найти все переменные и перед выводом обрмить их тегами `<B></B>`.

Для того чтобы найти переменные, нужно вспомнить правила формирования имени переменной. Имя переменной начинается символом `$`, после которого может стоять только буква (большая или маленькая — без разницы) или знак подчеркивания, но не цифра, а далее — любой алфавитно-цифровой символ. Поэтому для поиска переменных можно использовать выражение:

```
\${a-zA-Z_}[:,alnum:]]*
```

Нужная нам программа представлена в листинге 33.1.

Листинг 33.1. Пример поиска и замены в строке

```
<?php

$php_code = '$var = 1; echo $var; ';

$res = ereg_replace('\${a-zA-Z_}[:,alnum:]]*', "<b>\0</b>", $php_code);

echo "<PRE>".$res."</PRE>";

?>
```

ПРИМЕЧАНИЕ

Код этой программы вы найдете в файле `33-1.php` каталога `Glava_33` сопровождающего книгу электронного архива (см. приложение 4).

Переменная `$php_code` — это наша исходная строка. Первый параметр функции `ereg_replace()` обеспечивает поиск переменных. Но второй параметр (замена) уже не столь прост. Если бы мы знали, на какую именно строку заменять, мы бы просто написали `<B>строка</B>`, но мы ведь не знаем, какие у нас будут переменные: `$var`, `$a` или `$BDC`. Поэтому мы применяем модификатор `\0` для замены на найденное совпадение. Результат работы сценария представлен на рис. 33.1.

Рассмотрим еще один пример замены. Предположим, что у нас есть строка вида:

Название сайта (URL)

URL в скобках нам нужно заменить на ссылку — обрмить его тегами `<A href=> </A>`. Как и в предыдущем случае, сначала нужно составить регулярное выражение, а потом — шаблон замены. Выражение, которое будет обеспечивать поиск HTTP-адресов, выглядит так:

```
http://(( [A-Za-z0-9.\-] )*)
```

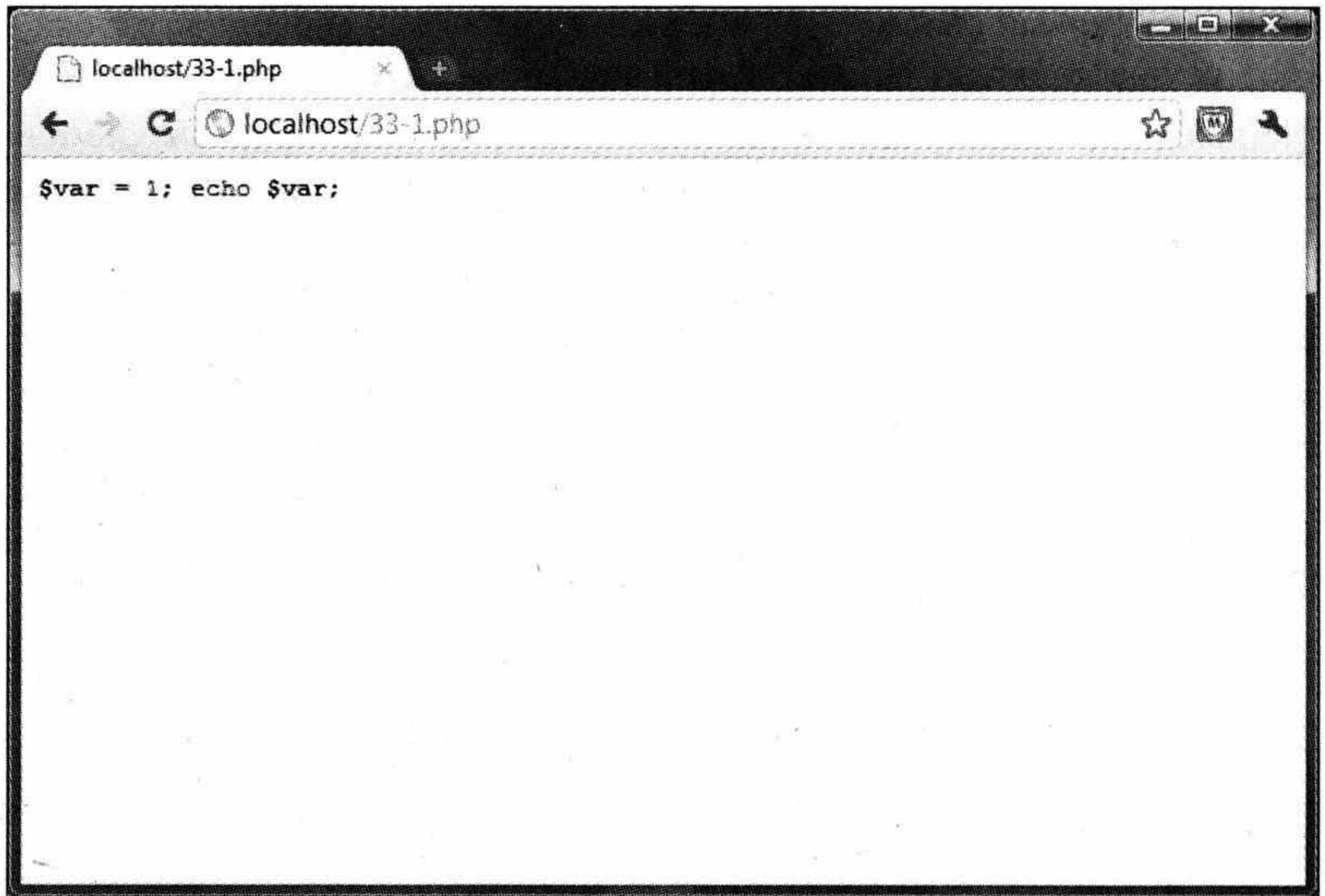



Рис. 33.1. Результат работы сценария из листинга 33.1

Сценарий, выполняющий нашу задачу, представлен в листинге 33.2. Результат работы сценария показан на рис. 33.2.

Листинг 33.2. Пример поиска HTTP-адресов

```
<?php
$url = "Мой сайт (http://www.dkws.org.ua)";
$url = ereg_replace("http://([A-Za-z0-9.\-])*",
    "<a href=\"\\0\">\\0</a>", $url);
echo $url;
?>
```

ПРИМЕЧАНИЕ

Код этого сценария вы найдете в файле *33-2.php* каталога *Glava_33* сопровождающей книгу электронного архива (см. приложение 4).

Кроме функций `ereg()` и `ereg_replace()`, имеются функции `eregi()` и `eregi_replace()` (листинг 33.3).

Листинг 33.3. Пример вызова функций `eregi()` и `eregi_replace()`

```
bool eregi(string $expression,
    string $str
    [, list &$Matches])
```



```
string eregi_replace(string $expression,
                    string $replacement,
                    string $str)
```

Работают они почти так же, как и предыдущие, но не учитывают регистра символов.

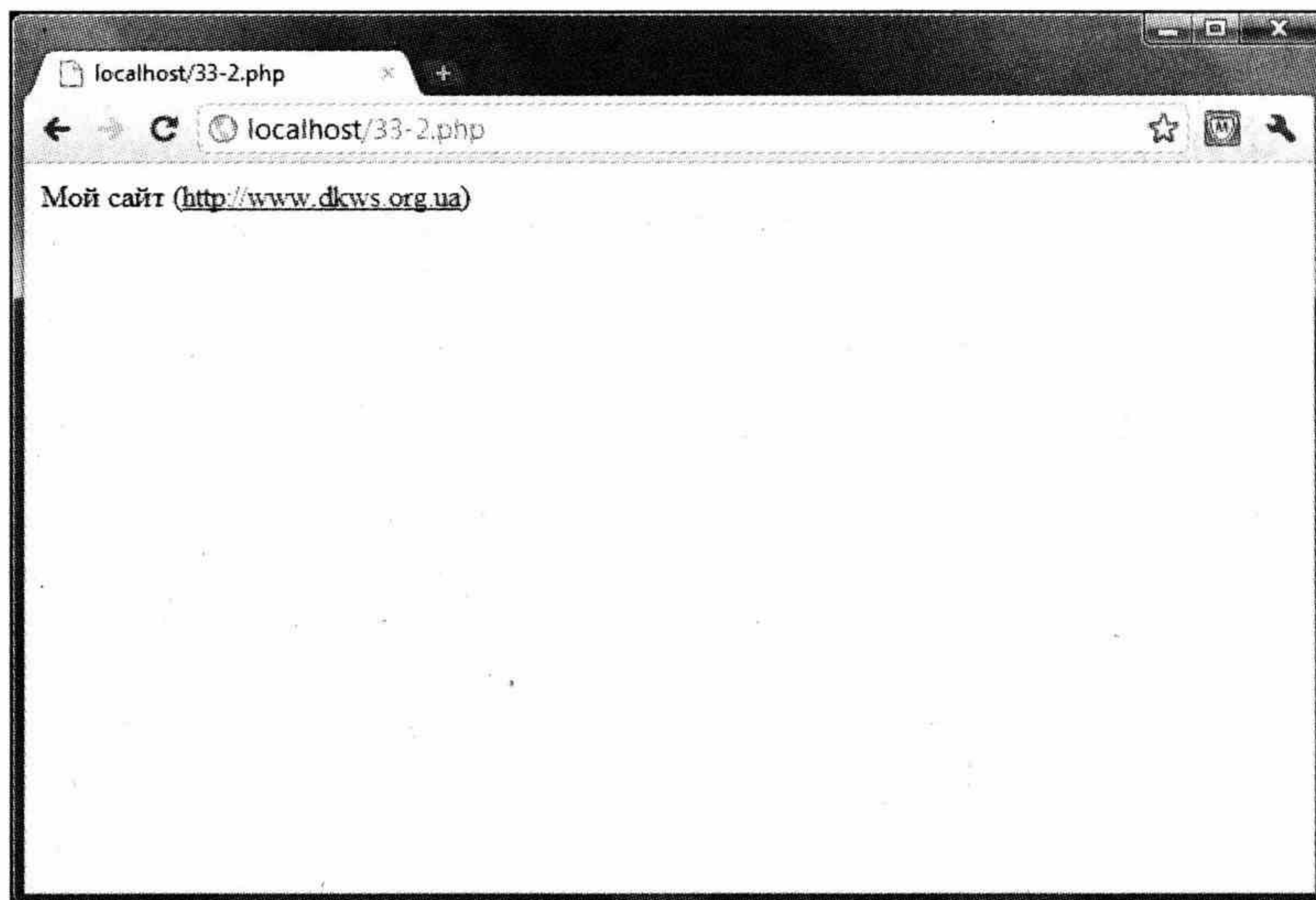


Рис. 33.2. Результат работы сценария из листинга 33.2

Рассмотрим еще несколько полезных примеров регулярных выражений. Регулярные выражения в них уже представлены в виде строк, которые можно использовать в PHP-функциях (с цитированием необходимых символов).

Следующее выражение является более корректным, чем приведенное ранее, для поисков URL-адресов:

```
((https?|ftp|gopher)://[[:alnum:]]-(\\.[[:alnum:]]-)+)*".
(/[[:alnum:]]+&.*%*(\\?[[:alnum:]]?+&%*)?)?)
```

Если присмотреться, то можно обнаружить, что оно состоит из трех частей: имени протокола, имени узла, а также имени файла и параметров CGI:

```
((https?|ftp|gopher)://           — имя протокола
[[:alnum:]]-(\\.[[:alnum:]]-)+)*". — имя узла
(/[[:alnum:]]+&.*%*(\\?[[:alnum:]]?+&%*)?)?) — имя файла +
                                           параметры CGI
```

Второй распространенный пример — поиск адресов электронной почты в тексте. Для этого можно применить следующее регулярное выражение:

```
([[:alnum:]]-.[[:alnum:]]-(\\.[[:alnum:]]-)+*(\\?([[:alnum:]]?+&%*)?)?)
```


И еще один пример. Предположим, что у нас есть полное имя файла, содержащее путь и само имя файла. Нам нужно разбить это имя на две части: имя каталога и имя файла. Сделать это можно с помощью двух операторов:

```
$fullname = 'c:\directory\file.txt';  
  
$filename = ereg_replace(".*[\\\/]", "", $fullname);  
$dirname = ereg_replace("[\\\/]?[^\\\/]*$", "", $fullname);
```

33.6. Фильтры, или Конец эры регулярных выражений

В 2014 году исполнилось ровно 13 лет с момента написания мной первого PHP-сценария. Познакомился я с PHP, когда была уже доступна четвертая версия, хотя на некоторых хостингах все еще оставалась третья. Потом состоялся плавный (местами, правда, не совсем) переход на 5-ю версию.

Третья версия PHP поддерживала только RegExr, в четвертой версии добавилась поддержка регулярных выражений в стиле PCRE. Зато в пятой версии появились фильтры. Самое интересное, что фильтры не были замечены большинством программистов. Почему так получилось, не знаю — то ли все привыкли к старому и доброму RegExr, то ли просто невнимательно читали руководство и список новых возможностей PHP 5.

Фильтры позволяют решить ряд задач, которые мы привыкли решать с помощью регулярных выражений. Как показывает практика, чаще всего использование регулярных выражений сводится к проверке корректности e-mail, URL и IP-адресов. Однако возможности фильтров гораздо шире. Чтобы не лишать вас удовольствия прочитать руководство и самостоятельно во всем разобраться, сейчас я приведу только ссылку на фильтры и покажу несколько примеров их использования, а дальше — уж вы сами...

Вот эта заветная ссылка: <http://ru.php.net/manual/en/book.filter.php>.

Да, на русском аналогичного материала пока нет. Но, думаю, скоро он появится.

Все фильтры делятся на три группы:

- ❑ проверяющие фильтры (validate filters). Основная задача таких фильтров — проверка, например, является строка адресом e-mail или нет;
- ❑ "санитарные" фильтры (sanitize filters) — позволяют удалить из строки потенциально опасные символы;
- ❑ другие фильтры (others filters) — позволяют вызвать пользовательскую функцию для проверки.

Основная функция для работы с фильтрами — `filter_var()`:

```
mixed filter_var ( mixed $variable [, int $filter = FILTER_DEFAULT  
[, mixed $options ] ] )
```


Первый параметр — переменная, подлежащая проверке, второй — номер применяемого фильтра (хотя удобнее использовать константу), третий параметр позволяет задать флаги фильтров (дополнительные параметры проверки).

Рассмотрим пример проверки корректности e-mail с использованием фильтра валидации `FILTER_VALIDATE_EMAIL`:

```
$var = 'mail@example.com';  
var_dump(filter_var($var, FILTER_VALIDATE_EMAIL));
```

Если e-mail некорректный, функция вернет значение `false`. Если e-mail корректный, функция просто вернет переданное ей значение (e-mail).

Проверить корректность IP-адреса можно фильтром `FILTER_VALIDATE_IP`:

```
$ip = '2011:272:1f11:161:140:23ff:dee0:2001';  
// если это не IPv4, вы получите значение false  
var_dump(filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));  
// если это не IPv6, вы получите значение false  
var_dump(filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
```

Первый вызов `var_dump` выведет `(bool) false` — что и следовало ожидать, поскольку указан IPv6-адрес. Второй вызов выведет просто переданный IPv6-адрес.

Для проверки корректности URL служит фильтр `FILTER_VALIDATE_URL`.

Рассмотрим несколько санитарных фильтров:

- фильтр `FILTER_SANITIZE_EMAIL` удаляет из строки все символы, кроме букв, цифр и символов `!#$%&'*+,-/=/?^_`{|}~@.[]`. Результат работы функции `filter_var()` — текст, из которого удалены все символы, кроме указанных;
- фильтр `FILTER_SANITIZE_URL` служит для "очистки" URL — удаляются все символы, которые не допускаются в URL.

ГЛАВА 34



Работаем с MP3

34.1. Формат MP3

В настоящее время в Интернете существует очень много сайтов, предлагающих скачать музыку в формате MP3. Для каждой композиции выводится соответствующая информация: название, исполнитель, альбом, год, жанр и др. Как вы думаете, где хранятся эти сведения? Скорее всего, ваш ответ будет: "в базе данных". Вполне вероятно, что это именно так. Но мы рассмотрим иной подход к созданию подобного музыкального сайта.

Конечно, можно хранить всю информацию в базе данных, но ведь формат MP3 поддерживает так называемые ID3-теги, в которых и содержится нужная нам информация! Зачем же дублировать информацию о композиции в базе данных, ведь она уже есть в MP3-файле? Достаточно достать ее оттуда.

Что же такое ID3-тег? Он представляет собой структуру размером 128 байтов, которая дописывается в конец MP3-файла. Структура ID3-тега версии 1.0 следующая:

- ☐ служебное слово TAG;
- ☐ название композиции (30 символов);
- ☐ исполнитель (30 символов);
- ☐ альбом (30 символов);
- ☐ год записи (четырёхзначное число);
- ☐ комментарий (30 символов);
- ☐ жанр (1 байт).

Но ID3-теги версии 1.0 встречаются редко. Значительно чаще используется версия 1.1, которая отличается от 1.0 тем, что в ней сокращена длина комментария — до 28 символов, но зато появилось новое поле, содержащее номер дорожки, т.е. номер композиции в альбоме. На рис. 34.1 показан фрагмент MP3-файла (в шестнадцатеричном виде): видно служебное слово TAG, после которого следуют название композиции, имя исполнителя, название альбома, а остальная информация для этого MP3-файла не указана.

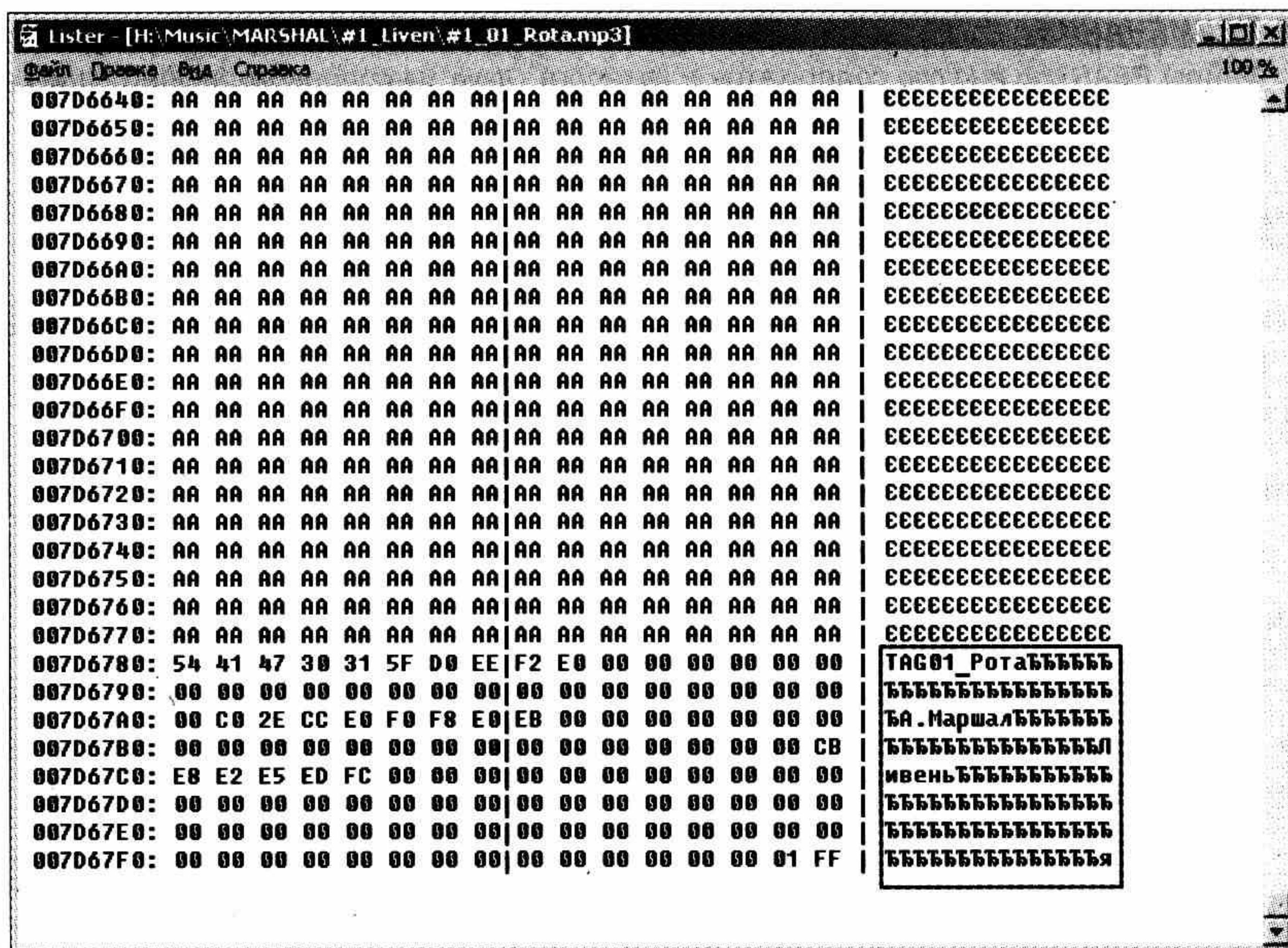


Рис. 34.1. MP3-файл (шестнадцатеричный вид)

34.2. Библиотека PEAR

Если честно, то копаться в MP3-файлах в поисках нужной информации как-то не хочется. Нет, с технической стороны это совсем не сложно, дело в отсутствии желания. Причем до такой степени, что можно уговорить себя на создание отдельной таблицы, которая будет содержать всю необходимую нам информацию. К счастью, на помощь приходит библиотека PEAR (подробно она была рассмотрена в главе 26). Ее возможности достаточно велики, но мы будем ее использовать (в паре с пакетом MP3_Id) для чтения ID3-тегов.

В каталоге Glava_34 сопровождающего книгу электронного архива (см. приложение 4) вы найдете:

- файл PEAR-1.9.0.tgz с версией 1.9.0 пакета PEAR¹. Последнюю версию пакета можно скачать по адресу: <http://pear.php.net/package/PEAR/download>;
- файл MP3_Id-1.2.0RC3.tgz с пакетом MP3_Id. Последняя версия пакета доступна по адресу: http://pear.php.net/package/MP3_Id/download/.

Из первого архива нам понадобится файл PEAR.php, из второго — Id.php. Распакуйте эти файлы в отдельный каталог.

¹ Это тот же файл, что и в каталоге Glava_26, однако удобно, когда все нужное лежит вместе, не так ли?

ПРИМЕЧАНИЕ

Файлы *PEAR.php* и *Id.php* содержатся в каталоге *Glava_34* сопровождающего книгу электронного архива (см. приложение 4).

34.3. Вывод ID3-тегов

Для вывода информации из ID3-тега потребуется сначала подключить библиотеку *MP3_Id*:

```
require_once 'Id.php';
```

Обращение к ID3-тегам осуществляется через объект класса *MP3_id*:

```
$id3 = &new MP3_Id();
```

После того как объект создан, следует прочитать MP3-файл методом *read()*:

```
$result = $id3->read(<имя_файла>);
```

Затем методом *getTag()* можно прочитать информацию о теге. Методу нужно передать название тега (табл. 34.1).

Таблица 34.1. ID3-теги

Название тега	Описание
<name>	Название композиции
<artists>	Исполнитель
<album>	Альбом
<year>	Год записи
<comment>	Комментарий
<genre>	Жанр
<genreno>	Номер жанра (число)
<track>	Номер композиции в альбоме

Теперь напомним небольшой сценарий, выводящий информацию об MP3-файле (листинг 34.1).

Листинг 34.1. Вывод информации об MP3-файле

```
<?php
// подключаем MP3_Id
require_once 'Id.php';

// создаем объект
$id3 = &new MP3_Id();
```



```
// читаем MP3-файл 1.mp3
$result = $id3->read('1.mp3');

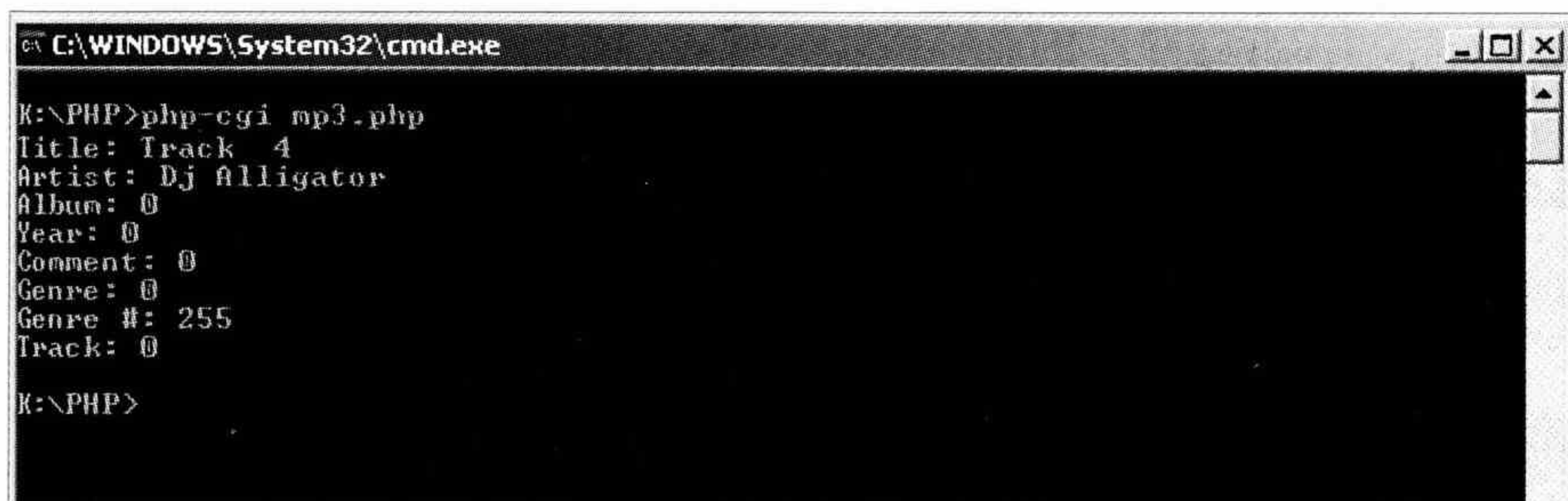
// есть ли ошибка?
if (PEAR::isError($result)) {
    die($result->getMessage() . "\n");
}

// читаем поля и выводим информацию
// название
echo 'Title: ' . $id3->getTag('name') . "\n";
// исполнитель
echo 'Artist: ' . $id3->getTag('artists') . "\n";
// альбом
echo 'Album: ' . $id3->getTag('album') . "\n";
// год
echo 'Year: ' . $id3->getTag('year') . "\n";
// комментарий
echo 'Comment: ' . $id3->getTag('comment') . "\n";
// жанр
echo 'Genre: ' . $id3->getTag('genre') . "\n";
// жанр (номер)
echo 'Genre #: ' . $id3->getTag('genreno') . "\n";
// номер дорожки
echo 'Track: ' . $id3->getTag('track') . "\n";
?>
```

ПРИМЕЧАНИЕ

Очищенный от комментариев код этого сценария вы найдете в файле *mp3.php* каталога *Glava_34* сопровождающего книгу электронного архива (см. приложение 4).

Результат работы сценария показан на рис. 34.2.



```
C:\WINDOWS\System32\cmd.exe
K:\PHP>php-cgi mp3.php
Title: Track 4
Artist: Dj Alligator
Album: 0
Year: 0
Comment: 0
Genre: 0
Genre #: 255
Track: 0
K:\PHP>
```

Рис. 34.2. Результат работы сценария (листинг 34.1)

34.4. Редактирование ID3-тегов

Библиотека MP3_id позволяет не только просматривать ID3-теги, но и редактировать их. Для этого предусмотрен метод `setTag()`, которому требуется передать два параметра (название и значение тега):

```
function setTag($name, $value)
```

Запись изменений в MP3-файл реализует метод `write()`:

```
$id3->write();
```

Рассмотрим небольшой пример установки всех тегов (листинг 34.2).

Листинг 34.2. Пример установки тегов

```
require_once 'Id.php';
$id3 = &new MP3_Id();
$id3->read('1.mp3');

$id3->setTag('name', 'Следуй за мной');
$id3->setTag('artists', 'Ария');
$id3->setTag('album', 'Кровь за кровь');
$id3->setTag('year', 1991);
$id3->setTag('comment', '');
$id3->setTag('genre', 'Rock');
$id3->setTag('track', 8);

// пытаемся записать теги в MP3-файл
$result = $id3->write();

if (PEAR::isError($result)) {
    // произошла ошибка
    die($result->getMessage() . "\n");
}

echo "Теги записаны!";
```

34.5. Удаление тега

Для удаления ID3-тега служит метод `remove()` (листинг 34.3). Удаляется весь ID3-тег (все его поля).

Листинг 34.3. Удаление ID3-тега

```
require_once 'Id.php';
$id3 = &new MP3_Id();
```



```
$id3->read('1.mp3');  
$id3->remove();  
if (PEAR::isError($result)) {  
die($result->getMessage() . "\n");  
}
```

```
echo "Тег стерт!";
```

Удаление тега, как и его запись, требуют изменения MP3-файла, поэтому убедитесь, что вы обладаете должными правами для выполнения этой операции.

ГЛАВА 35



Технология AJAX

35.1. Что такое AJAX?

Аббревиатура AJAX расшифровывается как **A**ynchronous **J**avaScript **A**nd **X**ML, что означает "асинхронный JavaScript и XML". Говорит это не много, поэтому иногда проще объяснить, что такое AJAX на примере, чем вникать в суть расшифровки его названия.

Предположим, у нас есть HTML-страница, содержащая поле для ввода имени пользователя, и есть PHP-сценарий, выводящий приветствие. Сначала рассмотрим HTML-страницу:

```
<form action=hello.php method=get>
Как вас зовут? <input type=text name=myname>
<p><input type=submit value=Передать>
</form>
```

Эта страница передает параметр с именем `myname` сценарию `hello.php`:

```
<?php
$myname = $_GET['myname'];

echo "$myname, welcome!";
?>
```

При нажатии кнопки **Передать** произойдет перезагрузка страницы — вместо исходной страницы будет загружена страница `hello.php`, которая выведет строку:

```
<Имя>, welcome!
```

Технология AJAX, используя JavaScript и XML, позволяет получить данные от сервера без перезагрузки страницы. То есть исходная страница будет обновлена данными, полученными от сервера, в то время, когда пользователь все еще с ней работает. Взаимодействию AJAX и PHP посвящена не одна книга, мы же рассмотрим только основы этого взаимодействия. Заинтересовавшиеся без проблем найдут нужную книгу в любом книжном магазине.

35.2. Ваше первое AJAX-приложение

Сейчас мы разработаем первое AJAX-приложение, состоящее из трех файлов: `index.html`, `ajax.js` и `ajax.php`. Первый файл — это обычная HTML-форма, содержащая вызов JavaScript-сценария `ajax.js`. В сценарии `ajax.js` находится JavaScript-код, выполняющий асинхронное обращение к PHP-сценарию `ajax.php`. В результате выполнения сценария `ajax.php` будет сгенерирован XML-код, который передается сценарию `ajax.js`. Сценарий `ajax.js`, получив XML-код, на его основании обновит HTML-страницу `index.html`.

Рассмотрим файл `index.html` (листинг 35.1).

Листинг 35.1. Файл `index.html`

```
<html>
<head>
<title>Ваше первое Ajax-приложение</title>
<script type="text/javascript" src="ajax.js"></script>
</head>
<body onload='async()'>
Как вас зовут?: <input type="text" id="UserName" />
<div id="WelcomeMessage" />
</body>
</html>
```

Полужирным выделены самые важные имена из этого файла:

- `ajax.js` — название сценария, содержащего AJAX-код;
- `async()` — название функции из файла `ajax.js`, которая будет выполнять асинхронный запрос;
- `UserName` — имя поля, содержащего ввод пользователя;
- `WelcomeMessage` — сюда будем выводить ответ от сервера.

Обратите внимание — JavaScript-функцию `async()` из файла `ajax.js` мы вызываем сразу при загрузке страницы, затем в коде `ajax.js` обеспечивается ее перезагрузка каждую секунду. Такой режим больше подходит для демонстрации возможностей AJAX, но в реальных проектах он не очень удобен — пользователь еще не успеет заполнить форму (ввести имя пользователя и пароль, например), а вы уже передаете данные сценарию `ajax.php`. Наш режим подойдет для проверки доступности имени пользователя при регистрации нового пользователя, а вот для самого входа в систему надо дожидаться, пока пользователь введет полностью все данные. Для этого следует исключить вызов `async()` из тега `<body>`, а вызывать его при нажатии какой-то кнопки, например:

```
<body>
Как вас зовут? <input type="text" id="name" />
<input type="BUTTON" value="OK" name="go" onClick="async()">
```


В этом случае введенное имя будет передано сценарию `ajax.php` только при нажатии кнопки **ОК** — сценарий обработает введенную информацию и выведет свой ответ в `<div>` с именем `WelcomeMessage`.

Теперь рассмотрим код сценария `ajax.js` (листинг 35.2).

Листинг 35.2. Сценарий `ajax.js`

```
// сохраняем объект XMLHttpRequest
var xml = createObject();

// получаем объект XMLHttpRequest
function createObject()
{
    var xml;
    // если Internet Explorer
    if(window.ActiveXObject)
    {
        try
        {
            xml = new ActiveXObject("Microsoft.XMLHTTP");
        }
        catch (e)
        {
            xml = false;
        }
    }
    // если Mozilla или другой браузер
    else
    {
        try
        {
            xml = new XMLHttpRequest();
        }
        catch (e)
        {
            xml = false;
        }
    }
    // возвращаем объект или сообщение об ошибке
    if (!xmlHttp)
        alert("Error");
    else
        return xml;
}

// делаем асинхронный HTTP-запрос
function async()
```



```
{
// только если объект xml не занят
if (xml.readyState == 4 || xml.readyState == 0)
{
// получаем введенное пользователем имя
name = encodeURIComponent(document.getElementById("UserName").value);

// передаем имя сценарию ajax.php методом GET как параметр name
xml.open("GET", "ajax.php?name=" + name, true);
xml.onreadystatechange = ServerResponse;

// делаем запрос серверу
xml.send(null);
}
else
    // если соединение занято, ждем 1 секунду (1000 мс)
    setTimeout('async()', 1000);
}

// запускается при получении ответа от сервера
function ServerResponse()
{
    if (xml.readyState == 4)
    {
        // код 200 – транзакция прошла успешно
        if (xml.status == 200)
        {
            // получаем XML от сервера
            xmlResponse = xml.responseXML;
            xmlDocumentElement = xmlResponse.documentElement;
            // получаем текстовое сообщение, которое является
            // первым Child-элементом
            // XML-документа
            welcome = xmlDocumentElement.firstChild.data;

            // обновляем страницу index.html
            document.getElementById("WelcomeMessage").innerHTML =
                '<i>' + welcome + '</i>';

            // перезапуск
            setTimeout('async()', 1000);
        }
        // если код <> 200 – ошибка
        else
        {
            alert("Error: " + xml.statusText);
        }
    }
}
```


Функция `createObject()` создает объект, который будет использоваться для осуществления асинхронного запроса к серверу. Функция `async()` производит этот запрос. Сначала она получает данные из формы (в нашем случае — это значения поля ввода `UserName`) и передает их сценарию `ajax.php`. Как только придет ответ сервера, будет запущена функция `ServerResponse()`:

```
xml.onreadystatechange = ServerResponse;
```

Эта функция распаковывает полученный XML-объект, извлекает из него текстовое сообщение и обновляет поле `WelcomeMessage` нашей формы из файла `index.html`.

Нам осталось реализовать только сценарий `ajax.php`, выводящий приветствие в формате XML. Код сценария `ajax.php` представлен в листинге 35.3.

Листинг 35.3. Сценарий `ajax.php`

```
<?php
// ответ нужно передать в формате XML
header('Content-Type: text/xml');

// XML-заголовок
print ('<?xml version="1.0" encoding="UTF-8" standalone="yes"?>');

// создаем элемент <response>
print ('<response>');

// получаем параметр name
$name = $_GET['name'];

// если ничего не введено, просим ввести имя
if (trim($name) == '')
print('Please input your name!');
else
print(htmlentities($name) . ', welcome!');

// закрываем <response>
print ('</response>');
?>
```

О ЗАПУСКЕ ПРИМЕРА

Поместите все три файла в каталог веб-сервера. Желательно создать для них отдельный каталог, например, с именем `ajax`. Откройте браузер и загрузите главный файл — `index.html`: <http://localhost/ajax/index.html>.

Проверить в действии созданное нами AJAX-приложение можно по адресу: <http://dkws.org.ua/temp/ajax/>.

ПРИМЕЧАНИЕ

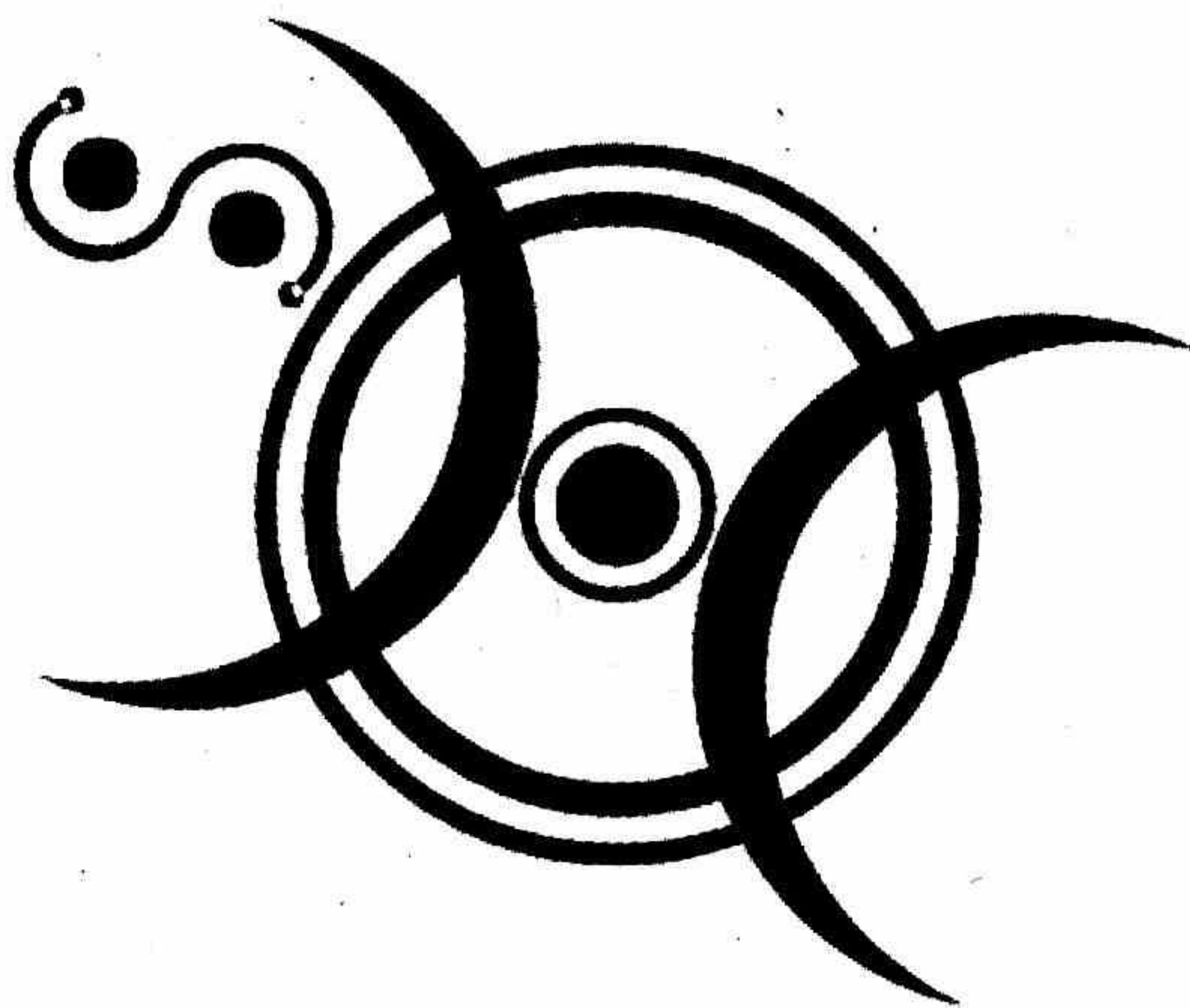
Файлы `index.html`, `ajax.php` и `ajax.js` вы найдете в каталоге `Glava_35` сопровождающего книгу электронного архива (см. приложение 4). Копии файлов из каталога `Glava_35` также можно скачать по адресу: <http://dkws.org.ua/files/php6/35.zip>.

Как видите, в AJAX нет ничего сложного. Конечно, наш сценарий `ajax.js` весьма примитивен. Ознакомиться с другими AJAX-инструментами для PHP можно по следующему адресу:

<http://www.dkws.org.ua/phpbb2/viewtopic.php?p=14075#14075>.

ПРИМЕЧАНИЕ

Если вы по каким-то причинам пропустили главу 29, настоятельно рекомендую к ней вернуться, поскольку там рассмотрен еще один пример использования AJAX (а также jQuery и jQuery-UI) — для создания индикатора загрузки файла.

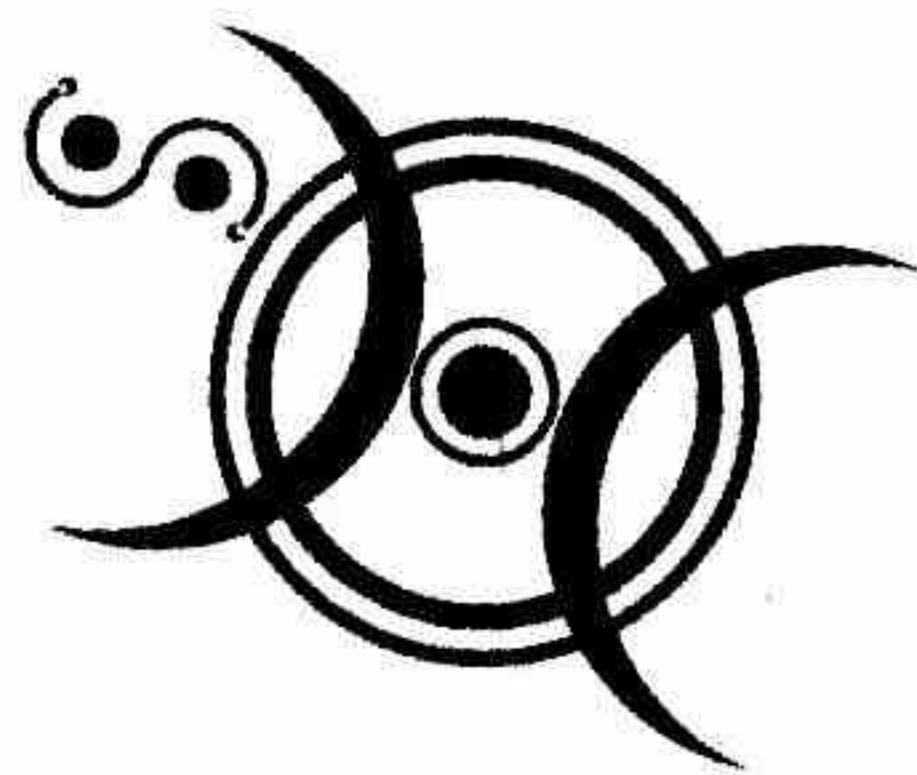


II. РАЗДЕЛ 8

Разработка типичного сайта

Глава 36.	Создание простейшего движка сайта
Глава 37.	Создание фотогалереи
Глава 38.	Гостевая книга
Глава 39.	Импорт новостей с чужого сайта в базу данных WordPress
Глава 40.	Создание счетчика сайта
Глава 41.	Статистика сайта
Глава 42.	Голосования и комментарии для сайта

ГЛАВА 36



Создание простейшего движка сайта

36.1. Планирование движка

36.1.1. Зачем нужно разрабатывать собственный движок?

Спрашивается, зачем нужно разрабатывать собственный движок сайта, если есть уже готовые решения, — например Joomla! или SLAED? Вполне может быть, что ни один "стандартный" движок вам не подошел — просто они не обладают нужными вам функциями. А, может, вы боитесь, что стандартные движки стандартно и взламываются. Причина может быть и другой — вам нужен простой и "легкий" движок.

ПРИМЕЧАНИЕ

Движком сайта также называют систему управления контентом сайта (CMS, Content Management System), поэтому иногда мы будем употреблять слово "движок", а иногда — "система".

Все эти причины имеют под собой почву. Стандартные движки на то и стандартные, их основная задача — удовлетворить большинство пользователей. Но довольно часто бывает так, что ни одна CMS не может выполнить поставленную перед ней задачу.

Для своего сайта я решил не использовать уже готовые решения, а написать движок самостоятельно. Почему? Во-первых, мне нравится сам процесс разработки. Но это, сами понимаете, не основная причина. Во-вторых, мне была нужна универсальная система, которая могла бы работать на почти любом хостинге. Первоначально мой сайт был размещен на бесплатном хостинге www.narod.ru, который не поддерживал PHP, поэтому ни о какой CMS и речи быть не могло. Весь сайт представлял собой набор HTML-файлов, связанных между собой перекрестными ссылками. Когда страниц было мало, поддержание сайта особого неудобства не доставляло. Но со временем оно стало занимать много времени, и пришло понимание необходимости автоматизировать процесс управления страницами. Понятно, что на HTML организовать автоматизацию невозможно. Нужен был PHP (или любой другой веб-

ориентированный язык программирования — например, Perl, но свой выбор я остановил на PHP). Исследовав предложения разных хостинг-провайдеров, я пришел к выводу, что далеко не всегда бесплатный хостинг поддерживает базу данных MySQL. Поэтому многие стандартные движки мне не подошли — ведь все они ориентированы на работу с MySQL. Было принято решение создать CMS, основанную на файлах (позже я приобрел нормальный хостинг — с PHP, MySQL и другими функциями, но при разработке CMS я о нем и не мечтал).

Да и, честно говоря, мне было не с руки создавать базу данных на основе своих страниц — ведь они все уже в готовом виде хранились на диске в формате HTML-файлов. Таким образом, мне нужна была CMS, ориентированная на работу без базы данных. Существовали и такие готовые решения, но, попробовав несколько из них, я понял, что будет проще и надежнее написать свою CMS, которую можно будет впоследствии развивать и дорабатывать.

Итак, с требуемой нестандартной функцией мы разобрались. Теперь приступим к обсуждению безопасности и надежности стандартных движков. С одной стороны, рекомендации по взлому того или иного движка (если взлом-таки возможен) легко найти в Интернете, поэтому вам придется постоянно следить за обновлениями движка — вдруг какое-то из них устраняет огромную "дыру" в системе безопасности вашего сайта. С другой стороны, ваш собственный движок — "темная лошадка" для неквалифицированных злоумышленников. Рекомендаций по его взлому, понятно, в Интернете не будет, а поэтому неквалифицированные злоумышленники, страдающие комплексом неполноценности, не смогут ничего сделать вашему сайту. Впрочем, квалификация разработчиков готовых CMS может оказаться значительно выше вашей, и есть вероятность того, что вы напишете самый "дырявый" движок в мире (и назовете его "Решето 1.0"). Квалифицированный злоумышленник (да, тот самый, который и написал рекомендации по взлому стандартных CMS) взломает ваше "Решето" за пару минут, если будет такая необходимость.

А теперь поговорим о третьей причине — о системных ресурсах. Не секрет, что большинство функций любой CMS попросту не используются. Как уже было отмечено, готовые CMS рассчитаны на удовлетворение потребностей разных пользователей. Кому-то нужен блок вывода рекламы, кому-то — нет. Кому-то нужна интеграция с форумом, а кому-то — не нужна. А иногда в состав CMS входит даже сам форум, который часто не используется, поскольку в нем нет никакой необходимости.

Все неиспользуемые функции попросту занимают место на хостинге, вынуждая пользователя покупать более дорогой хостинг. Например, для небольшого сайта вполне достаточно 50 Мбайт, но пользователь вынужден покупать хостинг на 100 Мбайт, потому что сама CMS занимает 30–50 Мбайт. Зачем платить больше?

Далее мы поговорим о функциях, которые собираемся реализовать в нашей CMS.

36.1.2. Необходимые нам функции движка

Успех любого дела зависит от хорошего плана. В этой главе мы определим нужные нам возможности движка сайта, а потом — реализуем их.

Основная функция любого движка — это вывод страниц сайта и управление ими. Учитывая, что хороший хостинг теперь стоит довольно дешево, страницы сайта будем хранить в базе данных MySQL.

Предположим, что мы создали базу данных и поместили в нее 100 страниц. Теперь нужно определиться, как наша CMS будет выводить эти страницы. Можно, конечно, вывести 100 ссылок — по одной на каждую страницу, но это очень неудобно. Надо хотя бы организовать постраничный вывод из базы. Правда, и это не панацея. Даже если выводить по 10 ссылок на страницу, то все равно получится уже 10 страниц.

Лучше всего разбить страницы на категории и организовать разделы сайта — по одному на каждую категорию. Например, вы создаете сайт по компьютерам. Тогда категориями станут компоненты компьютера: процессоры, материнские платы, жесткие диски, видеокарты и т. д. Понятно, что в такой базе данных будет информация не об одном процессоре, поэтому в категории "Процессоры" придется создать как минимум три подкатегории: "Процессоры Intel", "Процессоры AMD", "Процессоры прочие".

Под управлением страницами подразумевается добавление, редактирование и удаление страниц. Конечно, раз у нас появились подкатегории, то нужно обеспечить перемещение страницы из одной категории в другую. Все эти функции будут реализованы в панели управления сайтом.

Понятно, что для нашего сайта нужно разработать дизайн. Но дизайн — это штука не постоянная. Сегодня он один, а завтра вам захочется чего-то новенького. Соответственно, еще одна задача — разделить PHP-код CMS и HTML-код дизайна сайта. Другими словами, чтобы при изменении дизайна не пришлось редактировать CMS. Лучший способ разделения PHP-кода и HTML-кода — это шаблоны. Мы не станем использовать уже готовые шаблонизаторы типа Smarty — они сложны, тяжеловесны и не всегда надежно работают. Поэтому мы напишем свой простенький шаблонизатор, который будет выполнять только те функции, которые нам необходимы.

Если быть предельно точным, то мы уже спланировали почти всю CMS. Наша пока еще не созданная CMS "умеет" выводить страницы сайта, управлять ими и использует шаблоны для вывода этих самых страниц. Что ж, вполне неплохо. Теперь подумаем о дополнительных функциях нашей системы.

От постраничного вывода нам никуда не деться. Рано или поздно каждый раздел разрастется так, что без постраничного вывода ваш сайт будет неудобно читать.

Еще очень полезной является функция "Версия для печати", выводящая страницу сайта без обрамляющего ее дизайна, — так проще распечатать страницу, ведь на принтер будет выведено только содержимое страницы и ничего лишнего.

Крайне желательно, чтобы наша CMS для каждой страницы выводила ссылки **Вверх** и **Вниз** — чтобы перемещаться соответственно вверх и вниз по странице. Назовем данную функцию GotoTop.

На всякий случай желательно реализовать функцию вывода содержимого HTML-файла — она нам может понадобится.

Подумаем, что еще нам будет нужно. На первой странице сайта, как правило, размещают или информацию о компании, или новости. Новости можно искать и добавлять на сайт самостоятельно, а можно написать граббер новостей, который будет получать новости с других сайтов. Ранее мы уже рассмотрели пример RSS-граббера новостей (см. главу 32), позволяющего получить новости с RSS-ленты и вывести их в дизайне вашего сайта.

Вам нужна галерея, но места на хостинге мало? Тогда вы можете создать свою галерею на LiveJournal, а CMS будет выводить в дизайне сайта ваши фотографии, размещенные на LiveJournal, что очень удобно. То есть, у вас на сайте появится галерея, а физически фотографии будут размещены на сервере LiveJournal.

Перечислим список дополнительных функций нашей CMS:

- ☐ функция "Версия для печати";
- ☐ функция GotoTop;
- ☐ функция вывода содержимого HTML-файла;
- ☐ RSS-граббер новостей;
- ☐ интеграция с LiveJournal.

Пока мы не рассматриваем интеграцию с форумом, поскольку этому посвящен весь *раздел 9* этой книги. А далее мы рассмотрим схему нашей CMS.

36.1.3. Принципиальная схема движка

Итак, наша CMS состоит из следующих частей (рис. 36.1):

- ☐ базы данных — содержит все страницы нашего сайта и прочую информацию;
- ☐ панели управления — управляет страницами нашего сайта, а именно: позволяет создавать новые страницы, редактировать и удалять уже имеющиеся;
- ☐ движка — управляет показом страниц пользователю, навигацией по сайту, реализует дополнительные функции CMS;
- ☐ шаблонизатора — движок отправляет информацию шаблонизатору, который выводит ее в соответствии с созданными дизайнером HTML-шаблонами.

Как показано на схеме, двусторонний обмен данными может происходить между следующими объектами:

- ☐ базой данных и панелью управления — понятно, ведь панель управления позволяет просмотреть уже созданные страницы и отредактировать их, а также создать/удалить страницы сайта;
- ☐ базой данных и движком сайта — движок получает информацию из базы данных, передает ее шаблонизатору, а тот, в свою очередь, отправляет сгенерированный HTML-код в браузер пользователя. Пользователь может тоже отправлять информацию в базу данных — например, комментарии к страницам сайта. Однако общается он не напрямую с базой данных, а с движком сайта, который проверяет корректность данных (от проверки корректности почтового адреса

пользователя до проверки текста на цензуру) и помещает введенную пользователем информацию в базу данных.

Большинство уже готовых CMS имеют модульную структуру, дающую возможность использовать модули для добавления в CMS новых функций. У нашей CMS модулей не будет по следующим причинам:

- поскольку CMS разрабатывается для себя, не думаю, что ею заинтересуются тысячи пользователей. Следовательно, даже если добавить возможность использования модулей, она останется невостребованной;
- модульная структура необходима для быстрого и простого добавления нового кода в CMS, т. е. для пользователей, которые ничего не понимают в PHP-программировании. Поскольку мы сами напишем CMS, для нас не составит труда добавить в нее новую возможность, слегка отредактировав ее код.

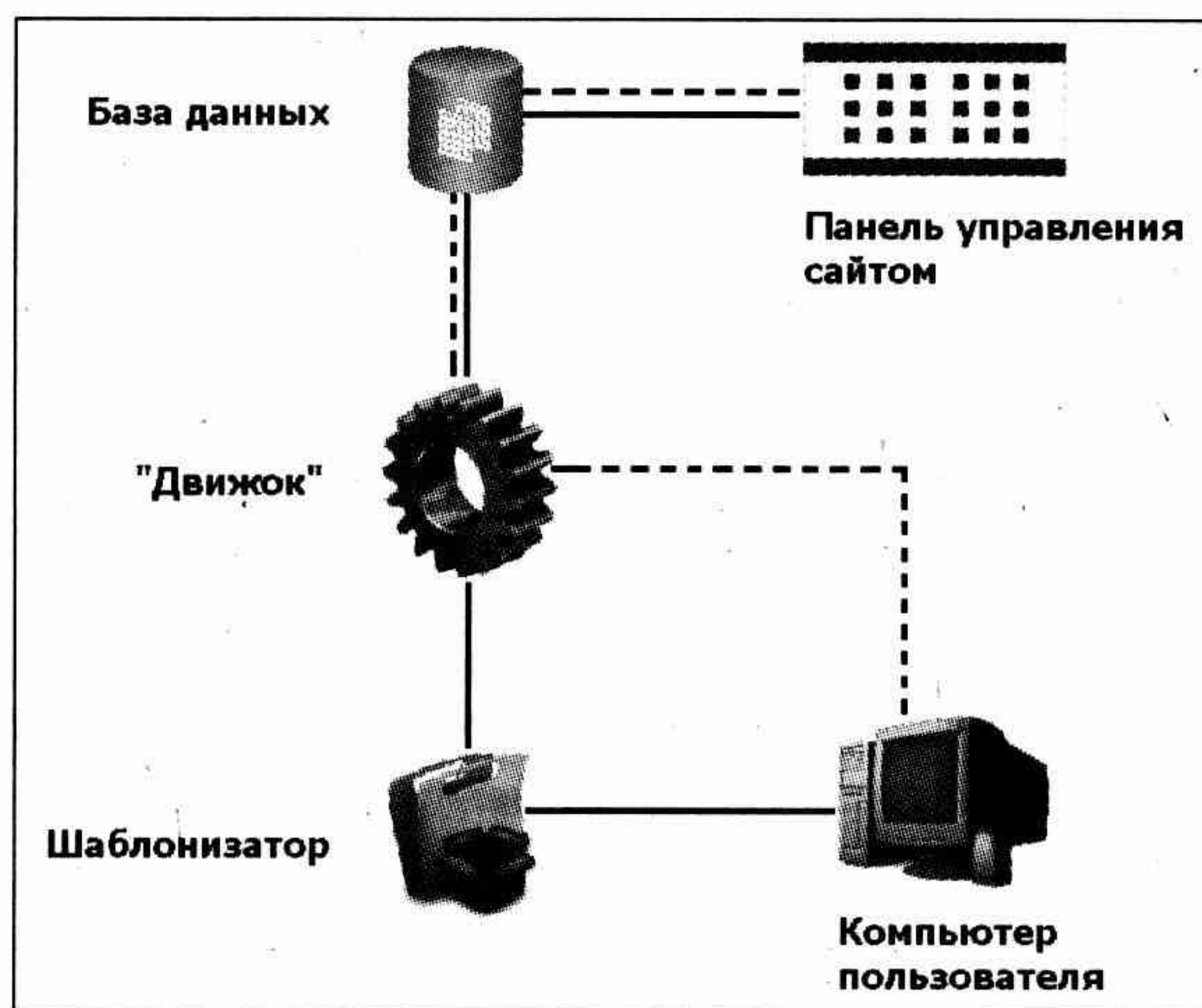


Рис. 36.1. Схема CMS

36.2. Основные функции движка

36.2.1. Разработка TPL-шаблона

В главе 22 мы разработали собственный шаблонизатор. Теперь пора создать TPL-шаблон, чтобы испытать шаблонизатор в действии. Мы создадим очень простой шаблон, содержащий мало HTML-кода, чтобы не усложнять нашу задачу его разбором. То есть фактически у нашего сайта дизайна как такового не будет. В реальных условиях вам (или вашему дизайнеру) нужно будет разработать TPL-шаблон на базе какой-нибудь HTML-страницы.

Наш шаблон мы назовем `page.tpl` — это будет единственный шаблон для всех страниц сайта (листинг 36.1).

Листинг 36.1. Шаблон page.tpl

```
<HTML>
<HEAD>
    <TITLE>{TITLE}</TITLE>
</HEAD>
<BODY>
    {DESCRIPTION}
    <TABLE BORDER=0 WIDTH=100%>
    <TR>
        <TD WIDTH=40% VALIGN=TOP>{MENU}</TD>
        <TD WIDTH=60% VALIGN=TOP>{PAGE}</TD>
    </TR>
    </TABLE>
    {INFO}
</BODY>
</HTML>
```

Как видите, шаблон очень прост, он выводит таблицу, содержащую меню и содержимое страницы. Назначение переменных шаблона следующее:

- ❑ {TITLE} — заголовок сайта;
- ❑ {DESCRIPTION} — описание (краткая информация о сайте);
- ❑ {MENU} — иерархическое меню сайта;
- ❑ {PAGE} — содержимое текущей страницы;
- ❑ {INFO} — некоторая информация, например информация об авторских правах.

Вы можете добавить в шаблон свои переменные, но пока это делать не стоит.

36.2.2. Файл настроек

Прежде чем приступить к написанию кода CMS, нам нужно создать файл настроек. В нем будут содержаться почти постоянные данные (константы, если хотите), например: имя сервера MySQL, имя базы данных, имя пользователя, пароль, информация о сайте. Согласитесь, данная информация очень редко изменяется.

Файл настроек назовем config.php — потом мы его будем подключать к нашей CMS. Код этого файла приведен в листинге 36.2.

Листинг 36.2. Файл config.php

```
<?

$dbhost = 'localhost';           // имя сервера БД
$dbname = 'db';                  // имя БД
$dbuser = 'user';                // имя пользователя БД
$dbpasswd = 'password';          // пароль пользователя
```



```
// информация о сайте
$title = 'CMS v0.01';
$description = 'Описание сайта';
$info = '<p>&copy; Денис Колисниченко';

?>
```

ПРИМЕЧАНИЕ

Приведенный здесь код вы найдете в файле *config.php* каталога *Glava_36* сопровождающего книгу электронного архива (см. приложение 4).

36.2.3. Основной файл CMS — index.php

Нам нужно разобраться, что будет делать этот файл. А делать он будет вот что:

- подключать файл настроек;
- подключать шаблонизатор;
- устанавливать соединение с сервером MySQL, используя параметры доступа из файла настроек;
- выводить содержимое выбранной пользователем страницы (точнее, передавать содержимое страницы шаблонизатору, который управляет выводом).

Первые три пункта ясны. Осталось определиться, как сценарий узнает, какую страницу выбрал пользователь. Я предлагаю использовать параметр-переменную *\$p*, которая будет указывать тип страницы, и параметр *\$id*, содержащий идентификатор выводимой страницы. В итоге URL в строке браузера примет вид: **http://сайт/index.php?p=тип&id=число**.

Если переменная *\$p* не задана, то выводится главная страница сайта. В этом случае URL станет выглядеть так: **http://сайт/index.php**.

Да, можно было бы упростить нашу CMS и передавать с помощью переменной *\$p* сразу идентификатор страницы. Но это не совсем удобно. Представим, что нам нужно отобразить страницы разных типов и для каждого типа использовать свое оформление. Или что нам нужно вывести информацию не из базы данных, а из HTML-файла или с другого сервера. Вот в этом случае нам и понадобится переменная *\$p*, которая будет указывать:

```
$p='db';           // вывод страницы из базы данных
$p='file';         // вывод страницы из HTML-файла
$p='server';       // вывод страницы с сервера
```

Теперь мы можем написать сценарий *index.php*. Первоначальный его вариант представлен в листинге 36.3 (обратите внимание на комментарии!).

Листинг 36.3. Первоначальный вариант index.php

```
<?php
```

```
// подключаем файл настроек
require "config.php";
```



```
// подключаемся к серверу БД
mysql_connect($dbhost,$dbuser,$dbpasswd);
mysql_select_db($dbname);
// подключаем шаблонизатор
require "template.php";

// открываем шаблон
$tpl->get_tpl('page.tpl');

// устанавливаем переменные шаблона
$tpl->set_value('TITLE',$title);
$tpl->set_value('DESCRIPTION',$description);
$tpl->set_value('INFO',$info);

// меню пока не будет
$menu = "скоро появится меню
<br>Пункт 1
<br>Пункт 2
<br>Пункт 3";

$tpl->set_value('MENU',$menu);

// переменная $p не установлена – нужно вывести главную страницу
if (!isset($p)) {

$page = "Главная страница";
$tpl->set_value('PAGE',$page);

}

// запускаем парсинг шаблона
$tpl->tpl_parse();

// выводим HTML
echo $tpl->html;
?>
```

Чтобы проверить, как работает этот сценарий, нужно скопировать файлы `index.php`, `template.php`, `config.php` и `page.tpl` в какой-нибудь каталог сервера и ввести в строку адреса браузера следующий URL: **`http://сервер/каталог/index.php`**.

Наш сценарий `index.php` ничего особенного не делает (рис. 36.2), но вас можно поздравить — вы заложили фундамент своей CMS. Теперь вам осталось сформировать меню по страницам сайта, хранящимся в базе данных, и вывести одну из страниц как переменную шаблона `{PAGE}`.

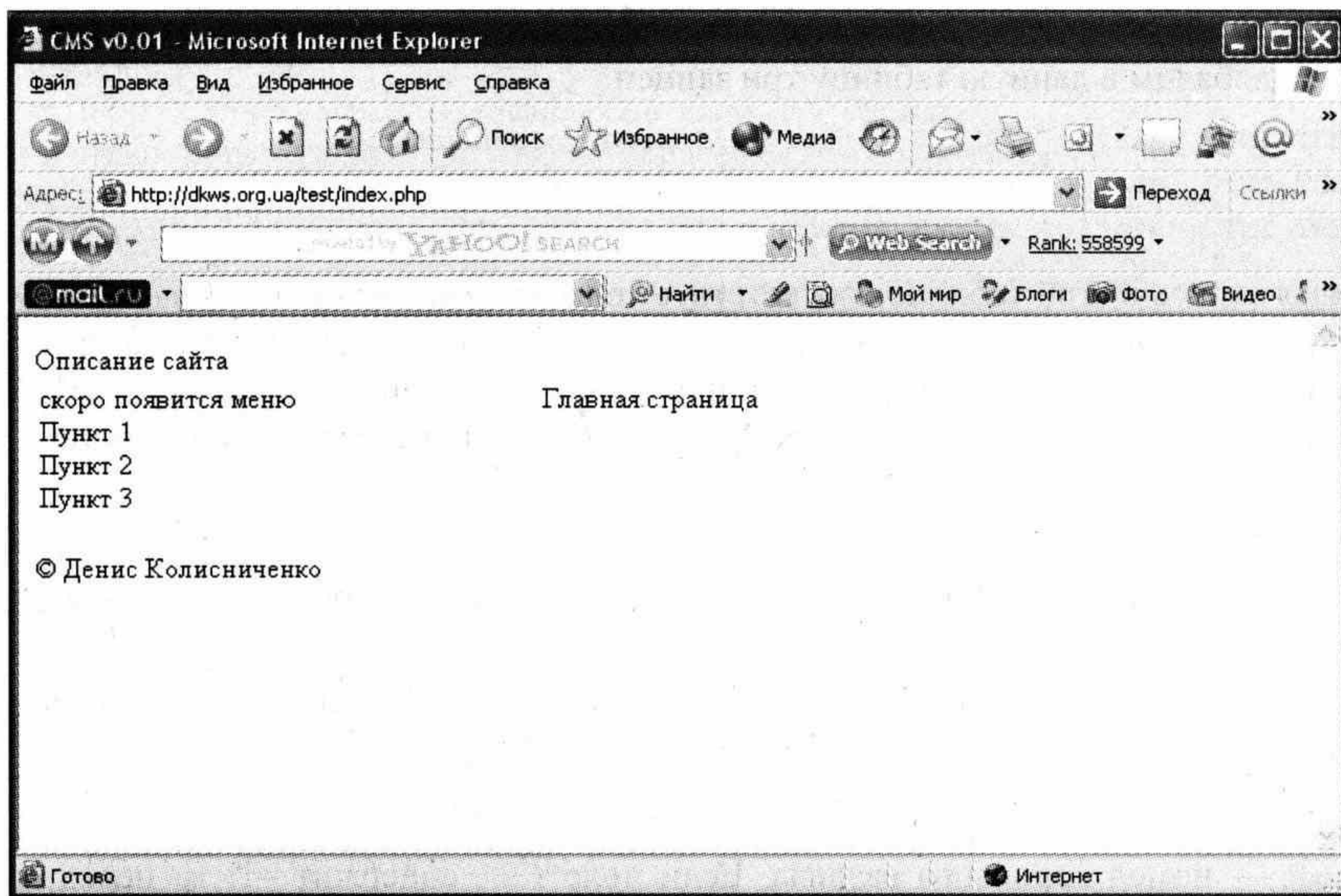


Рис. 36.2. Вывод сценария `index.php`

36.2.4. Проектирование базы данных

Наш движок будет хранить информацию в трех таблицах:

- `static` — статическая информация — например: главная страница, контакты и т. д.;
- `cats` — таблица категорий (разделов сайта). По этой таблице будет формироваться иерархическое меню сайта;
- `pages` — таблица страниц, содержит страницы сайта.

Таблица *static*

Таблица `static` — самая простая. Она содержит всего два поля: `id` (идентификатор страницы) и `content` (содержимое страницы).

Чтобы создать эту таблицу, нужно выполнить следующий SQL-запрос:

```
CREATE TABLE `static` (  
  `id` varchar(15) NOT NULL,  
  `content` text,  
  PRIMARY KEY (`id`)  
) DEFAULT CHARSET=cp1251;
```


ПРИМЕЧАНИЕ

Этот SQL-запрос можно ввести в программе phpMyAdmin, описание которой вы найдете в приложении 1. Основы SQL изложены в главе 18. Код самого запроса содержится в файле *db.sql*, находящемся в том же каталоге электронного архива, — необходимо только скопировать нужные строки. Можно даже импортировать сразу весь файл *db.sql*, но в целях углубленного понимания примера я это делать не рекомендую.

Теперь добавим в данную таблицу три записи:

```
INSERT INTO `static` VALUES ('main', 'Главная страница');
INSERT INTO `static` VALUES ('prices', 'Прайс-листы');
INSERT INTO `static` VALUES ('contacts', 'Контакты');
```

Каждая запись в этой таблице соответствует одной странице сайта. В нашем случае это три страницы: Главная страница, Прайс-листы и Контакты. Конечно, на практике содержимое таких страниц будет представлять собой полноценную HTML-страницу, но не вижу особого смысла усложнять код примера на данном этапе.

Таблица *cats*

Таблица *cats* (сокращение от англ. *categories* — ничего общего с котами!) будет содержать четыре поля:

- ❑ *id* — номер раздела. Поле автоматически увеличивается (*auto_increment*), чтобы нам было удобнее создавать разделы;
- ❑ *title* — название раздела (будет отображено в меню сайта);
- ❑ *root* — номер корневого раздела. Если поле *root* содержит *-1*, то перед нами корневой раздел. Если же поле *root* содержит положительное значение, то перед нами — подраздел, а поле *root* указывает номер его родительского раздела. Значение по умолчанию для этого поля *-1*, т. е. если поле *root* не задано, мы подразумеваем, что будет создан корневой (родительский) раздел;
- ❑ *txt* — описание раздела, может содержать любую информацию.

Структура таблицы *cats* показана на рис. 36.3.

	Поле	Тип	Сравнение	Атрибуты	Ноль	По умолчанию	Дополнительно
☐	id	int(11)			Нет		auto_increment
☐	title	varchar(100)	cp1251_general_ci		Да	NULL	
☐	root	int(11)			Да	-1	
☐	txt	text	cp1251_general_ci		Да	NULL	

Рис. 36.3. Структура таблицы *cats*

Запрос для создания этой таблицы выглядит так:

```
CREATE TABLE `cats` (
  `id` int(11) NOT NULL auto_increment,
  `title` varchar(100) default NULL,
  `root` int(11) default '-1',
```



```
`txt` text,
PRIMARY KEY (`id`)
) DEFAULT CHARSET=cp1251;
```

Теперь добавим в нашу таблицу несколько разделов:

```
INSERT INTO `cats` VALUES (1, 'Дисковые пилы для стали', -1, 'Информация
о разделе');
INSERT INTO `cats` VALUES (2, 'Дисковые пилы для алюминия', -1, NULL);
INSERT INTO `cats` VALUES (3, 'Линии обработки стальных профилей', -1, NULL);
INSERT INTO `cats` VALUES (4, 'Центры обработки листового металла', -1, NULL);
INSERT INTO `cats` VALUES (5, 'Отрезные станки', 3, NULL);
INSERT INTO `cats` VALUES (6, 'Линии сверления профилей', 3, NULL);
INSERT INTO `cats` VALUES (7, 'Роботы для обработки профилей', 3, NULL);
```

Первые четыре раздела являются корневыми, т. е. родительскими. Разделы 5–7 это подразделы раздела с номером 3.

Таблица *pages*

Таблица *pages*, как и таблица *cats*, тоже состоит из четырех полей:

- *id* — идентификатор (номер) страницы;
- *cat* — номер раздела, в который помещена страница;
- *header* — заголовок страницы (будет выводиться в меню);
- *content* — содержимое страницы.

	Поле	Тип	Сравнение	Атрибуты	Ноль	По умолчанию	Дополнительно
□	<i>id</i>	int(11)			Нет		auto_increment
□	<i>cat</i>	int(11)			Нет		
□	<i>header</i>	varchar(250)	cp1251_general_ci		Да	NULL	
□	<i>content</i>	text	cp1251_general_ci		Да	NULL	

Рис. 36.4. Структура таблицы *pages*

Структура таблицы приведена на рис. 36.4, а для ее создания используется следующий SQL-запрос:

```
CREATE TABLE `pages` (
  `id` int(11) NOT NULL auto_increment,
  `cat` int(11) NOT NULL,
  `header` varchar(250) default NULL,
  `content` text,
  PRIMARY KEY (`id`)
) DEFAULT CHARSET=cp1251;
```

Для добавления новых страниц служит следующий SQL-оператор:

```
INSERT INTO `pages` VALUES (0, номер, 'заголовок', 'содержимое');
```


Номер страницы можно не указывать (0), поскольку значение поля будет автоматически увеличено. Но не забывайте при добавлении страницы указывать номер раздела, к которому принадлежит страница, а также заголовок и содержимое страницы.

36.2.5. Иерархическая структура сайта

Алгоритм работы меню

Сейчас нам предстоит самое сложное — реализовать вывод меню. Если переменная `$p` не указана, то будут выведены родительские разделы меню и главная страница, как показано на рис. 36.5 (разрабатываемая в книге CMS уже успешно используется на некоторых сайтах).



Рис. 36.5. Вывод родительских разделов меню и главной страницы

Пусть посетитель щелкнул на одном из разделов сайта. Далее поворот событий зависит от того, есть ли у раздела подразделы. Если подразделов нет, то будут выведены название раздела, список страниц и информация о разделе (как мы помним, поле `txt` таблицы `cats` может содержать произвольный HTML-код) — все это изображено на рис. 36.6.

Если же в разделе имеются подразделы, то будет выведен список подразделов и информация о разделе (рис. 36.7). На рис. 36.7 информация о разделе не заполнена.

Обычно если у раздела есть подразделы, то такой раздел не содержит страниц. Не следует представлять разделы сайта как каталоги файловой системы, а страницы — как файлы. Это в каталогах могут быть как подкаталоги, так и файлы. В случае с разделами сайта все иначе. Предположим, вы делаете каталог продукции фирмы. У вас есть раздел "Процессоры", а в нем три подраздела: "Процессоры AMD", "Процессоры Intel" и "Процессоры прочие". Следовательно, все страницы, описывающие процессоры разных производителей, будут помещены в соответ-

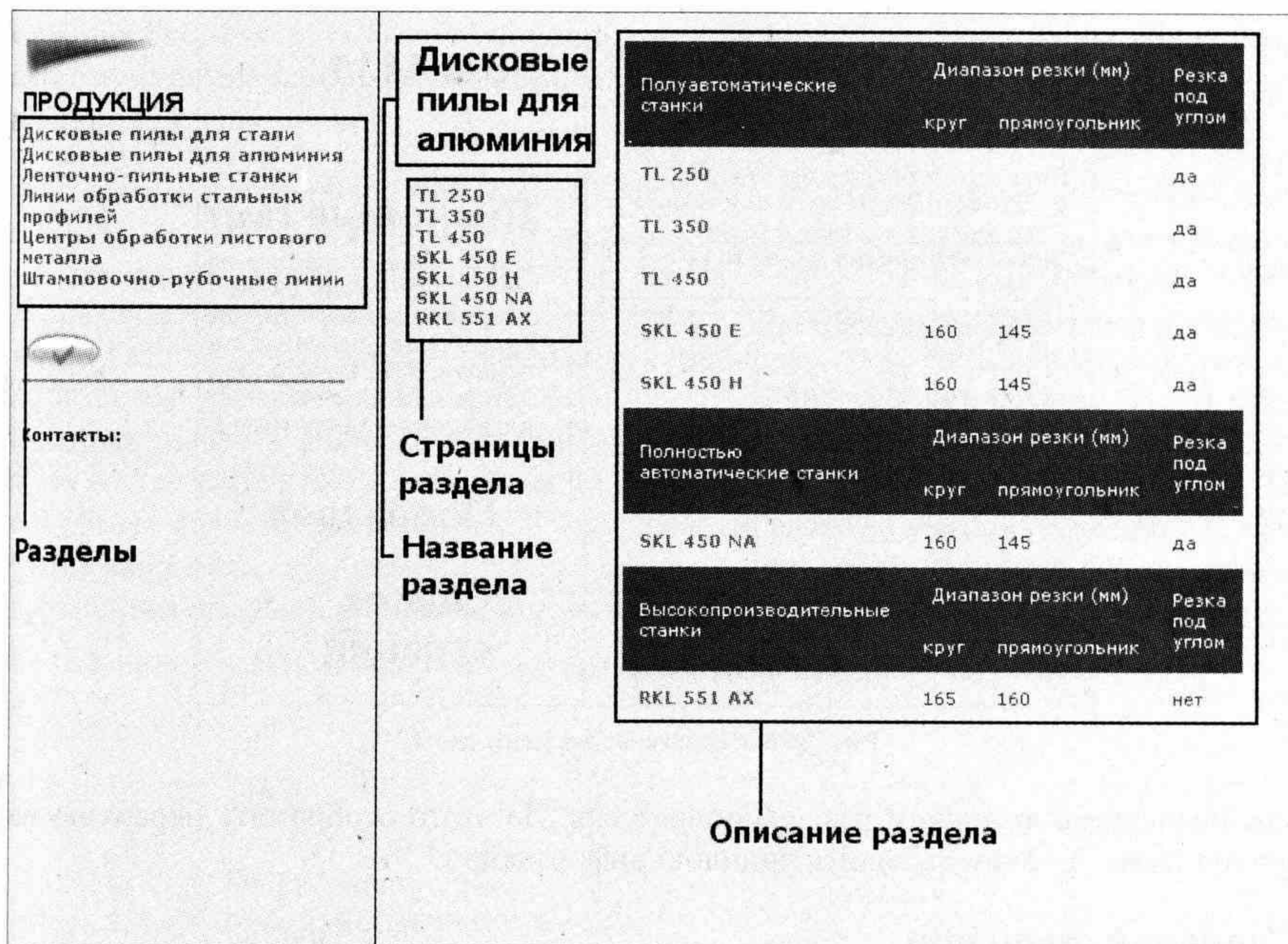


Рис. 36.6. Вывод содержимого раздела

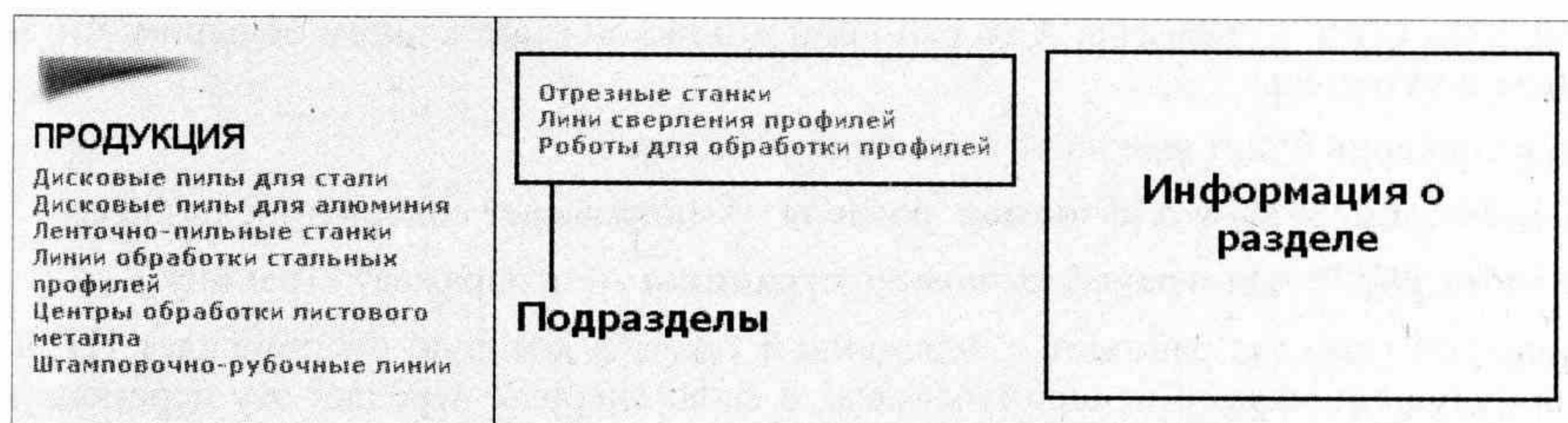


Рис. 36.7. Вывод раздела, содержащего подразделы

вующий подраздел. Иного не дано. Поэтому если раздел содержит подразделы, то не нужно искать в базе данных страницы, которые относятся непосредственно к самому разделу. Это особенность нашей разработки. Конечно, вы можете немного изменить приведенный в книге пример и сделать так, чтобы разделы вашей CMS могли содержать и обычные страницы, и подразделы. То есть, чтобы разделы и страницы сайта были похожи на файлы и каталоги файловой системы, следует модифицировать код CMS (он будет представлен далее).

Итак, когда пользователь откроет страницу, меню станет иерархическим (рис. 36.8), т. е. будут отображены разделы и подразделы (если это нужно), а также содержи-

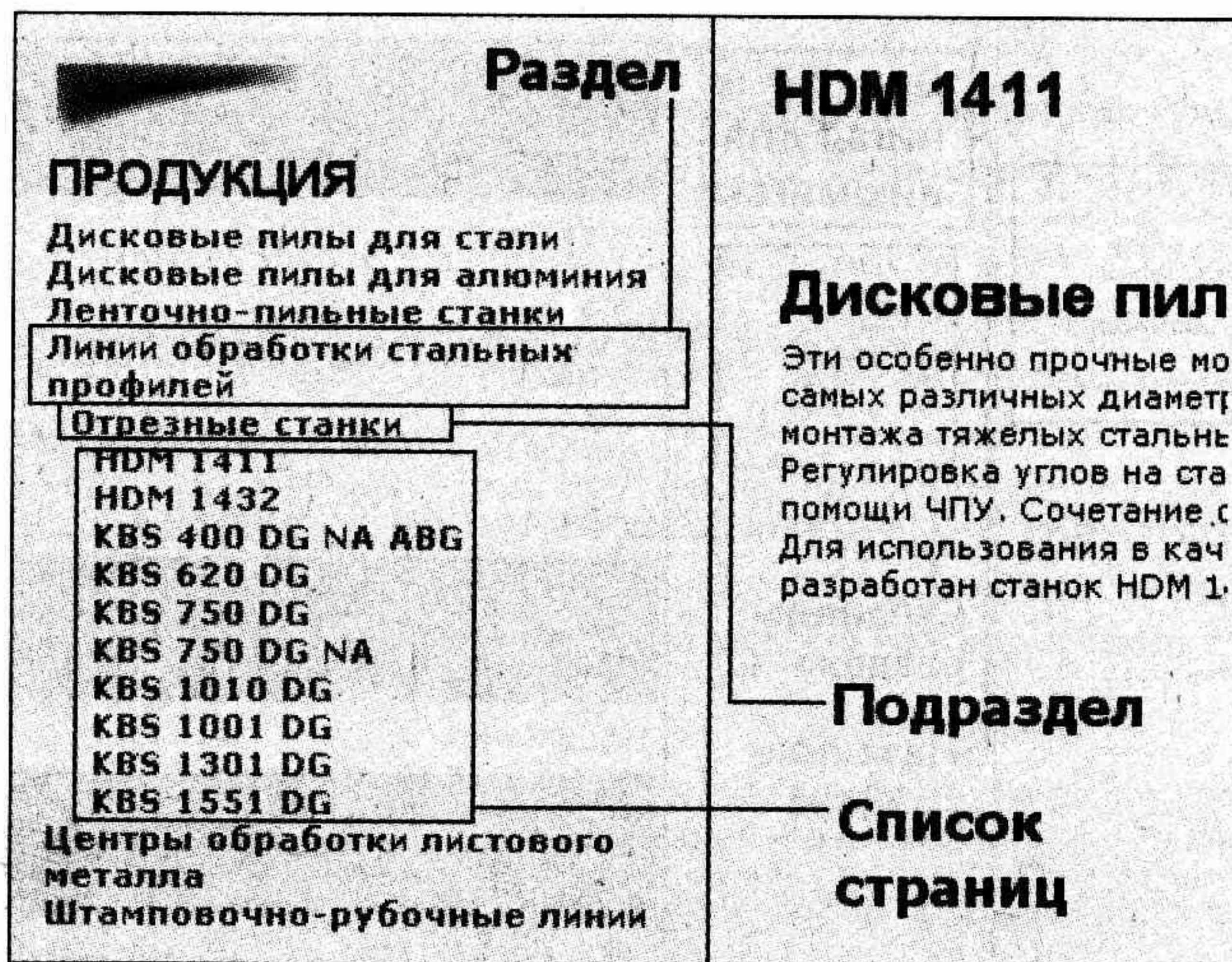


Рис. 36.8. Иерархическое меню сайта

мое последнего активного раздела/подраздела. До этого отображать иерархию меню нет смысла, зачем выводить лишнюю информацию?

Сценарий menu.php

Теперь, когда нам ясен алгоритм работы меню, напомним сценарий menu.php, который будет его выводить. Отдельный сценарий нужен, чтобы не перегружать основной файл CMS — index.php. А то рано или поздно он станет таким большим, что вы в нем запутаетесь.

Наш сценарий будет выводить ссылки двух видов:

- ❑ `index.php?p=show&id=номер_раздела` — показывает содержимое раздела;
- ❑ `index.php?p=showpage&id=номер_страницы` — отображает страницу.

Сценарий menu.php работает с переменной \$menu, в которую он помещает HTML-код будущего меню. Сценарий index.php, в свою очередь, передает эту переменную шаблонизатору.

Мы воспользуемся несколькими служебными переменными:

- ❑ \$sub — если данная переменная принимает значение 1, то выводимый раздел является подразделом;
- ❑ \$is_ext_menu — если данная переменная равна 1, то нужно вывести иерархическое меню.

Первым делом наш сценарий получает список корневых разделов и помещает его в переменную результата \$r:

```
$q = "select * from cats where root=-1";
$r = mysql_query($q);
```


Затем необходимо установить переменную `$is_ext_menu`, поскольку мы подразумеваем, что сначала нам не нужно раскрывать меню:

```
$is_ext_menu = 0; // не раскрывать меню
```

Мы с вами договорились, что меню будет раскрыто только во время отображения страницы (когда переменная `$p` принимает значение `'showpage'`). Следовательно, наш сценарий должен выполнить некоторые подготовительные действия перед отображением страницы, а именно: определить номер раздела выводимой страницы (номер раздела, к которому принадлежит страница) и выяснить, является ли раздел подразделом. Выяснить это очень легко, когда мы знаем номер раздела. Тогда нужно найти значение поля `root` таблицы `cats` для нашего раздела. Если поле `root` равно `-1`, то перед нами родительский раздел, а если же поле `root` больше `-1`, то наш раздел является подразделом. В случае, если раздел является подразделом, нам нужно запомнить его номер (переменная `$orig`) и номер родительского раздела (переменная `$number`), а также установить переменную `$sub` в `-1`.

Все сказанное можно реализовать с помощью следующего кода:

```
if ($p=="showpage") {

    // узнаем номер раздела выводимой страницы (поле cat)
    $qur = "select * from pages where id=$id";
    $res = @mysql_query($qur);
    $row = @mysql_fetch_array($res);
    // номер раздела
    $number = $row[cat];

    // нужно раскрыть меню — мы выводим страницу
    $is_ext_menu = 1;

    // если переменная $sub = 1, то перед нами — подраздел
    $sub = 0;

    // проверяем, является ли раздел подразделом
    $qur2 = "select * from cats where id=$number";

    $res2 = mysql_query($qur2);
    $row2 = mysql_fetch_array($res2);

    // получаем значение поля root и анализируем его
    $root = $row2[root];

    if ($root > -1) {

        // наш раздел является подразделом, поэтому нужно
        // установить переменные:
        // $orig — номер исходного раздела
```



```
// $number — номер корневого раздела
// $sub=1 — признак подраздела
$orig=$number; $number=$root; $sub=1;

}
```

```
}
```

После выполнения подготовительных действий мы можем с чистой совестью вывести корневые (родительские) разделы с помощью следующего кода:

```
while ($row = mysql_fetch_array($r)) {

    $menu = $menu . "<p><a
href=index.php?p=show&id=$row[id]>$row[title]</a>";
```

```
// далее будет продолжение кода
```

Этот код выводит ссылки первого вида:

```
<a href=index.php?p=show&id=номер_раздела> название_раздела </a>
```

Но одним выводом родительских разделов мы не ограничимся. Нужно проанализировать переменную `$is_ext_menu`. Ведь, возможно, пользователь открыл страницу сайта, и теперь нам нужно вывести иерархическое меню, т. е. отобразить список страниц раздела/подраздела и название самого раздела/подраздела. Все сказанное реализуется с помощью следующего кода:

```
// если нужно раскрыть меню
if ($is_ext_menu == 1) {

    if ($row[id]==$number) {

        // если перед нами подраздел
        if ($sub==1) {

            // выводим имя подраздела
            $qur3 = "select * from cats where id=$orig";
            $res3 = mysql_query($qur3);

            $row3 = mysql_fetch_array($res3);

            $menu = $menu .
                "<br>&nbsp;&nbsp;<a
href=index.php?p=show&id=$row[id]>$row3[title]</a>";

        }

    }

}
```

```
// получаем и выводим список страниц раздела
if ($sub==1) $number=$orig;
$qur = "select * from pages where cat=$number";
$res = mysql_query($qur);
```


В листинге 36.4 представлен полный текст файла `menu.php`. Чтобы листинг был менее громоздким, я удалил все комментарии, которые были приведены ранее.

```
<?
$q = "select * from cats where root=-1";
$r = mysql_query($q);

$sis_ext_menu = 0;

if ($p=="showpage") {

    $qur = "select * from pages where id=$id";
    $res = @mysql_query($qur);
    $row = @mysql_fetch_array($res);

    $number = $row[cat];
    $sis_ext_menu = 1;
    $sub = 0;
    $qur2 = "select * from cats where id=$number";

    $res2 = mysql_query($qur2);
    $row2 = mysql_fetch_array($res2);

    $root = $row2[root];

    if ($root > -1) { $orig=$number; $number=$root; $sub=1; }

}

while ($row = mysql_fetch_array($r)) {

    $menu = $menu . "<p><a
href=index.php?p=show&id=$row[id]>$row[title]</a>";
```



```

if ($is_ext_menu == 1) {

if ($row[id]==$number) {

if ($sub==1) {

$qur3 = "select * from cats where id=$orig";
$res3 = mysql_query($qur3);

$row3 = mysql_fetch_array($res3);

$menu = $menu . "<br>&nbsp;&nbsp;<a
href=index.php?p=show&id=$row[id]>$row3[title]</a>";
}

if ($sub==1) $number=$orig;

$qur = "select * from pages where cat=$number";
$res = mysql_query($qur);
while ($row = mysql_fetch_array($res))
    $menu = $menu .
"<br>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href=index.php?p=showpage&id=$row[id]>$row[header]</a>";
}
} // if ($is_ext_menu == 1)

}
?>

```

ПРИМЕЧАНИЕ

Приведенный здесь код вы найдете в файле *menu.php* каталога *Glava_36* сопровождающего книгу электронного архива (см. приложение 4).

Теперь нам нужно изменить файл *index.php*. Откройте его и найдите оператор присваивания:

```

$menu = "скоро появится меню
<br>Пункт 1
<br>Пункт 2
<br>Пункт 3";

```

Замените его оператором `include`:

```
include "menu.php";
```

Запустите файл *index.php*. Наша CMS уже отображает корневые разделы меню (рис. 36.9), но еще не готова к выводу содержимого раздела/страницы — вместо этого отображается только строка {PAGE}.

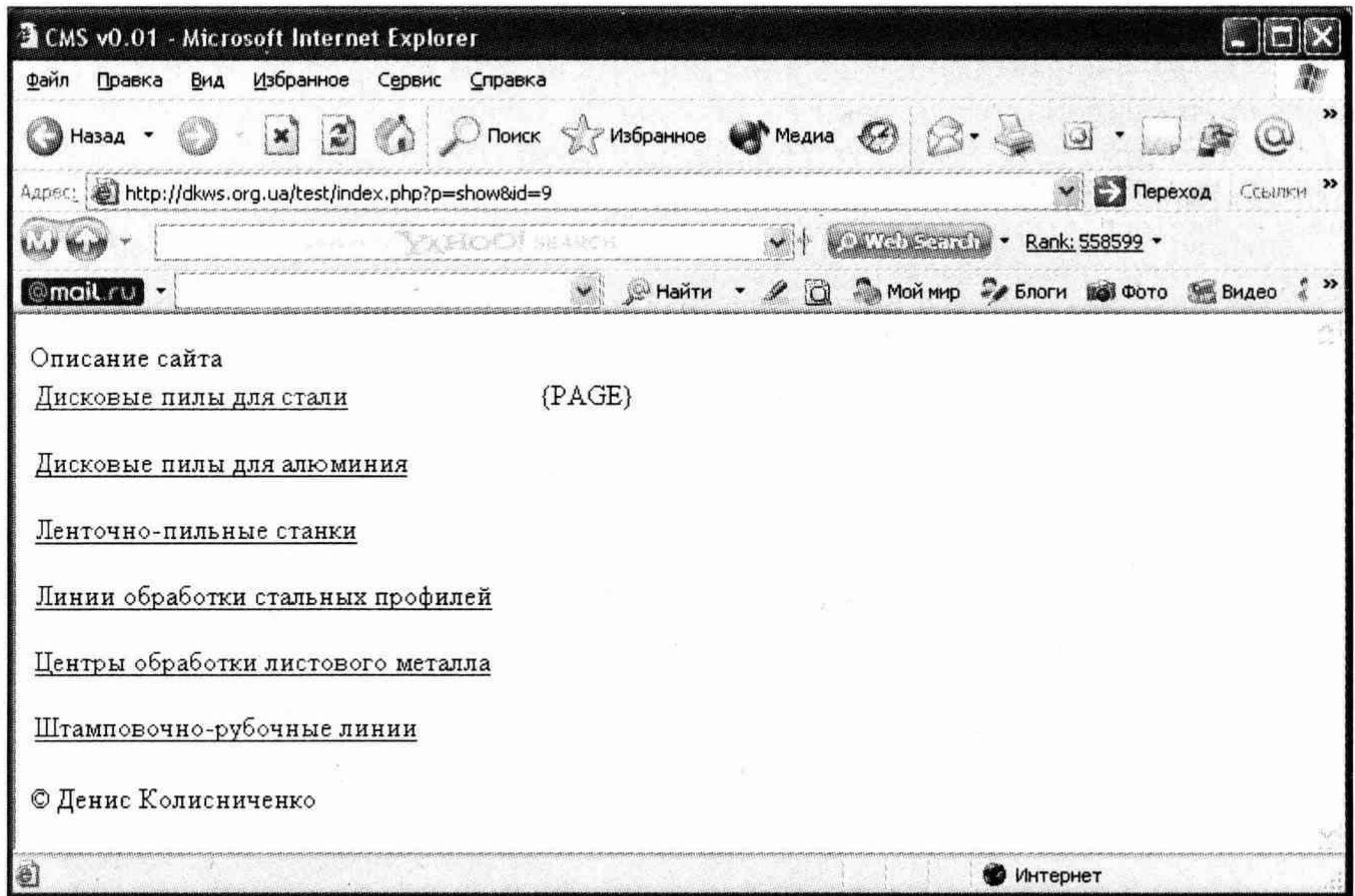


Рис. 36.9. CMS пока не готова к выводу содержимого раздела/страницы

Вывод содержимого раздела и страницы

Сценарий `index.php` должен реагировать на значения `'show'` и `'showpage'` переменной `$p`. Сначала реализуем вывод содержимого раздела. Откройте файл `index.php` и найдите следующий фрагмент кода:

```
if (!isset($p)) {

$page = "Главная страница";
$tpl->set_value('PAGE', $page);

}
```

Сразу после него добавьте реакцию на значение `'show'` переменной `$p` (листинг 36.5).

Листинг 36.5. Вывод содержимого раздела

```
elseif ($p=="show") {

// выводим содержимое раздела

// удаляем из $id все, что может представлять опасность
// в последующих главах вы узнаете, зачем мы это сделали
```



```

$id = @htmlspecialchars($id);
$id = @strip_tags($id);

// ищем подразделы, т. е. разделы, у которых поле root
// равно номеру выводимого раздела
$query = "select * from cats where root=$id";
$result = mysql_query($query);
if (mysql_num_rows($result)==0) {
// подразделов нет, выводим страницы
    $q2 = "select * from cats where id=$id";
    $res2 = mysql_query($q2);

    $q3 = "select * from cats where id=$id";
    $res3 = mysql_query($q3);

    $row3 = mysql_fetch_array($res3);

    $info = $row3[txt];

    $page = $page . "<table width=100% border=0>
        <td valign=top width=40%>";

    if (mysql_num_rows($res2)>0) {

        $row2 = mysql_fetch_array($res2);
        $page = $page . "<h1>$row2[title]</h1><p>";

        $q = "select * from pages where cat=$id";

        $res = mysql_query($q);

        while ($row = mysql_fetch_array($res))

            $page = $page .
                "<br><b>
                <a href=index.php?p=showpage&id=$row[id]>
                $row[header]</a></b>";

// поле info. — это информация о разделе
    $page = $page . "</td><td valign=top>$info</td></table>";
    }
    else $page = $page . "<h1>Нет такого раздела!</h1>";
}

// есть подразделы, выводим их
while ($row = mysql_fetch_array($result))
    $page = $page . "<br><a href=index.php?p=show&id=$row[i
        $row[title]>";
}

```


Приведенный код проверяет наличие у выводимого раздела подразделов. Если подразделов нет, то выводится список страниц раздела и информация о разделе. Если же подразделы есть, то выводится только список подразделов. Выводить информацию о разделе я не стал, чтобы не усложнять код. В случае необходимости вы можете добавить эту возможность самостоятельно. На рис. 36.10 показано, как наша CMS выводит содержимое раздела — список страниц и информацию о разделе.

Теперь напишем код вывода страницы. По сравнению с кодом вывода раздела он будет довольно прост. Заодно мы реализуем функцию `GotoTop`, добавив ссылки-якоря для перехода вниз и вверх по документу. Нужный нам код представлен в листинге 36.6. Этот код нужно добавить сразу после последней закрывающей фигурной скобки сценария `index.php`.

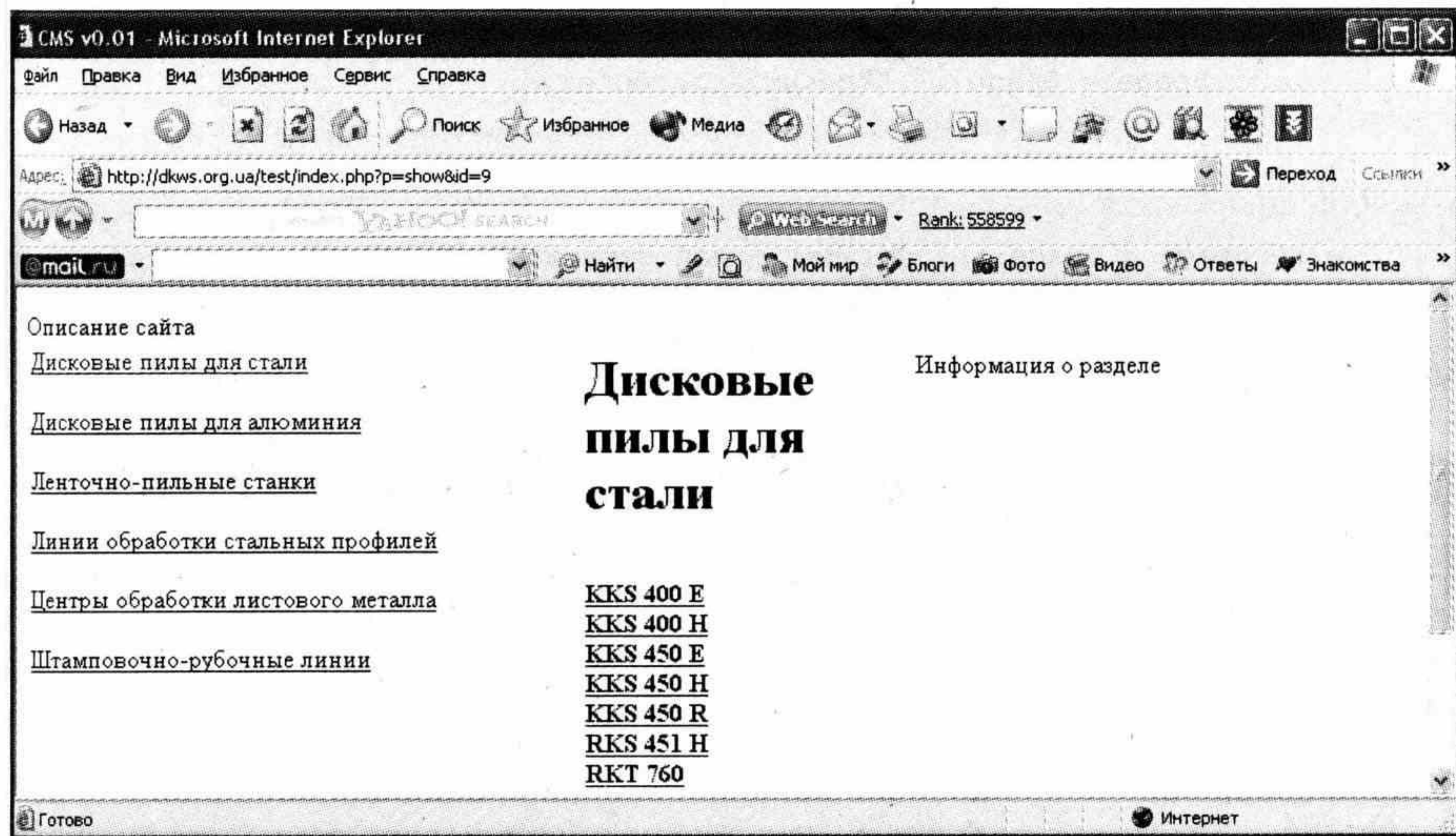


Рис. 36.10. Вывод раздела сайта

Листинг 36.6. Код вывода страницы

```
elseif ($p=="showpage") {

$id = @htmlspecialchars($id);
$id = @strip_tags($id);

$q = "select * from pages where id=$id";
$r = mysql_query($q);

    if (mysql_num_rows($r)>0) {
        $row = mysql_fetch_array($r);
```



```

// выводим название страницы
$page = $page . "<h1>$row[header]</h1>";

// выводим ссылку "Вниз"
$page = $page .
    "<p><center><a name=top></a>
    <a href=#down>Вниз</a></center><p>";

// выводим содержимое страницы
$page = $page . "<p><br><br>$row[content]";

// выводим ссылку на версию для печати
$page = $page . "<p><p><a target=_blank
    href=print.php?id=$id>Версия для печати</a>";

// выводим ссылку "Вверх"
$page = $page . "<p><br><p><center>
    <a name=down></a><a href=#top>Наверх</a></center>";
}
else $page = $page . "<h1>Нет такой страницы!</h1>";
}

$tpl->set_value('PAGE',$page);

```

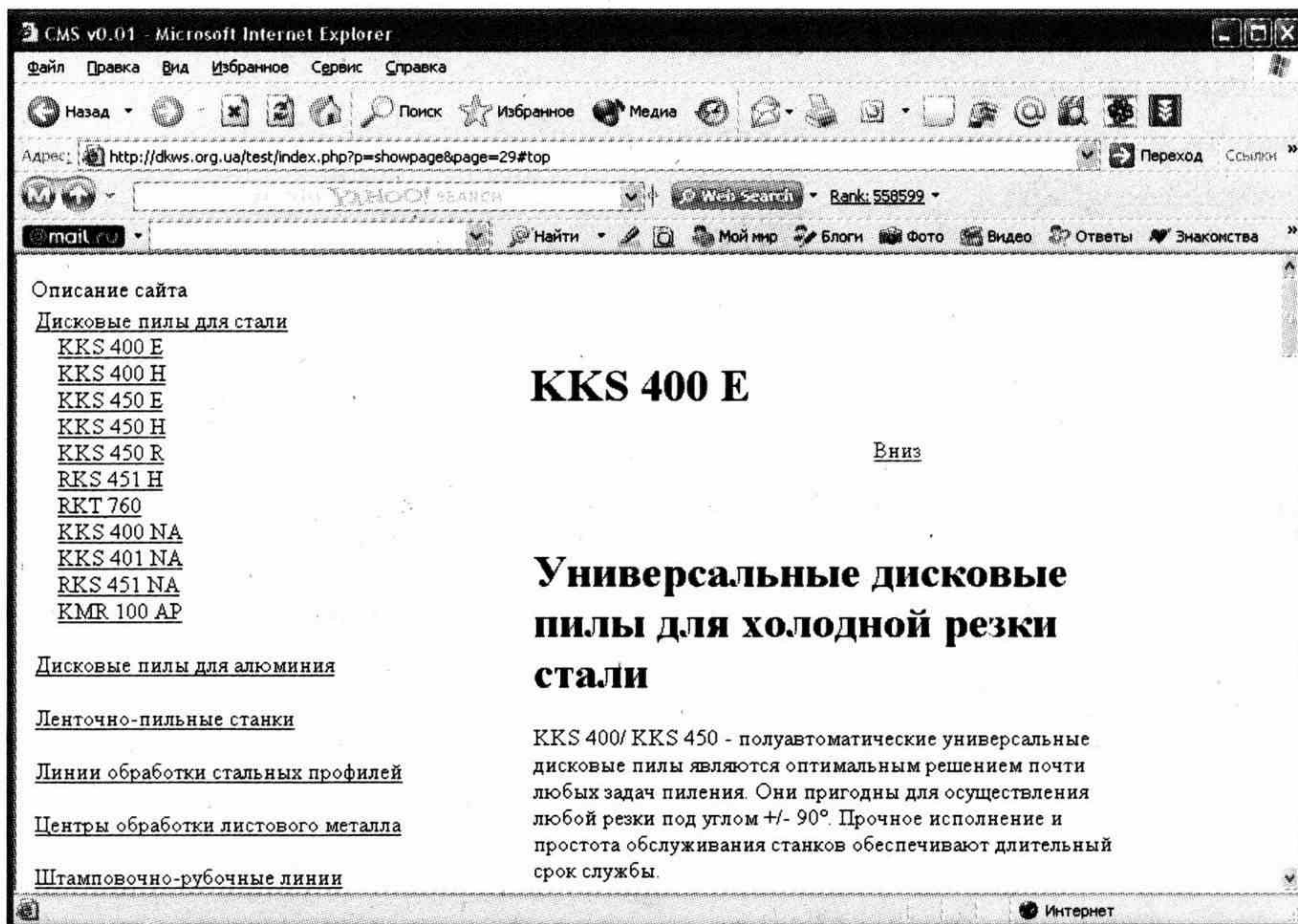


Рис. 36.11. Меню готово

Последняя строка кода необходима для установки переменной шаблона {PAGE} по переменной сценария \$page. Результат вывода страницы представлен на рис. 36.11. Как видно из этого рисунка, наш сценарий раскрыл меню и отобразил список страниц раздела.

36.3. Дополнительные функции движка

36.3.1. Вывод информации из таблицы *static*

Мы совершенно забыли о таблице *static* и трех страницах (вы можете добавить в нее столько страниц, сколько вам захочется), содержащихся в ней.

Откройте шаблон *page.tpl* и добавьте три следующие ссылки:

```
<a href=index.php?p=static&id=main>Главная</a>
<a href=index.php?p=static&id=prices>Прайсы</a>
<a href=index.php?p=static&id=contacts>Контакты</a>
```

Место добавления этих ссылок особого значения не имеет, но желательно добавить их после строки {DESCRIPTION}.

ПРИМЕЧАНИЕ

Полный код этого шаблона вы найдете в файле *page.tpl* каталога *Glava_36* сопровождающего книгу электронного архива (см. приложение 4).

Обратите внимание на формат ссылки: *index.php?p=static&id=имя*.

Имя — это идентификатор страницы в таблице *static*. Теперь осталось добавить код отображения страницы из таблицы *static*. Для этого после последней закрывающей фигурной скобки файла *index.php* нужно добавить следующий код:

```
elseif ($p=="static") {

    $id = @htmlspecialchars($id);
    $id = @strip_tags($id);

    $q = "select, * from static where id=\"\$id\" limit 1";
    $r = mysql_query($q);

    $row = mysql_fetch_array($r);
    $page = $row[content];

}
```

Но это еще не все. Найдите следующий фрагмент кода:

```
if (!isset($p)) {

    $page = "Главная страница";
    $tpl->set_value('PAGE', $page);

}
```


Замените его следующим фрагментом:

```
if (!isset($p)) {  
    $q = "select * from static where id=\"main\" limit 1";  
    $r = mysql_query($q);  
    $row = mysql_fetch_array($r);  
    $page = $row[content];  
}
```

Вот теперь все в порядке — если не указана переменная `$p`, будет выведено содержимое страницы с идентификатором 'main', то есть отображена главная страница. Поскольку страницы из таблицы `static` не относятся ни к одному разделу, меню сайта при выводе таких страниц раскрыто не будет.

Полную версию файла `index.php` (со всеми изменениями) вы найдете в каталоге `Glava_36` сопровождающего книгу электронного архива (см. приложение 4).

ПРИМЕЧАНИЕ

Поскольку на протяжении этой главы мы будем изменять файл `index.php`, наращивая возможности CMS, то версия файла с поддержкой таблицы `static` будет называться `index-static.php`. И чтобы вы не запутались в ссылках на сопровождающий архив, в разд. 36.4 все они будут систематизированы.

36.3.2. Функция вывода содержимого HTML-файла

Иногда бывает нужно вывести содержимое какого-нибудь HTML-файла. Реализовать функцию вывода HTML-файла довольно просто. Прежде чем приступить к написанию кода, поговорим о формировании ссылок для вызова данной функции. Вот пример таких ссылок:

`index.php?p=file&f=about`

`index.php?p=file&f=text/about`

В первом случае будет открыт файл `about.html`, который находится в одном каталоге с файлом `index.php`, а во втором случае будет открыт файл `about.html` из каталога `text`. Обратите внимание — расширение файла указывать не нужно. Это сделано из соображений безопасности, чтобы злоумышленник не смог с помощью этой функции открыть в системе произвольный файл.

Как злоумышленник может открыть любой файл? Дело в том, что ваш сценарий `index.php` находится на жестком диске веб-сервера. Пусть этот файл лежит в каталоге `/var/www/client01/html`. При неправильной настройке веб-сервера (если у администратора, извините, кривые руки) наш сценарий может получить доступ к файловой системе самого сервера. Следовательно, доступ к файловой системе может получить и злоумышленник. Стоит ему указать имя файла `../../../../etc/passwd`, и он получит доступ к базе данных учетных записей веб-сервера.

Чтобы защитить сервер, мы будем к имени файла добавлять `'.html'`. В случае с HTML-файлами это даже удобно, поскольку нам не нужно указывать расширение файла, а только его имя. А вот если злоумышленник попытается открыть файл

.././../etc/passwd, то наш сценарий будет пытаться открыть файл .././../etc/passwd.html, которого никогда не существовало на нашем сервере.

Итак, чтобы ваша CMS умела открывать HTML-файлы, после последней закрывающей фигурной скобки файла index.php добавьте следующий код:

```
elseif($p=="file") {  
    $f = $f . ".html";  
    $page = join('',file($f));  
}
```

Можно еще, конечно, добавить проверку файла на существование (функция `file_exists()`), но с этим, я думаю, вы справитесь и без моей помощи.

36.3.3. Версия для печати

Наша CMS при выводе страницы выводит ссылку на сценарий `print.php`, который должен вывести версию отображаемой страницы, предназначенную для печати. Версия для печати отличается от обычной версии полным отсутствием дизайна сайта — просто выводится сама страница. В таком виде страницу проще распечатать.

Вносить изменения в файл `index.php` мы не будем, а просто напишем файл `print.php`. Этому сценарию нужно передать переменную `$id`, содержащую номер страницы. Код сценария `print.php` приведен в листинге 36.7.

Листинг 36.7. Сценарий `print.php`

```
<?  
// подключаем файл настроек  
require "config.php";  
  
// подключаемся к серверу БД  
mysql_connect($dbhost,$dbuser,$dbpasswd);  
mysql_select_db($dbname);  
$id = @htmlspecialchars($id);  
$id = @strip_tags($id);  
  
$q = "select * from pages where id=$id limit 1";  
$r = mysql_query($q);  
  
if (mysql_num_rows($r)>0) {  
    $row = mysql_fetch_array($r);  
    $page = $page . "<h1>$row[header]</h1>";  
    $page = $page . "<p><br><br>$row[content]";  
    $page = $page . "&copy; $SERVER_NAME";  
}  
else $page = "<h1>Нет такой страницы!</h1>";  
  
echo $page;  
  
?>
```


36.4. Где взять листинги этой главы?

Понимая, что набирать вручную длинные листинги этой главы никому не захочется, я решил облегчить вам жизнь. Все приведенные здесь листинги доступны в каталоге Glava_36 сопровождающего книгу электронного архива (см. приложение 4). Там вы найдете следующие файлы:

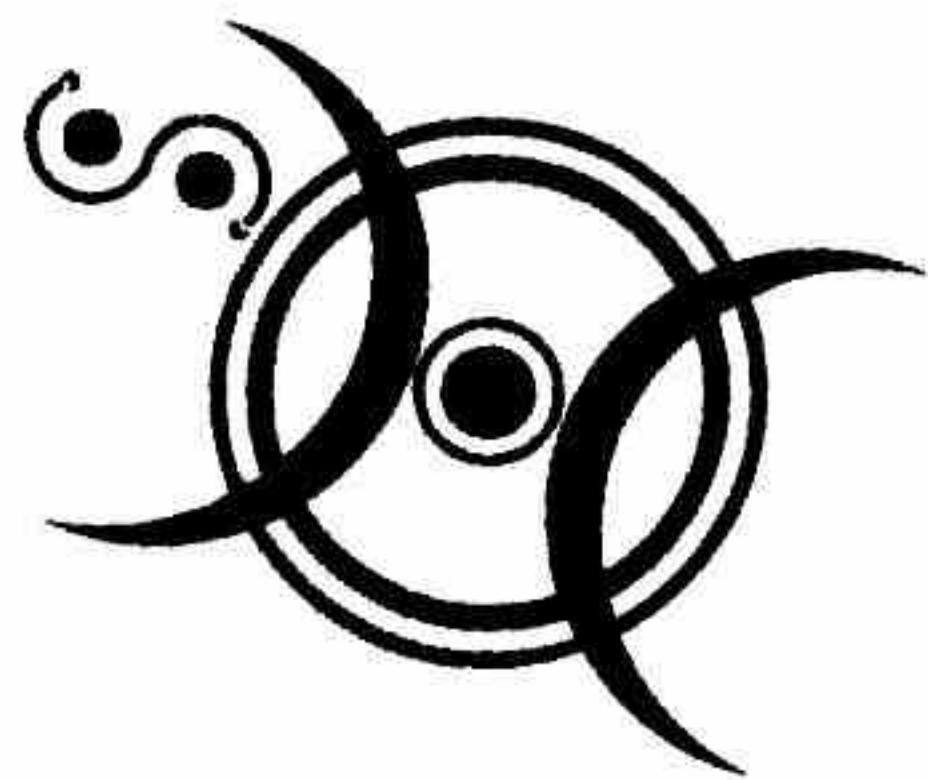
- ☐ `.htaccess` — применяется для настройки сервера (включает директиву `register_globals` и отображение всех ошибок);
- ☐ `36-3.php` — листинг 36.3, это наш первоначальный `index.php`, чтобы вы убедились, что он работает;
- ☐ `index.php` — основной файл нашего движка;
- ☐ `index-static.php` — версия `index.php` с поддержкой таблицы `static`;
- ☐ `config.php` — файл параметров CMS;
- ☐ `menu.php` — сценарий вывода меню;
- ☐ `page.tpl` — TPL-шаблон;
- ☐ `template.php` — шаблонизатор;
- ☐ `print.php` — сценарий, выводящий версию для печати запрашиваемой страницы;
- ☐ `db.sql` — SQL-файл, создающий все необходимые таблицы и заполняющий их первоначальной информацией. Вы можете отредактировать этот файл по своему усмотрению. Файл используется для импорта таблиц CMS с помощью программы `phpMyAdmin`.

Итак, в каталоге Glava_36 находится полностью (но, понятное дело, без дизайна и без панели управления) рабочая версия движка, которая используется в нескольких моих проектах. Дизайн вам не нужен, а вот панель управления придется разработать самостоятельно — это будет вашим домашним заданием.

ПРИМЕЧАНИЕ

Этот движок требует включения директивы `register_globals` для нормальной работы. С одной стороны, использование этой директивы в наши дни не совсем корректно. С другой стороны, у вас будет стимул разобраться в коде движка и переделать его так, чтобы он мог работать с выключенной `register_globals`. В любом случае движок абсолютно рабочий, но чтобы работа эта выполнялась корректно, скопируйте файл `.htaccess` в каталог, в который вы поместили файлы движка.

ГЛАВА 37



Создание фотогалереи

37.1. Постановка задачи

В этой главе мы разработаем совсем простую галерею изображений. Мы не станем организовывать аутентификацию пользователей, поэтому добавить изображение в нашу галерею сможет любой посетитель. Понятно, мы установим ограничение на размер файла и размеры фотографии. При загрузке файла в галерею будет создаваться его миниатюра содержащегося в нем изображения.

Наша галерея будет состоять из двух файлов:

- `gallery_show.php` — показывает галерею и форму для загрузки изображений;
- `gallery_upload.php` — управляет загрузкой файла на сервер и создает миниатюру.

37.2. Загрузка изображений на сервер

Прежде всего напомним сценарий, управляющий загрузкой изображения на сервер, — ведь перед просмотром галереи нужно загрузить в нее хотя бы пару картинок. Создайте файл `gallery_form.html` следующего содержания:

```
<FORM ENCTYPE=multipart/form-data ACTION=gallery_upload.php  
METHOD=POST>
```

```
Выберите файл: <INPUT NAME=myfile TYPE=file>
```

```
<INPUT TYPE=submit VALUE=Загрузить>
```

```
</FORM>
```

ПРИМЕЧАНИЕ

Приведенный здесь код вы найдете в файле `gallery_form.html` каталога *Глава_37* сопровождающего книгу электронного архива (см. приложение 4).

Как вы уже догадались, эта форма будет передавать изображение сценарию `gallery_upload.php`. Теперь осталось создать этот сценарий. В главе 29 была рассмотрена загрузка файлов на сервер. Здесь мы используем приведенный там код. Наша форма предусматривает загрузку только одного файла, но вы можете модернизировать

вать ее, чтобы загружать сразу несколько файлов. Как это сделать, было также показано в *главе 29*.

После загрузки файла нам нужно создать его миниатюру. Создавать миниатюру нам поможет функция `imageresize()`, которую мы разработали в *главе 15*. При необходимости вы сможете получить дополнительную информацию о загруженных изображениях с помощью функции `exif_read_data()`. Пример использования этой функции был показан в той же *главе 15*.

В листинге 37.1 представлен сценарий `gallery_upload.php`.

Листинг 37.1. Сценарий `gallery_upload.php`

```
<?php
// если upload файла
if(isset($_FILES["myfile"]))
{
    $myfile = $_FILES["myfile"]["tmp_name"];
    $myfile_name = $_FILES["myfile"]["name"];
    $myfile_size = $_FILES["myfile"]["size"];
    $myfile_type = $_FILES["myfile"]["type"];
    $error_flag = $_FILES["myfile"]["error"];

    // если ошибок не было
    if($error_flag == 0)
    {

        $f_thum="gallery/thum_" . $myfile_name;

        print("Имя файла на нашем сервере (во время запроса): ".$myfile."<br>");
        print("Имя файла на компьютере пользователя: ".$myfile_name."<br>");
        print("Имя файла на сервере: ".$f."<br>");
        print("Имя thumb-файла на сервере: ".$f_thum."<br>");
        print("MIME-тип файла: ".$myfile_type."<br>");
        print("Размер файла: ".$myfile_size."<br><br>");
        // если размер файла больше 512 Кбайт...
        if ($myfile_size > 512*1024)
            die('Размер файла больше 512 Кб! Уменьшите файл и повторите попытку');

        // копируем файл из временного каталога в галерею
        copy($myfile,"gallery/".$myfile_name);

        // делаем миниатюру
        function imageresize($outfile,$infile,$neww,$newh,$quality) {

            $im=imagecreatefromjpeg($infile);
            $im1=imagecreatetruecolor($neww,$newh);
            imagecopyresampled($im1,$im,0,0,0,0,$neww,$newh,imagesx($im),imagesy($im));
```



```
    imagejpeg($im1,$outfile,$quality);
    imagedestroy($im);
    imagedestroy($im1);
}

// создаем миниатюру размером 160×100, качество JPEG 75
imageresize($f_thum,"gallery/$myfile_name",160,100,75);

}
}
else
{
    // выводим форму
    echo file_get_contents('gallery_form.html');
}

?>
```

ПРИМЕЧАНИЕ

Код этого сценария вы найдете в файле *gallery_upload.php* каталога *Glava_37* сопровождающего книгу электронного архива (см. приложение 4).

Проанализируем код этого сценария. Сначала мы проверяем, был ли передан файл. Если нет, то выводим форму (файл *gallery_form.html*). А вот если файл был передан на сервер, мы выводим его краткую характеристику (рис. 37.1). Вообще, это не обязательно, и вы можете удалить все `print`-операторы. При желании можно, наоборот, вывести дополнительную информацию, получаемую с помощью функции `exif_read_data()`.

Далее, если размер файла превышает 512 Кбайт, мы ничего копировать в галерею не станем, а просто завершим работу сценария:

```
if ($myfile_size > 512*1024)
    die('Размер файла больше 512 КБ! Уменьшите файл и повторите попытку');
```

Если размер файла не превышает лимита, мы копируем файл из временного каталога в каталог *gallery*:

```
copy($myfile,"gallery/$myfile_name");
```

Понятно, что нужно создать этот каталог и установить для него права доступа 777:

```
mkdir gallery
chmod 777 gallery
```

Можно попытаться создать этот каталог в РНР-сценарии, но я не совсем уверен, что у вас хватит прав для выполнения этого оператора (напомню, что интерпретатор РНР выполняется от бесправного пользователя):

```
@mkdir("gallery", 0777);
```

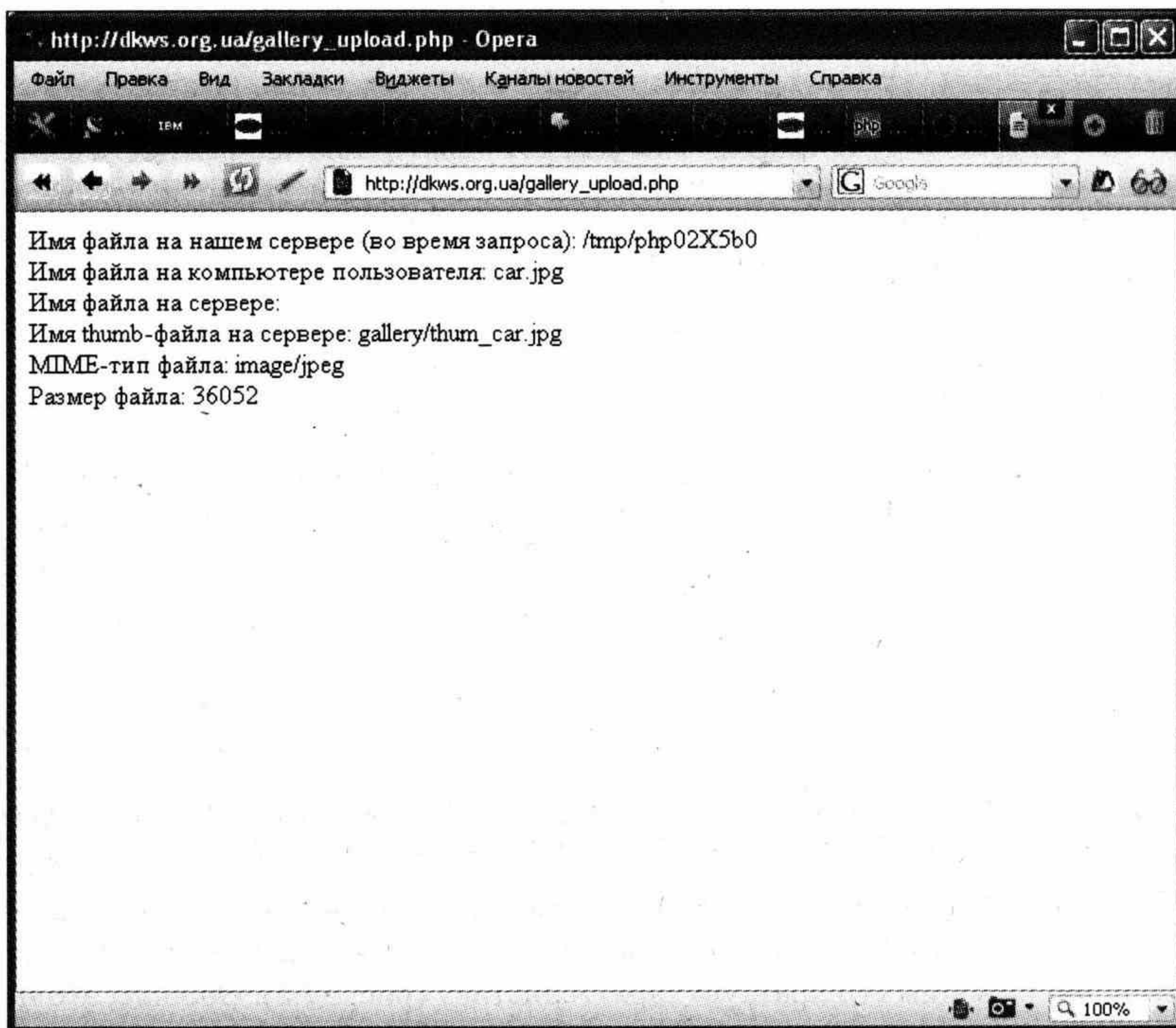



Рис. 37.1. Характеристики загруженного файла

Вернемся к копированию загруженного файла. Мы используем переменную `$myfile_name`, содержащую имя файла на компьютере пользователя. Если пользователь загружает, например, файл `car.jpg`, то и на сервере он будет называться `car.jpg`. У этого метода уйма недостатков, в частности:

- ☐ если другой пользователь загрузит файл с таким же именем (`car.jpg`), то первый файл будет перезаписан, что нежелательно;
- ☐ если имя файла содержит русские символы или пробелы, файл на сервере будет называться так же, а это тоже нежелательно, поскольку сервер обычно работает под управлением UNIX, и, учитывая, что Windows и UNIX используют разные кодировки, могут возникнуть проблемы с именем файла.

Поэтому вам нужно придумать систему создания уникальных имен файлов, которая будет проверять существование файла на сервере, а также содержит ли имя файла русские буквы. Все это сложно.

Я предпочитаю не изобретать колесо заново. При загрузке файла на сервер ему присваивается уникальное имя — например, `/tmp/php02X5b0`. Мы можем использовать это же имя, только нужно удалить из него имя каталога (`/tmp/`) и добавить расширение `jpg`. Все, что вам нужно, — это заменить строку копирования файла:

```
copy($myfile, "gallery/$myfile_name");
```


следующими строками:

```
$fname = "gallery/" . basename($myfile) . ".jpg";  
copy($myfile,$fname);
```

Теперь файлы в нашей галерее не будут перезаписываться.

Далее мы вызываем функцию `imageresize()` для создания миниатюры изображения:

```
imageresize($f_thum,"gallery/$myfile_name",160,100,75);
```

Первый параметр — это имя файла миниатюры, которое формируется так:

```
$f_thum="gallery/thum_" . $myfile_name;
```

то есть к имени файла дописывается префикс `thum_`. Если вы внедрили систему уникальности имен файлов, тогда имя файла миниатюры нужно формировать так:

```
$f_thum="gallery/thum_" . basename($myfile) . ".jpg";
```

Вызов функции `imageresize()` нужно изменить так:

```
imageresize($f_thum,$fname,160,100,75);
```

Вы можете изменить размеры миниатюры и ее качество. Чтобы миниатюры создавались более качественно, вам надо с помощью функции `exif_read_data()` узнать высоту и ширину картинки. Обычно картинки имеют горизонтальную (альбомную) ориентацию, т. е. ширина их больше, чем высота. Но если пользователь будет загружать картинку вертикальной (портретной) ориентации, когда высота больше, чем ширина, то вам следует вызвать функцию `imageresize()` так:

```
imageresize($f_thum,$fname,100,160,75);
```

Как видите, мы поменяли местами значения высоты и ширины.

С загрузкой почти все. В идеале хорошо бы реализовать систему одобрения загружаемых картинок. Для этого создается MySQL-таблица, в которую вносятся все имена файлов из каталога `gallery`. Для каждого файла в таблице имеется отметка (поле `APR`) — одобрен он или нет. Тогда при показе галереи отображаются только одобренные файлы. Сейчас мы этим заниматься не будем, а пример создания системы одобрения будет представлен в следующей главе при создании гостевой книги.

37.3. Вывод галереи

Теперь нам осталось самое малое — вывести галерею: показать все миниатюры из каталога `gallery` и вывести форму добавления нового изображения. Этот сценарий очень прост. Сначала мы с помощью функции `scandir()` получим массив, содержащий список файлов каталога `gallery`, затем пройдемся по этому массиву и выведем как картинки только те файлы, имя которых начинается с `"thum_"`. При выводе миниатюр мы создадим ссылки на оригинальные изображения, которые будут открываться в новом окне (параметр `target=_blank` тега `A`). Полный код сценария `gallery_show.php` приведен в листинге 37.2. Результат работы сценария показан на рис. 37.2.

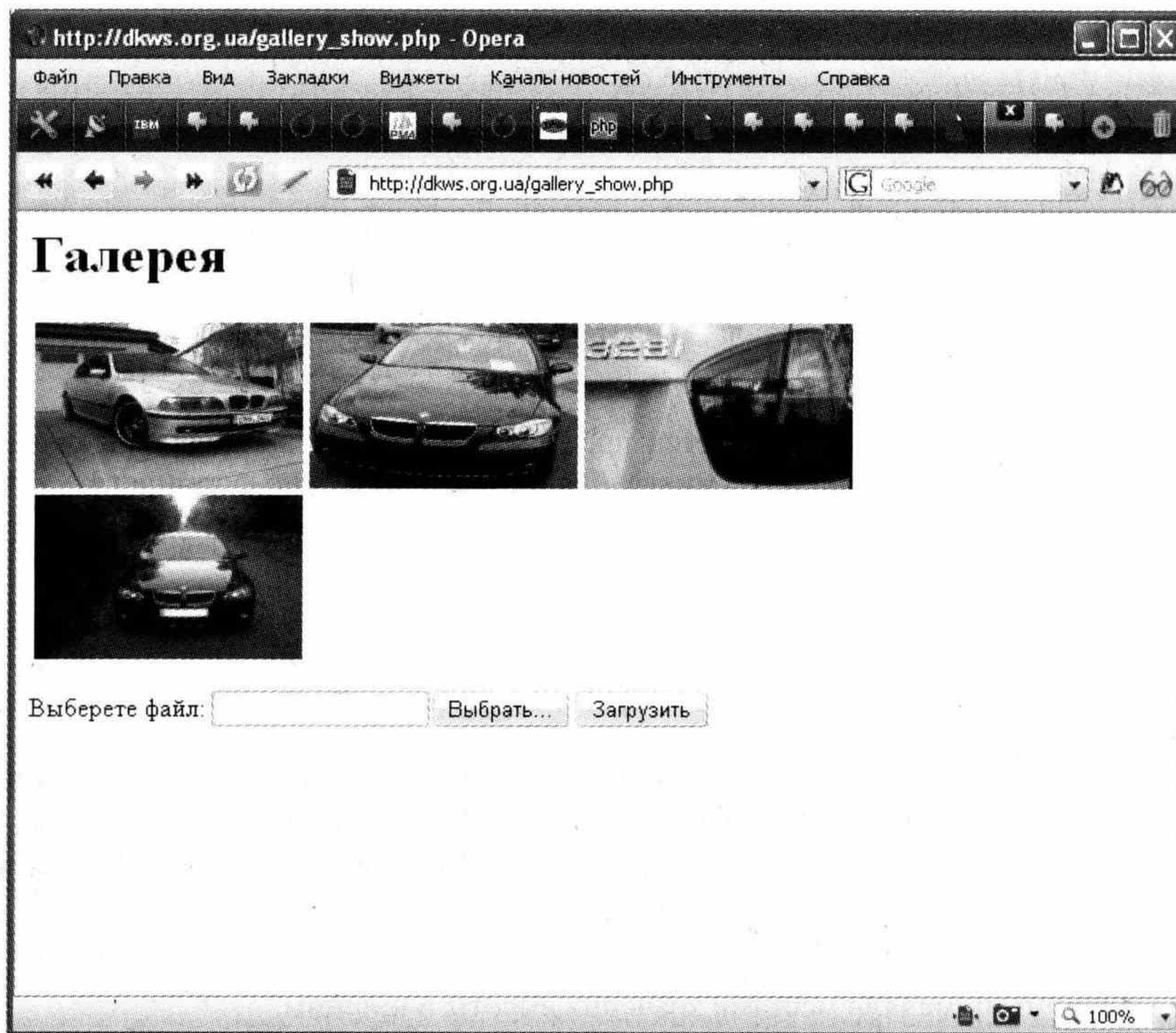


Рис. 37.2. Созданная нами галерея

Листинг 37.2. Вывод галереи — сценарий gallery_show.php

```
<?php

$dir = "gallery";
$N = 3;                // количество картинок в строке

echo "<h1>Галерея</h1><p><table width=100$ cols=$N>";

$i = 0;                // счетчик итераций

// получаем все элементы каталога gallery в массив images
$images = scandir($dir);

// выводим таблицу с картинками
foreach ($images as $k=>$v)
{
    // выводим только миниатюры и ссылки на большие изображения
    if (strpos($v, "thum_") !== false)
```



```
{
    // каждые N итераций выводим новую строку таблицы
    if ($i % $N == 0) echo "<tr>";
    echo "<td>";
    $image = substr($v, 5);
    echo "<a target=_blank href=\""$dir//"$image\"">
        <img border=0 src=\""$dir//"$v\""></a>";
    echo "</td>";
    $i++;
}

}
echo "</table>";

echo file_get_contents('gallery_form.html');

?>
```

ПРИМЕЧАНИЕ

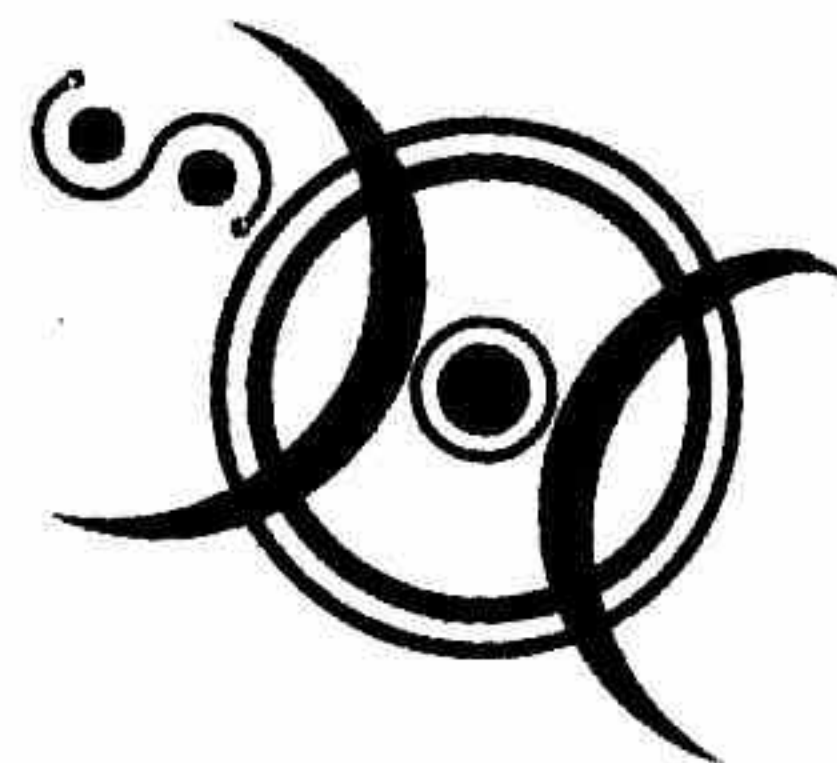
Код этого сценария вы найдете в файле *gallery_show.php* каталога *Glava_37* сопровождающего книгу электронного архива (см. приложение 4).

Конечно, нет предела совершенству. В наш сценарий можно добавить, как минимум, постраничный вывод галереи, но здесь я это делать не стал, чтобы не усложнять код сценария вывода галереи. Принцип постраничного вывода содержимого описан в *главе 21*.

Для подключения галереи к нашему движку (см. *главу 36*) нужно в определенном (по вашему усмотрению) месте кода движка добавить следующий оператор:

```
echo file_get_contents('gallery_show.php');
```


ГЛАВА 38



Гостевая книга

38.1. Пережиток прошлого?

Ранее гостевая книга была чуть ли не единственным средством обратной связи пользователей сайта с его администрацией. Сегодня гостевая книга отошла на второй план. Ее заменили форумы и блоги, предоставляющие гораздо больше возможностей. Однако если вы не планируете устанавливать на свой сайт ни блог, ни форум, тогда можно ограничиться гостевой книгой, тем более что она довольно проста в реализации.

В этой главе мы разработаем не совсем простую гостевую книгу. Простая гостевая книга выводит сообщения пользователей и принимает новые сообщения в базу данных. Мы же немного расширим ее функции.

- ❑ Во-первых, мы станем выводить не все сообщения, а только проверенные администратором. Это вполне логично — ведь как только спамеры узнают о вашей гостевой книге, она сразу же будет завалена спамом. При получении нового сообщения администратору отправляется в e-mail с просьбой подтвердить возможность размещения полученного сообщения. Если сообщение корректно, администратор подтверждает его размещение в базе данных. В противном случае он может его удалить (в e-mail, отправленном администратору, имеется соответствующая ссылка). Как видите, наша гостевая книга даже не нуждается в панели управления.
- ❑ Во-вторых, перед помещением в базу данных из сообщения будет удален весь вредоносный код, который, возможно, поместили в сообщение спамеры.
- ❑ В-третьих, мы установим переменную сессии сроком в одну неделю, что не позволит одному и тому же пользователю отправлять повторные сообщения (это также оградит нас от спамеров).

38.2. Разработка базы данных и структура гостевой книги

База данных будет состоять всего из одной таблицы:

```
CREATE TABLE gb (
  ID INT NOT NULL AUTO_INCREMENT,
  UNAME CHAR( 40 ),
  t TEXT,
  APR INT DEFAULT 0,
  PRIMARY KEY ( ID )
)
```

Первое поле — уникальный номер сообщения (ID), второе — имя пользователя, оставившего сообщение, а третье — само сообщение. Если поле APR равно 1, то сообщение подтверждено администратором, и его можно вывести в гостевой книге. В противном случае сообщение не отображается.

Наша гостевая книга будет состоять из трех файлов:

- gb_show.php — основной сценарий, выводящий гостевую книгу и форму для добавления нового сообщения;
- gb_add.php — добавляет новое сообщение в гостевую книгу;
- gb_service.php — служебный сценарий, удаляющий спам и одобряющий нормальные сообщения.

ПРИМЕЧАНИЕ

Все файлы гостевой книги вы можете найти в каталоге *Глава_38* сопровождающего книгу электронного архива (см. приложение 4).

38.3. Вывод гостевой книги

Сценарий вывода гостевой книги очень похож на сценарий постраничного вывода таблицы, созданный нами в *главе 21*, но у него есть два отличия: он выводит только подтвержденные сообщения, для которых поле APR = 1. Подробно этот сценарий рассматривать мы не станем. Все, что изменилось, — это названия полей и самого сценария, добавлены также некоторые дополнительные операторы (листинг 38.1).

Листинг 38.1. Сценарий gb_show.php

```
<?php
// подключаемся к серверу, выбираем базу данных и устанавливаем кодировку

require_once "config.php";

@mysql_connect($dbhost, $dbuser, $dbpasswd) or
die("Не могу подключиться к БД");
```



```
@mysql_select_db($dbname) or die("Не могу выбрать БД");

mysql_query('SET NAMES cp1251;');

// количество объявлений на странице
$N = 5;

echo "<h1>Добро пожаловать в гостевую книгу!</h1>";

// использовать функцию mysql_num_rows() нельзя,
// для того чтобы узнать общее количество записей, нужно выполнить
// запрос select * from gb, т. е. получить все записи, а это
// нежелательно, поэтому используем функцию count

$r1=mysql_query("select count(*) as rec from gb where apr=1");
$f = mysql_fetch_row($r1);
$rec = $f[0]; // всего записей в таблице

// если страница не указана, выводим первую
if (!isset($_GET['page'])) $page=0;
else $page = $_GET['page'];

// записи, которые нужно вывести
$records = $page * $N;
// запрос
$q="select * from gb where apr=1 limit ".$records.", $N";
echo "Всего записей $rec<br>";

$r=mysql_query($q); // выполняем запрос
$n = mysql_num_rows($r); // к-во записей

// если страница не первая, выводим ссылку "Назад"
if ($page > 0) {
    $p = $page - 1;
    echo "<a href=gb_how.php?page=$p>Назад</a>&nbsp;";
}

$page++; // увеличиваем страницу

// выводим ссылку на следующие 5 записей (на след. страницу),
// если она есть, т. е. количество записей, которые нужно вывести, и
// смещение не превышает общее количество записей
if ($records+$N < $rec)
    echo "<a href=gb_show.php?page=$page>Далее</a>";

// выводим объявления
for ($i=0; $i<$n; $i++)
{
    $f=mysql_fetch_array($r);
```



```
// выводим дату (поле dt)
echo "<p><table width=100%>
    <tr><td bgcolor=navy width=10%>
        <font name=tahoma size=2 color=white>$f[uname]</font></td>";

// выводим текст объявления (поле t)
echo "<td colspan=2 bgcolor=grey>
    <font name=tahoma size=2 color=white>$f[t]</td>
</tr></table>";
}

// форма добавления сообщения
echo '<p>Новое сообщение:
<p><form action=gb_add.php method=post>
Ваше имя: <input type=text name=uname>
<p><textarea cols=50 rows=10 name=t></textarea>
<p><input type=submit value=Добавить>';

?>
```

Работа сценария gb_show.php показана на рис. 38.1.

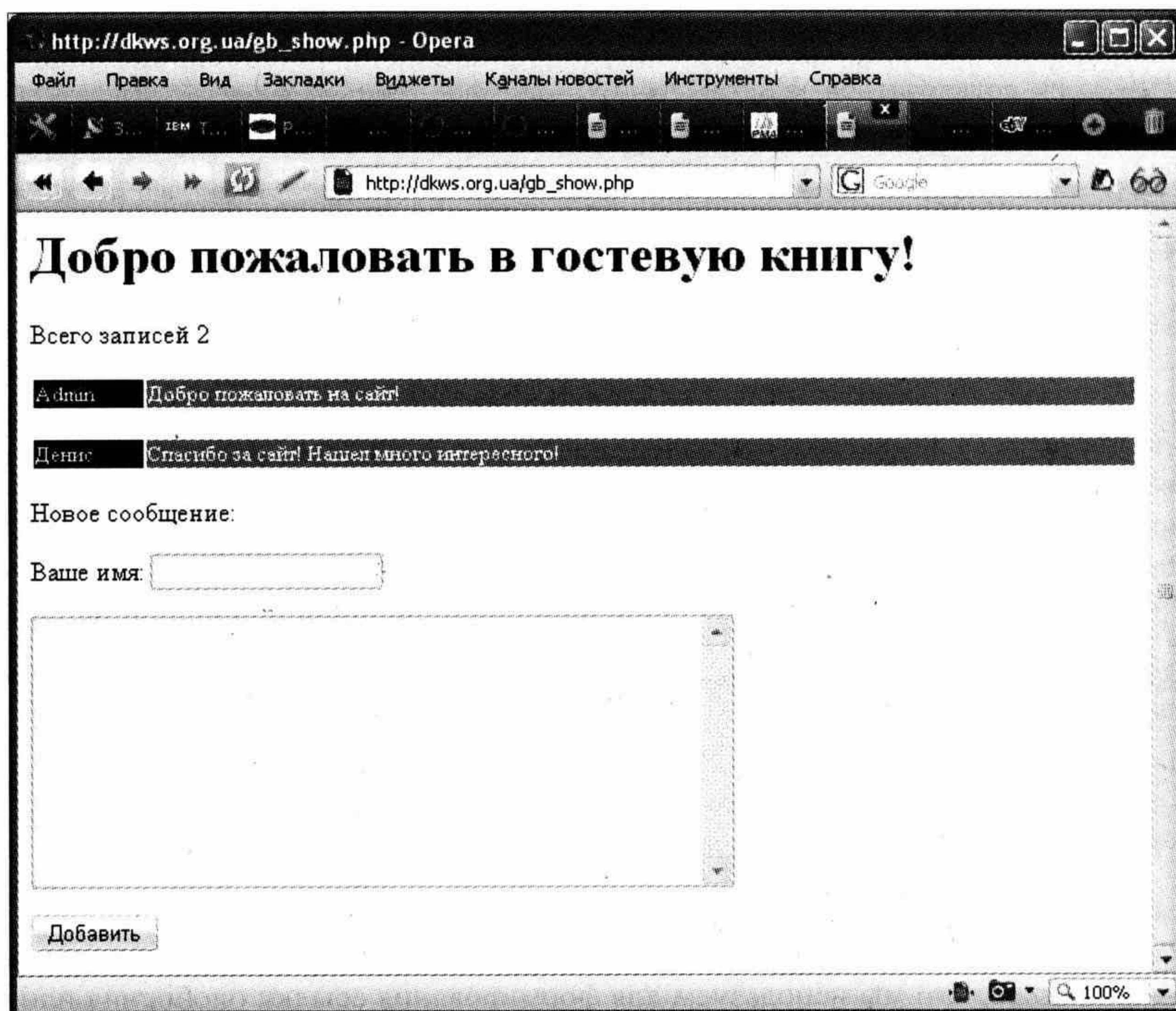


Рис. 38.1. Наша гостевая книга

38.4. Добавление записей в гостевую книгу

Теперь приступим к реализации сценария `gb_add.php`. Прежде всего наш сценарий проверяет переданные ему параметры `$uname` и `$t` и удаляет из них весь вредоносный код. Имя пользователя в нашей гостевой книге указывать не обязательно — если пользователь забудет представиться, будет использовано имя Гость. А вот сообщение ввести обязательно. Если оно длиннее 1 Кбайт, то мы отказываемся добавлять его в базу данных. Записи в гостевой книге обычно небольшие, поэтому 1024 байт вполне хватит:

```
<?php
// подключаемся к серверу, выбираем базу данных и устанавливаем //
кодировку

require_once "config.php";

// задаем e-mail администратора и URL сайта
$admin = "admin@server.ru";
$site = "http://dkws.org.ua/";

@mysql_connect($dbhost, $dbuser, $dbpasswd) or
die("Не могу подключиться к БД");

@mysql_select_db($dbname) or die("Не могу выбрать БД");

mysql_query('SET NAMES cp1251;');

$uname = $_POST['uname'];
$t = $_POST['t'];

// избавляемся от вредоносного кода
$uname = strip_tags($uname);
$uname = htmlspecialchars($uname);

$t = strip_tags($t);
$t = htmlspecialchars($t);
// имя пользователя можно не указывать, а вот текст сообщения —
// обязательно
if (strlen($uname)==0) $uname = "Гость";
if (strlen($t)==0) die('Вы забыли указать текст сообщения');
if (strlen($t)>1024) die('Сообщение слишком большое!');
```

Затем формируется запрос на добавление новой записи в нашу таблицу. После запроса мы получаем идентификатор последней вставленной записи с полем типа `AUTO_INCREMENT`. Его мы используем для формирования ссылки одобрения или удаления сообщения.


```
$q = "insert into gb values(0, \"$uname\", \"$t\", 0)";
mysql_query($q) or die('MySQL insert error');
$ID = mysql_insert_id();
```

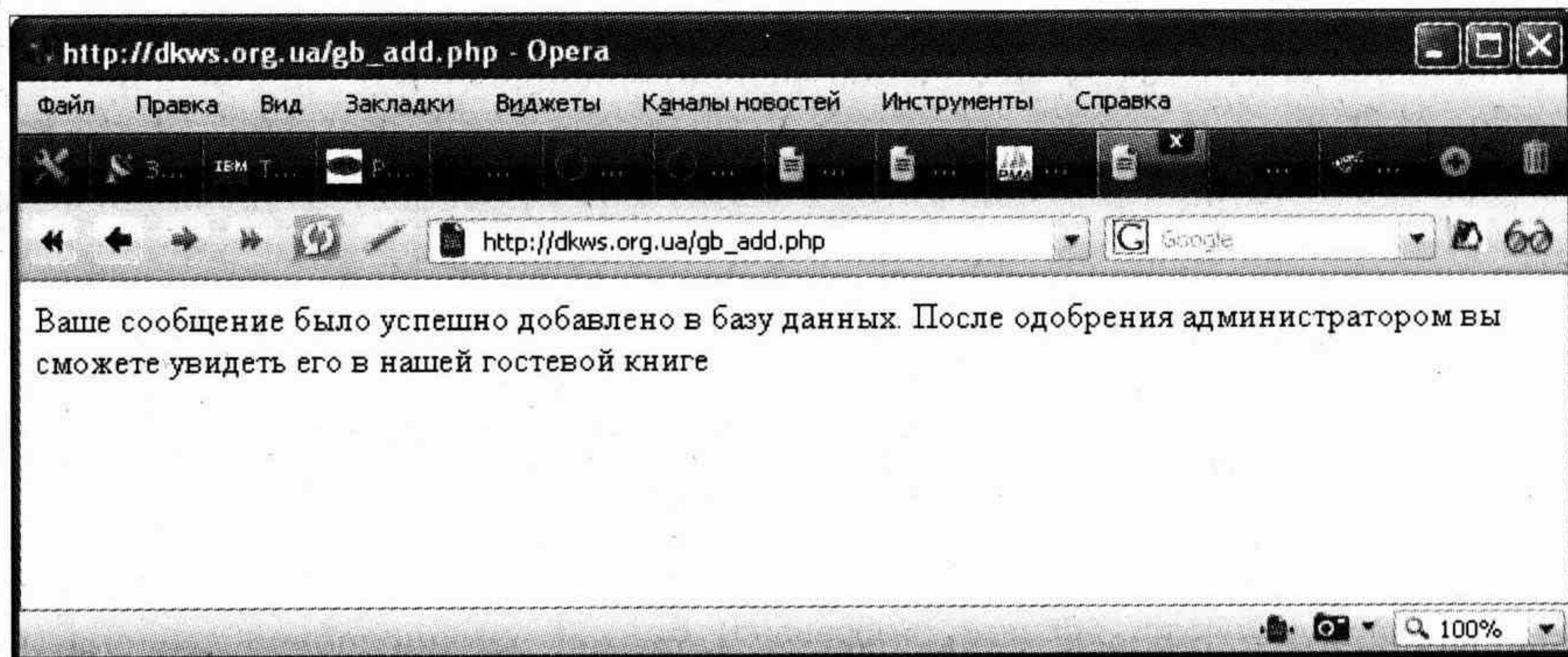


Рис. 38.2. Сообщение, которое видит пользователь после добавления новой записи

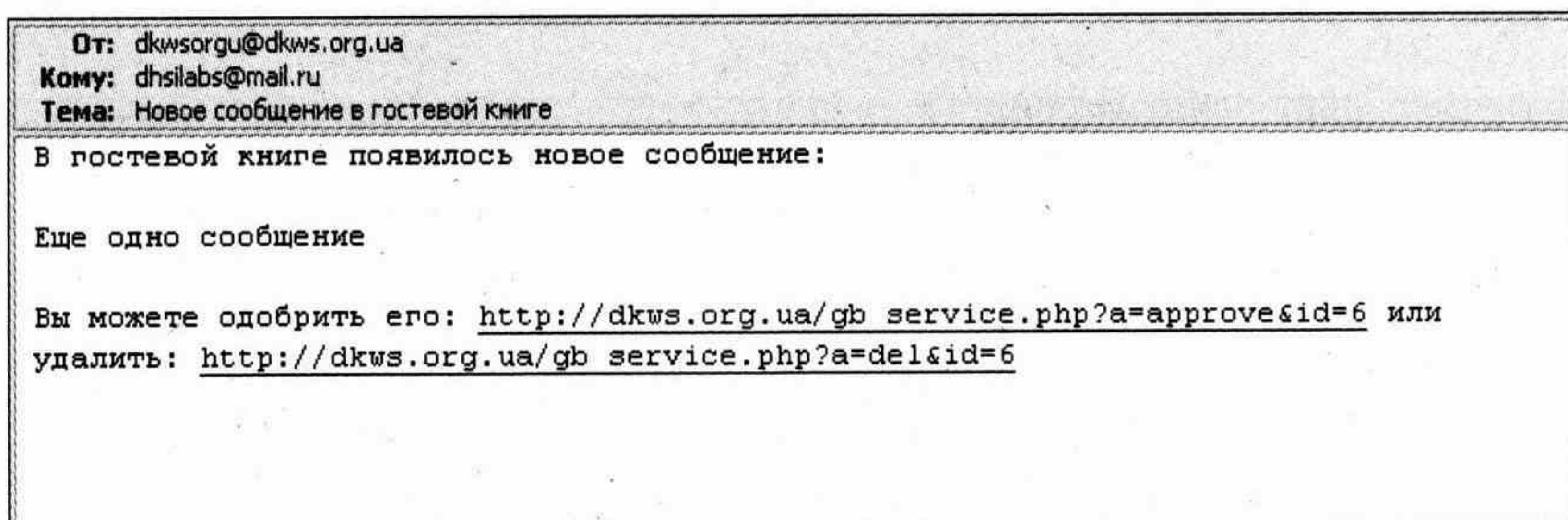


Рис. 38.3. Сервисное сообщение администратору

И в завершение мы выводим сообщение пользователю о том, что его сообщение добавлено (рис. 38.2), а администратору отправляем сообщение пользователя и ссылки, позволяющие добавить или удалить это сообщение (рис. 38.3):

```
// сообщение пользователю
```

```
echo "<p>Ваше сообщение было успешно добавлено в базу данных. После одобрения
администратором вы сможете увидеть его в нашей гостевой книге";
```

```
// сообщение администратору
```

```
$msg = "В гостевой книге появилось новое сообщение:
```

```
$t
```

```
Вы можете одобрить его: " . $site . "gb_service.php?a=approve&id=$ID" . " или
удалить: " . $site . "gb_service.php?a=del&id=$ID";
```



```
// отправляем сообщение администратору
mail($admin, "Новое сообщение в гостевой книге", $msg);

?>
```

Полный код сценария gb_add.php представлен в листинге 38.2.

Листинг 38.2. Сценарий gb_add.php

```
<?php
// подключаемся к серверу, выбираем базу данных и устанавливаем кодировку

require_once "config.php";

$admin = "admin@server.ru";
$site = "http://dkws.org.ua/";
@mysql_connect($dbhost, $dbuser, $dbpasswd) or
die("Не могу подключиться к БД");

@mysql_select_db($dbname) or die("Не могу выбрать БД");

mysql_query('SET NAMES cp1251;');

$username = $_POST['uname'];
$t = $_POST['t'];

// избавляемся от вредоносного кода
$username = strip_tags($username);
$username = htmlspecialchars($username);

$t = strip_tags($t);
$t = htmlspecialchars($t);

// имя пользователя можно не указывать, а вот текст сообщения — // обязательно
if (strlen($username)==0) $username = "Гость";
if (strlen($t)==0) die('Вы забыли указать текст сообщения');

$q = "insert into gb values(0, \"$username\", \"$t\", 0)";

mysql_query($q) or die('MySQL insert error');

$ID = mysql_insert_id();

// сообщение пользователю
echo "<p>Ваше сообщение было успешно добавлено в базу данных. После одобрения
администратором вы сможете увидеть его в нашей гостевой книге";
```



```
// сообщение администратору
$msg = "В гостевой книге появилось новое сообщение:

$t
Вы можете одобрить его: " . $site . "gb_service.php?a=approve&id=$ID" . " или
удалить: " . $site . "gb_service.php?a=del&id=$ID";

// отправляем сообщение администратору
mail($admin, "Новое сообщение в гостевой книге", $msg);

?>
```

38.5. Сервисный сценарий gb_service.php

Сервисный сценарий, хотя и выполняет важнейшие функции по управлению гостевой книгой, самый простой. Он получает параметры `a` и `id`. Первый параметр — это действие над сообщением с номером `id` (`approve` — одобрить, `del` — удалить). Затем, в зависимости от значения параметра `a`, сценарий или одобряет сообщение (устанавливает его поле `APR` в 1), или удаляет его. Полный код сценария приведен в листинге 38.3.

Листинг 38.3. Сценарий gb_service.php

```
<?php
require_once "config.php";

@mysql_connect($dbhost, $dbuser, $dbpasswd) or
die("Не могу подключиться к БД");

@mysql_select_db($dbname) or die("Не могу выбрать БД");

$action = $_GET['a'];
$ID = $_GET['id'];

if (!is_numeric($ID)) die("Номер записи указан неверно!");

if ($action === "approve") {
    $q = "update gb set apr = 1 where id = $ID";
    @mysql_query($q) or die ("MySQL error");
    echo "Сообщение $ID было одобрено";
}

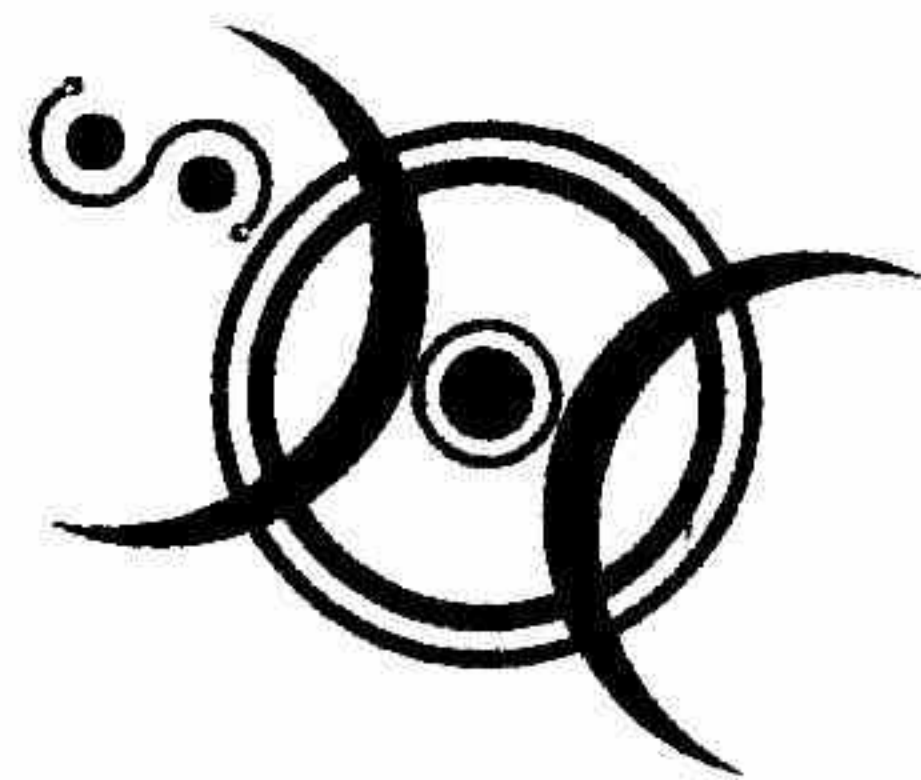
if ($action === "del") {
    $q = "delete from gb where id = $ID";
```



```
@mysql_query($q) or die ("MySQL error");  
echo "Сообщение $ID было удалено";  
  
}  
?>
```

Этот сценарий, однако, содержит две дыры в системе безопасности. Если злоумышленник узнает о его существовании, то он сможет одобрить или удалить любое сообщение, зная его номер (номер можно просто угадать). Чтобы защитить нашу гостевую книгу от злоумышленника, мы можем проверять пароль (который будет храниться как MD5-хэш в файле конфигурации) при каждой операции одобрения/удаления записи. Если же таких операций будет много, и вам не захочется каждый раз вводить пароль администратора, можно для аутентификации пользователя использовать сессии. Если установить длительность сессии, скажем, в 15 минут, этого вполне хватит на администрирование вашей гостевой книги. При этом пароль нужно будет ввести только раз, следующий раз пароль будет запрошен через 15 минут. Учитывая, что сессии привязываются к конкретному компьютеру, и данные сессии хранятся на сервере, вероятность взлома минимальна. Сессии были подробно рассмотрены в *главе 7*. Если вы усвоили приведенный в ней материал, то без проблем реализуете защиту гостевой книги от злоумышленника.

ГЛАВА 39



Импорт новостей с чужого сайта в базу данных WordPress

39.1. Постановка задачи

С помощью PHP можно решать самые разнообразные задачи. Представим некоторый новостной сайт, с которого вам нужно получить новости и внести их в базу данных своего сайта. Конечно, большинство новостных сайтов предоставляют RSS-ленты (см. главу 32), которые как раз и можно использовать для этих целей. Однако на сей раз мы усложним задачу и будем считать, что этот новостной сайт вообще не предоставляет никаких механизмов экспорта новостей. Что делать? Первое, что приходит в голову — получить список новостей и отобразить его на своем сайте. Когда пользователь попытается открыть одну из новостей в списке, сценарий загрузит всю страницу с новостью, избавится от лишнего текста и HTML-кода, вырезав лишь саму новость, и отобразит посетителю результат.

Такой алгоритм хотя и прост, но весьма нерационален. В упрощенном варианте представим, что на наш сайт зашли и прочитали одну новость из списка 1000 пользователей. Следовательно, к новостному сайту было сделано 1000 запросов — как минимум. Это при условии, что все посетители, кроме одной новости, больше ничего не открывали. Как можно видеть, предложенный алгоритм крайне неэффективно расходует системные ресурсы как нашего сервера, так и сервера, обслуживающего новостной сайт.

Что делать? Можно попробовать создать сценарий, который будет запускаться несколько раз в сутки и получать последние новости с новостного сайта. Если полученных новостей еще нет в нашей базе данных, они будут туда внесены. Когда же пользователь зайдет на сайт, ему будут отображены новости из локальной базы данных. Следовательно, никакой дополнительной нагрузки на новостной сервер создаваться не будет. Вместо нескольких тысяч запросов в сутки новостной сервер получит, скажем, 24 лишних запроса в сутки (если вы будете загружать новости каждый час). Согласитесь, так гораздо гуманнее.

Новости могут быть проиллюстрированы изображениями. Мы их также загрузим на свой сервер, чтобы при просмотре новости не были инициированы дополнительные GET-запросы к новостному серверу. Загрузка файлов изображений будет осуществляться с помощью программы cURL (см. разд. 39.3).

Чтобы стало интереснее, загруженные новости мы будем помещать не в какую-то произвольную таблицу с новостями, а в стандартную таблицу WordPress. WordPress — очень популярная платформа, и многие разработчики (особенно, когда нужно создать сайт на скорую руку) предпочитают использовать ее вместо разработки "движка" сайта с нуля.

ПРИМЕЧАНИЕ

Познакомиться с платформой WordPress, зарегистрироваться и начать с ней работать можно на ее сайте <http://ru.wordpress.com>.

Создаваемый нами парсер новостей будет рассмотрен далее. Но прежде, чем вы приступите к его реализации и использованию на практике, рекомендую получить разрешение на перепубликацию новостей с новостного сайта с его помощью. Возможно, вам вообще это запретят, и тогда ваш сайт рискует попасть в Единый реестр запрещенных сайтов (<http://eais.rkn.gov.ru/>). Попасть в него очень просто, а вот чтобы "выбраться" оттуда, придется приложить большие усилия. Если же вам разрешат перепубликацию новостей, то, скорее всего, в обмен предложат определенные условия — например, размещение на вашем сайте обратной ссылки на исходный сайт или его баннера.

39.2. Вспомогательные функции

Для реализации разрабатываемого парсера нам понадобятся две вспомогательные функции. Первая функция — `title_not_exist()`. Она возвращает `true`, если заголовок новости отсутствует в нашей базе данных. Функция очень полезная, поскольку позволяет избежать повторной публикации в базе этой новости. Код функции приведен в листинге 39.1.

Листинг 39.1. Функция `title_not_exist()`

```
// проверяет нет ли заголовка в БД
function title_not_exist($title) {
    $q = "select post_title from wp_posts where post_title=\"$title\"";
    // Раскомментируйте следующую строку для отладки
    // echo "<br>Query $q";
    $r = mysql_query($q);
    // строк больше 0, значит, заголовок есть в базе, возвращаем false
    if (mysql_num_rows($r)>0)
        return false;
    else
        return true;    // заголовок нет в базе, возвращаем true
}
```

Вторая функция — `ru2Lat()`. Она осуществляет преобразование транслитерацией, а именно — преобразует кириллицу в латиницу (листинг 39.2). Функция корректно работает как с русским, так и с украинским языками.

Листинг 39.2. Функция ru2Lat()

```

function ru2Lat($string) {
    $replace=array(
        "' "=>"",
        "` "=>"",
        "a"=>"a", "A"=>"a",
        "б"=>"b", "Б"=>"b",
        "в"=>"v", "В"=>"v",
        "г"=>"g", "Г"=>"g",
        "д"=>"d", "Д"=>"d",
        "е"=>"e", "Е"=>"e",
        "ж"=>"zh", "Ж"=>"zh",
        "з"=>"z", "З"=>"z",
        "и"=>"i", "И"=>"i",
        "й"=>"y", "Й"=>"y",
        "к"=>"k", "К"=>"k",
        "л"=>"l", "Л"=>"l",
        "м"=>"m", "М"=>"m",
        "н"=>"n", "Н"=>"n",
        "о"=>"o", "О"=>"o",
        "п"=>"p", "П"=>"p",
        "р"=>"r", "Р"=>"r",
        "с"=>"s", "С"=>"s",
        "т"=>"t", "Т"=>"t",
        "у"=>"u", "У"=>"u",
        "ф"=>"f", "Ф"=>"f",
        "х"=>"h", "Х"=>"h",
        "ц"=>"c", "Ц"=>"c",
        "ч"=>"ch", "Ч"=>"ch",
        "ш"=>"sh", "Ш"=>"sh",
        "щ"=>"sch", "Щ"=>"sch",
        "ъ"=>"", "Ъ"=>"",
        "ы"=>"y", "Ы"=>"y",
        "ь"=>"", "Ь"=>"",
        "э"=>"e", "Э"=>"e",
        "ю"=>"yu", "Ю"=>"yu",
        "я"=>"ya", "Я"=>"ya",
        "і"=>"i", "І"=>"i",
        "ї"=>"yi", "Ї"=>"yi",
        "є"=>"e", "Є"=>"e"
    );
    return $str=iconv("UTF-8", "UTF-8//IGNORE", strtr($string, $replace));
}

```

Обе эти функции мы объединим в файл `tools.php`, подключаемый к нашему сценарию так:

```
include "tools.php";
```


ПРИМЕЧАНИЕ

Файл *tools.php* вы найдете в каталоге *Glava_39* сопровождающего книгу электронного архива (см. приложение 4).

39.3. Вкратце о cURL

Библиотека *libcurl* (cURL) позволяет серверу передавать файлы на удаленный компьютер по множеству интернет-протоколов. Хотя библиотека эта очень мощная, мы воспользуемся ею лишь для получения файла по HTTP. Адрес получаемого файла задается переменной *\$url*. Полученный файл сохраняется с именем, как у оригинала, в папке *news_img/news_site01*.

Далее приведен соответствующий код, который получает файл изображения, заданный в переменной *\$url*. При этом библиотека cURL представляется удаленному серверу как обычный браузер Firefox, работающий под управлением ОС Windows NT 5.1 (что то же самое, что и Windows XP):

```
$filename = basename($url);  
$ch = curl_init();  
$fp = fopen("news_img/news_site01/" . $filename, "w");  
curl_setopt($ch, CURLOPT_URL, $url);  
curl_setopt($ch, CURLOPT_FILE, $fp);  
curl_setopt($ch, CURLOPT_HEADER, 0);  
curl_setopt($ch, CURLOPT_AUTOREFERER, 1);  
curl_setopt($ch, CURLOPT_USERAGENT, 'Mozilla/5.0 (Windows; U; Windows NT 5.1;  
en-US; rv:1.9.0.3) Gecko/2008092417 Firefox/3.0.3');  
curl_exec($ch);  
curl_close($ch);  
fclose($fp);
```

Дополнительная информация о библиотеке cURL доступна по адресу: <http://se2.php.net/manual/ru/ref.curl.php>.

39.4. Основной сценарий

Поскольку мы используем базу данных WordPress, то в начале сценария следует объявить константу *WP_USE_THEMES* и подключить сценарий *wp-blog-header.php*. В нашем случае данные определения никакого значения не имеют, поскольку вся работа с БД WordPress осуществляется напрямую — средствами самого PHP, а не API WordPress. Однако, если вы надумаете использовать некоторые служебные функции WordPress, сделанные нами определения пригодятся:

```
define('WP_USE_THEMES', false);  
include('../wp-blog-header.php');
```

ПРИМЕЧАНИЕ

Сценарий *wp-blog-header.php* входит в стандартный набор файлов WordPress и находится в корневом каталоге WordPress.

Затем мы подключаемся к базе данных WordPress и устанавливаем рабочую кодировку UTF-8 (WP работает только с ней):

```
mysql_connect("localhost", "admin", "123456");
mysql_select_db("news");
mysql_query("SET NAMES utf8");
```

Далее мы подключаемся к новостному сайту, получаем список последних новостей и просто выводим его в целях отладки, чтобы вы видели, что парсер работает:

```
$f = file_get_contents('http://www.some.news.site');
$begin=strpos(strtolower($f), '<div class="view-content">', 0);
// Ищем начало списка...
echo $begin . " ";
// ...и его конец
$end=strpos(strtolower($f), 'Страницы', $begin);
echo $end;

// Выводим список новостей
$txt=substr($f, $begin, $end-$begin);
// когда сценарий будет работать как нужно, прокомментируйте
// следующую строку
echo $txt;
```

Как видите, начало и конец текста списка задаются маркерами `$begin` и `$end`. Началом списка новостей служит строка HTML-кода `'<div class="view-content">'`, а ее концом — строка `'Страницы'`. В вашем случае все будет несколько иначе, поэтому маркеры начала и конца новости придется изменить. Чтобы узнать, какие именно маркеры будут у вас, нужно исследовать HTML-код списка, который вы получили с новостного сайта.

Обычно в списке новостей имеются следующие элементы: заголовок, ссылка на страницу с новостью, картинка и краткое описание новости. Иногда список может быть сокращенным — только заголовок новости и ссылка на страницу с новостью. Нас только это и интересует: ссылка и заголовок. Именно поэтому мы удаляем все теги, кроме `<a>`:

```
$txt = strip_tags($txt, "<a>");
```

В результате у нас останутся одни ссылки. После этого с помощью следующего ужасного регулярного выражения мы получаем два списка: заголовки (`$titles`) и сами ссылки (`$urls`):

```
preg_match_all('~<a.*?href="([^\"]+)".*?>(.*?)</a>~s', $txt, $matches);
// для отладки:
// print_r($matches);

$urls = $matches[1];
$titles = $matches[2];
```


Теперь нам нужно обработать заголовок за заголовком:

```

krsort($titles);
foreach ($titles as $k=>$v)
{
    // В переменной $v – оригинальный заголовок, как правило, содержит
    // русские буквы и пробелы
    // если заголовок пустой, переходим на следующую итерацию
    if ($v==='') continue;

```

Для создания URL для WordPress мы должны преобразовать заголовок в латиницу и заменить все пробелы символами подчеркивания — как это делает WordPress:

```

$title = ru2Lat($v);
$title = str_replace(" ", "_", $title);
$title = substr($title, 0, 255);

```

Далее мы проверяем, есть ли заголовок \$v в нашей базе данных, и если нет, то продолжаем обработку заголовка:

```

if (title_not_exist($v)) {
    echo "<br>Заголовок нет в БД, добавляем<br>URL новости - $urls[$k]";

```

Затем надо получить страницу с новостью и вырезать из нее текст, содержащий новость (остальной код нам не нужен). Вырезать текст новости мы, как обычно, будем с помощью маркеров \$begin и \$end. Решение может и не самое лучшее, зато интуитивно понятное:

```

$f = JOIN('', file('http://www.edimsovkusom.com' . $urls[$k]));
// Ищем начало новости...
$begin = strpos(strtolower($f), '<div class="start">', 0);

// ...и её конец
$end = strpos(strtolower($f), '<div class="end">', $begin);

// получаем текст новости
$txt = substr($f, $begin, $end - $begin);

```

Как уже отмечалось, маркеры следует изменить в соответствии с HTML-кодом страницы с новостью.

Теперь нужно получить адрес (URL) иллюстрации (если она есть) из новостной статьи. Мы здесь получим адрес только первого изображения — для простоты изложения, да и поставленная передо мной задача, когда я разрабатывал этот парсер, предусматривала загрузку именно первой (часто единственной) картинки.

Для этого можно использовать или DOM-парсер вроде SimpleHTML или изощряться с манипулированием HTML-кодом. Первое решение более элегантно, да и зачем изобретать колесо заново? Следующая функция получает адрес первого изображения в HTML-коде с помощью парсера SimpleHTML:

```

function get_first_image($html) {
    require_once('SimpleHTML.class.php')

```



```
$post_html = str_get_html($html);  
$first_img = $post_html->find('img', 0);  
if($first_img !== null) {  
    return $first_img->src;  
}  
return null;  
}
```

ПРИМЕЧАНИЕ

Парсер SimpleHTML (файл *SimpleHTML.class.php*) вы найдете в каталоге *Glava_39* сопровождающего книгу электронного архива (см. приложение 4). Его также можно скачать по ссылке <http://simplehtmldom.sourceforge.net/>.

В качестве параметра мы передаем функции весь HTML-код — в нашем случае он хранится в переменной *\$txt*. Как видите, с помощью метода *find()* мы получаем первый элемент **. Если нужно получить последний элемент, укажите значение *-1*:

```
$last_img = $post_html->find('img', -1);
```

Если требуется получить все элементы, тогда вообще не указывайте второй параметр. Вы получите массив, содержащий информацию обо всех тегах **.

Ну, а если вы не хотите использовать сторонние парсеры, покажу для сравнения как можно получить первое изображение иным способом:

```
$img_pos = strpos($txt, 'img src="');  
$quote_pos = strpos($txt, '" wi', $img_pos);  
$len = $quote_pos - $img_pos;  
$url = substr($txt, $img_pos+9, $len-9);
```

Как можно видеть, этот код весьма нерационален. Константа 9 будет изменяться в зависимости от реального HTML-кода. Так что показанный код не следует применять на практике. Он приведен лишь для демонстрации того, что в большинстве случаев лучше использовать уже готовые решения типа SimpleHTML.

Узнав адрес (URL) изображения, следует получить его и поместить в каталог *news_img/news_site01*. Осуществляющий это код уже был показан ранее — для загрузки файла изображения мы использовали библиотеку *cURL* (см. *разд. 39.3*), и я не вижу смысла приводить его еще раз.

Когда картинка будет загружена в наш каталог, в тексте новости (переменная *\$txt*) нужно заменить ее адрес (переменная *\$url*) на новый локальный адрес — адрес картинки на нашем сервере:

```
$local_url = "http://our.site/news_img/news_site01/" . $filename;  
// производим замену URL в тексте  
$txt = str_replace($url, $local_url, $txt);
```

Остальное дело техники — поместить текст новости в нашу базу данных. Однако сначала нужно выполнить некоторые подготовительные действия — например, вычислить текущую дату и получить идентификатор последней записи в таблице

`wp_posts`. Кстати, для этого можно использовать как средства PHP, так и средства WordPress API, а именно объект `$wpdb` (WordPress database):

```
$dt = date('Y-m-d h:i:s');
$last = $wpdb->get_row("SHOW TABLE STATUS LIKE 'wp_posts'");
$lastid = $last->Auto_increment;
```

Перед добавлением новости в таблицу удалим лишние HTML-теги, оставив только необходимые:

```
$txt = strip_tags($txt, "<a><p><b><strong><img>");
```

Сформируем запросы к таблицам `wp_posts` и `wp_term_relationships` (в первом запросе измените фрагмент `our.site` на адрес вашего сайта):

```
$q = "insert into wp_posts (
    ID, post_author, post_date, post_date_gmt,
    post_content, post_title, post_excerpt, post_status,
    comment_status, ping_status, post_password,
    post_name, to_ping, pinged, post_modified,
    post_modified_gmt, post_content_filtered, post_parent,
    guid, menu_order, post_type,
    post_mime_type, comment_count) VALUES (
    0, 3, '$dt', '$dt', '$txt', '$v', '', 'publish',
    'open', 'open', '', '$title', '', '', '$dt', '$dt',
    '', 0, 'http://our.site/?p=$lastid', 0, 'post',
    '', 0)";
```

```
mysql_query($q);
```

```
$q2 = "INSERT INTO wp_term_relationships
    (object_id,term_taxonomy_id) VALUES ('$lastid','1')";
```

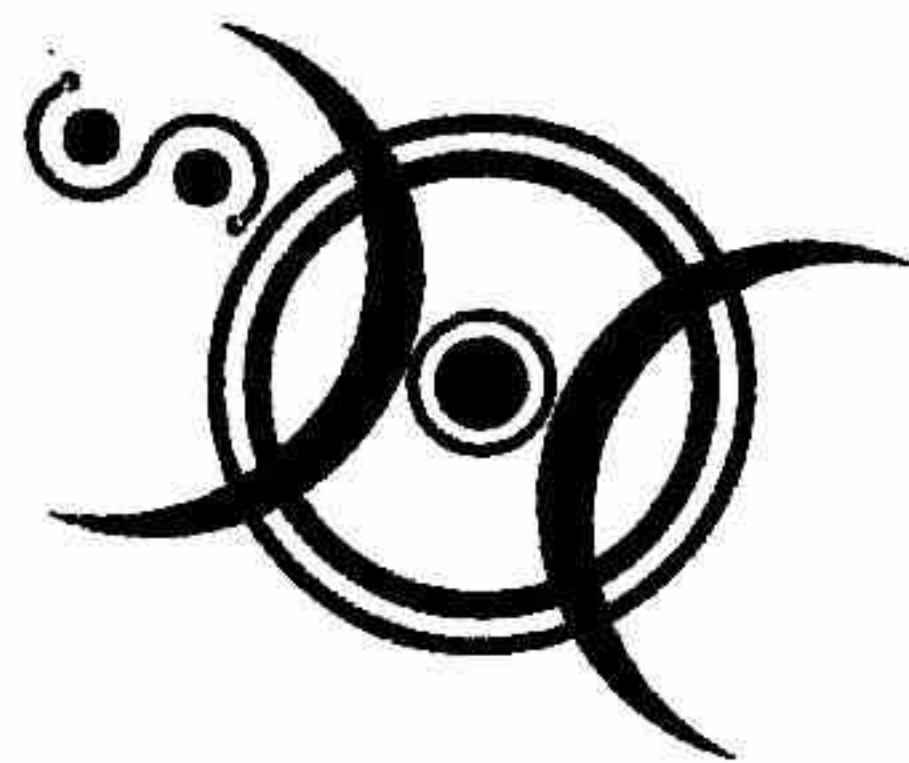
```
mysql_query($q2) or die(mysql_error());
```

ПРИМЕЧАНИЕ

Полный исходный код этого сценария приведен в файле *parser.php* каталога *Glava_39* сопровождающего книгу электронного архива (см. приложение 4). Из соображений экономии места в книге комплектный код парсера не приводится (хотя он и так фрагментарно приведен практически на 98 % за исключением некоторых деталей).

Прежде чем попробовать исходный код на практике, установите WordPress, а также внесите необходимые изменения в код сценария (параметры доступа к MySQL, адрес новостного сервера, маркеры начала и конца списка новостей и новости и т. д.).

ГЛАВА 40



Создание счетчика сайта

40.1. Постановка задачи

Здесь мы разработаем простейший классический счетчик посещений, выводящий два числа: количество посещений за сутки и общее количество посещений. Этот счетчик будет только подсчитывать посещения, но не станет вести какую-либо статистику. Если вам нужна система статистики сайта, то в следующей главе мы рассмотрим несколько уже готовых таких систем, а вы выберете ту, которая вам больше подходит.

Работает наш счетчик очень просто. Значения счетчика хранятся в небольшой таблице `counter`, у которой всего два поля: `daily` (количество посещений за день) и `total` (общее количество посещений). Счетчик считывает значения `daily` и `total`. Если посетитель уникален (т. е. на протяжении последних суток он еще не заходил на сайт), то мы увеличиваем оба значения и выводим их в браузер в виде картинки.

Откуда счетчик знает, что посетитель уникален? У нас имеется еще таблица IP-адресов, в которую будут вноситься IP-адреса всех пользователей, посетивших наш сайт сегодня. Если IP-адрес есть в этой таблице, следовательно, посетитель не уникален.

Раз в сутки (в 00:00) нужно будет запускать сценарий `reset_counter.php`, обнуляющий значение `daily` таблицы `counter` и удаляющий все записи из таблицы IP-адресов. Это можно сделать с помощью менеджера расписаний `crond`, который доступен на любом хостинге.

Все, что нам осталось, — это реализовать счетчик.

40.2. Файл конфигурации

Наш счетчик будет использовать файл конфигурации `config.php`, разработанный в главе 36. Чтобы вы лишний раз не листали книгу, я его вам напомним (листинг 40.1).

Листинг 40.1. Файл config.php

```
<?

$dbhost = 'localhost';           // имя сервера БД
$dbname = 'db';                  // имя БД
$dbuser = 'user';                // имя пользователя БД
$dbpasswd = 'password';          // пароль пользователя

// информация о сайте
$title = 'CMS v0.01';
$description = 'Описание сайта';
$info = '<p>&copy; Денис Колисниченко';

?>
```

40.3. Разработка таблиц *counter* и *ipaddr*

Теперь создадим таблицы *counter* и *ipaddr*. Их можно создать с помощью SQL-операторов, представленных в листинге 40.2. Эти операторы можно или передать функции `mysql_query()`, или ввести в поле SQL-запроса программы `phpMyAdmin`.

Листинг 40.2. SQL-операторы для создания таблиц счетчика

```
CREATE TABLE counter
(
daily int,
total int
);
CREATE TABLE ipaddr
(
ip char(20)
);
```

Наши таблицы весьма примитивны, и, по большому счету, для нашей цели хватило бы двух файлов, но использование MySQL значительно упрощает сценарий, поскольку нам не приходится беспокоиться о правах доступа к файлу.

Если вам не с руки устанавливать `phpMyAdmin` из-за пары запросов, вы можете использовать следующий сценарий (листинг 40.3).

Листинг 40.3. Сценарий создания таблиц *create_counter_tables.php*

```
<?

include "config.php";
mysql_connect($dbhost, $dbuser, $dbpasswd);
```



```
mysql_select_db($dbname);

// создаем таблицу счетчика
$q = "CREATE TABLE counter
(
daily int,
total int
);";

mysql_query($q);

// создаем таблицу IP-адресов
$q= "CREATE TABLE ipaddr
(
ip char(20)
);";

mysql_query($q);

?>
```

После создания таблиц нужно заполнить их начальными данными. В первую таблицу мы добавим два нуля, а в таблицу IP-адресов — адрес 127.0.0.1.

```
INSERT INTO counter VALUES(0, 0);
INSERT INTO ipaddr VALUES('127.0.0.1');
```

40.4. Сценарий counter.php

Теперь приступим к разработке главного сценария — counter.php. Но сначала следует продумать его логику. Первым делом сценарий должен получить текущие значения счетчика и список IP-адресов. Затем — проанализировать, уникален ли IP-адрес. Если его нет в таблице IP-адресов, то адрес уникален, его нужно добавить в эту таблицу и увеличить значения счетчиков. Новые значения счетчиков надо поместить в таблицу counter. Затем нам останется лишь вывести эти значения в виде картинки.

Напишем первую часть нашего сценария:

```
<?

// подключаем конфигурационный файл
require "config.php";

// подключаемся к MySQL-серверу
mysql_connect($dbhost, $dbuser, $dbpasswd) or die("MySQL error");
mysql_select_db($dbname);
```



```
// получаем текущие значения счетчиков
```

```
$q = 'select * from counter';
```

```
$r = mysql_query($q);
```

```
$row = mysql_fetch_array($r);
```

```
$daily = $row['daily'];
```

```
$total = $row['total'];
```

Мы подключились к MySQL-серверу и получили текущие значения счетчиков. Теперь нужно проверить, уникален ли посетитель. Для этого мы определяем IP-адрес пользователя, а затем запрашиваем все IP-адреса из таблицы, совпадающие с нашим. Если результат содержит 0 строк (IP-адреса в таблице нет), то считаем, что посетитель уникальный. Все сказанное реализует приведенный далее фрагмент кода:

```
// получаем IP-адрес пользователя
```

```
$IP = $_SERVER['REMOTE_ADDR'];
```

```
// считаем пользователя уникальным
```

```
$unique = true;
```

```
// есть ли в таблице IP-адресов наш адрес?
```

```
$q = "select * from ipaddr where ip = \"$IP\"";
```

```
$r = mysql_query($q);
```

```
// если IP есть в таблице, пользователь не уникален
```

```
if (mysql_num_rows($r)>0) $unique = false;
```

```
// если уникален, то увеличиваем счетчики и добавляем
```

```
// его IP-адрес в таблицу
```

```
if ($unique)
```

```
{
```

```
$daily++; $total++;
```

```
// обновляем таблицу счетчиков
```

```
$q = "update counter set daily=$daily, total=$total";
```

```
mysql_query($q);
```

```
// добавляем новый IP в таблицу адресов
```

```
$q = "insert into ipaddr values(\"$IP\")";
```

```
mysql_query($q);
```

```
}
```


Осталось только вывести значения `$daily` и `$total` в виде картинки:

```
// выводим как картинку
$img = imagecreatefrompng('dkws.png');
// цвет шрифта — белый
$color = imagecolorallocate($img, 255, 255, 255);

// выводим строку
ImageString($img, 10, 60, 3, "$daily / $total", $color);

// выводим картинку
Header("Content-type: image/png");
ImagePng($img);
```

Мы не станем создавать картинку с нуля, а используем заранее подготовленный шаблон из файла `dkws.png`. После загрузки шаблона мы задаем белый цвет, которым будем выводить значения счетчиков на шаблоне. Затем мы выводим свою строку. Работая со своим шаблоном, обратите внимание на координаты `$x` и `$y` функции `ImageString`:

```
bool imagestring ( resource $image, int $font, int $x, int $y, string $string,
int $color )
```

Последние два оператора — вывод заголовка картинки и вывод самой картинки. Результат представлен на рис. 40.1. При первом запуске сценарий выведет 1/1. Счетчик будет увеличен, только если зайдет пользователь с другим IP-адресом или если вы удалите свой IP-адрес из таблицы IP-адресов.

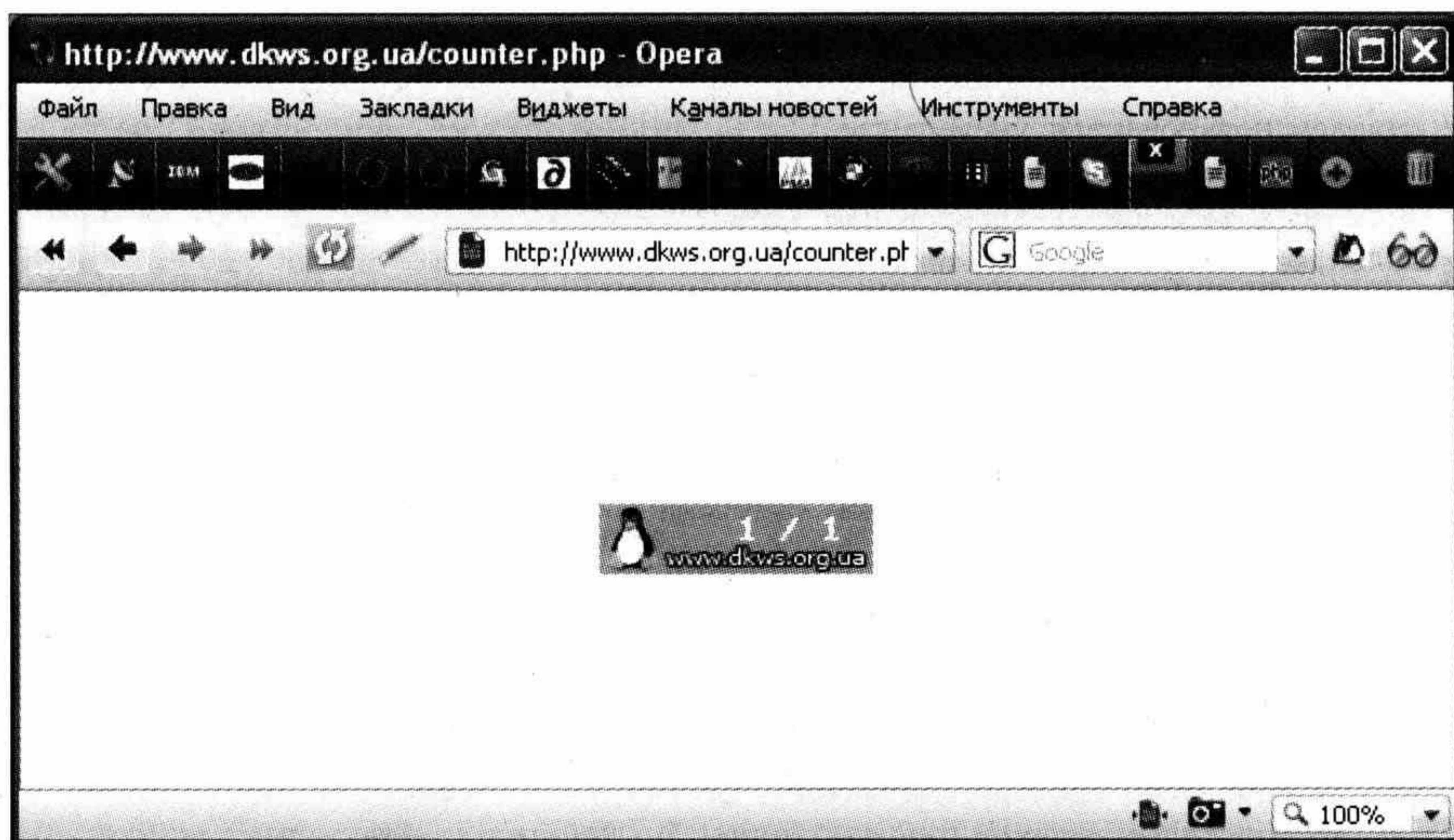


Рис. 40.1. Счетчик в работе

Для использования счетчика в HTML-код вашего сайта нужно вставить следующий тег:

```
<img src=counter.php>
```

Полный код сценария представлен в листинге 40.4.

Листинг 40.4. Сценарий counter.php

```
<?

// подключаем конфигурационный файл
require "config.php";

// подключаемся к MySQL-серверу
mysql_connect($dbhost, $dbuser, $dbpasswd) or die("MySQL error");

mysql_select_db($dbname);

// получаем текущие значения счетчиков
$q = 'select * from counter';

$r = mysql_query($q);

$row = mysql_fetch_array($r);

$daily = $row['daily'];
$total = $row['total'];

// получаем IP-адрес пользователя
$IP = $_SERVER['REMOTE_ADDR'];

// считаем пользователя уникальным
$unique = true;
// есть ли в таблице IP-адресов наш адрес?
$q = "select * from ipaddr where ip = \"$IP\"";
$r = mysql_query($q);

// если IP есть в таблице, пользователь не уникален
if (mysql_num_rows($r)>0) $unique = false;

// если уникален, тогда увеличиваем счетчики и добавляем его IP-адрес
// в таблицу
if ($unique)
{

    $daily++; $total++;

    // обновляем таблицу счетчиков
    $q = "update counter set daily=$daily, total=$total";
    mysql_query($q);
}
```



```
// добавляем новый IP в таблицу адресов
$q = "insert into ipaddr values(\"$IP\")";
mysql_query($q);
}

// выводим как картинку

$img = imagecreatefrompng('dkws.png');

// цвет шрифта — белый
$color = imagecolorallocate($img, 255, 255, 255);

// выводим строку
ImageString($img, 10, 60, 3, "$daily / $total", $color);

// выводим картинку
Header("Content-type: image/png");
ImagePng($img);

?>
```

Нам осталось реализовать сценарий `reset_counter.php`, обнуляющий дневной счетчик.

40.5. Сценарий `reset_counter.php`

Сценарий сброса счетчика достаточно прост, поэтому не нуждается в комментариях (листинг 40.5).

Листинг 40.5. Сценарий `reset_counter.php`

```
<?
// подключаем конфигурационный файл
require "config.php";

// подключаемся к MySQL-серверу
mysql_connect($dbhost, $dbuser, $dbpasswd) or die("MySQL error");

mysql_select_db($dbname);

// обнуляем дневной счетчик
$q = "update counter set daily=0";
mysql_query($q);
```



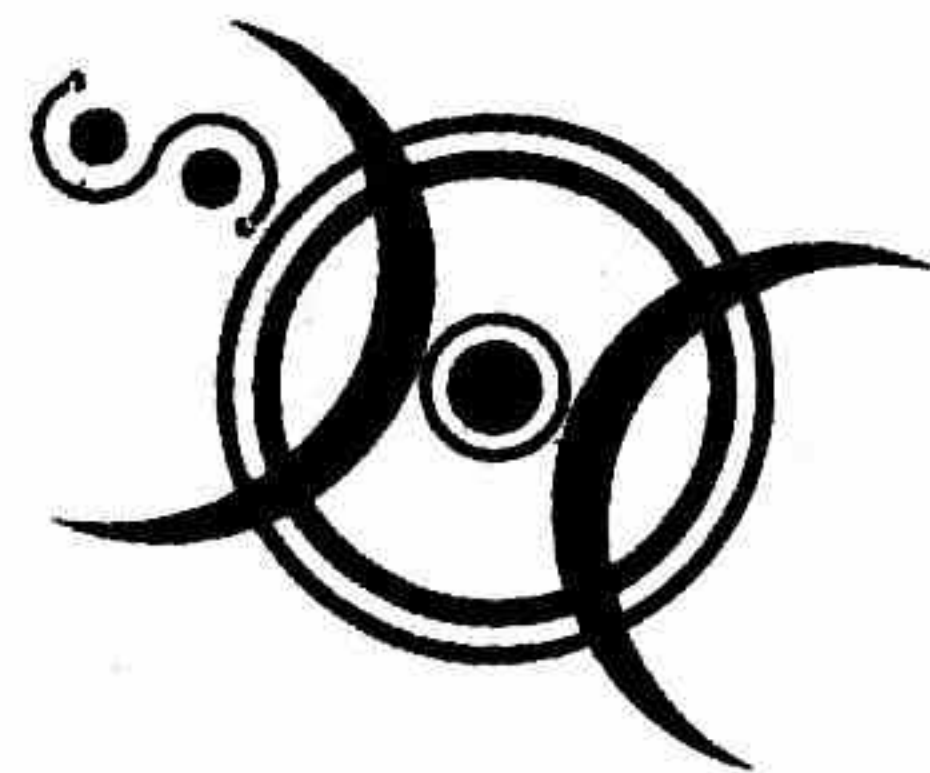
```
// обнуляем таблицу IP-адресов  
$q = "delete from ipaddr";  
mysql_query($q);
```

?>

Запустите этот сценарий. Затем запустите счетчик. Он покажет — 1/2, т. е. 2 посещения всего, 1 за день. Все сходится! Осталось наладить запуск сценария `reset_counter.php` в 00:00. О том, как это сделать, проконсультируйтесь в службе технической поддержки своего провайдера.

ПРИМЕЧАНИЕ

Файлы всех приведенных здесь листингов вы можете найти в каталоге *Glava_40* сопровождающего книгу электронного архива (см. приложение 4). Описание программы `phpMyAdmin` приведено в приложении 1.



ГЛАВА 41

Статистика сайта

41.1. Методы сбора статистики

Понятно, что вам, как любому веб-мастеру, интересно знать все о ваших посетителях: сколько человек посетило сайт за день или за определенный период, какими операционными системами пользуются ваши посетители, какие у них браузеры, откуда они (имеется в виду географическое местонахождение). Анализируя эту информацию, вы можете больше узнать о целевой аудитории своего сайта и предлагать более интересную аудитории информацию. Например, если 90 % вашей аудитории — русскоязычная публика, то английская версия сайта вам точно не нужна. Если же более 30 % аудитории составляют англоязычные пользователи, то стоит задуматься об английской версии сайта.

Существуют два основных метода сбора статистики: анализ журналов сервера и маркировка страниц (page tagging). У каждого метода есть свои преимущества и недостатки. Рассмотрим, в чем заключается каждый способ.

Пользователи постоянно посещают ваш сайт, а веб-сервер для каждого посещения записывает в свой журнал дату и время посещения, IP-адрес пользователя, его агент (браузер), а также запрошенный пользователем файл. Вот пример такой записи из файла журнала веб-сервера:

```
81.19.66.89 - - [06/Aug/2008:04:17:31 +0300] "GET /index.php?photo=17
HTTP/1.1" 200 8583 "-" "StackRambler/2.0 (MSIE incompatible)"
```

Эта запись означает, что пользователь с IP-адресом 81.19.66.89 6 августа 2008 г. в 4:17 посетил страницу /index.php?photo=17. Значение 200 после адреса странички — это код ответа сервера, означающий, что страница была успешно найдена и передана клиенту. Далее мы узнаем браузер пользователя — StackRambler/2.0 (MSIE incompatible). Исходя из названия браузера, мы понимаем, что на наш сайт зашел не человек, а поисковый бот Рамблера.

Метод анализа журналов заключается в том, что вы устанавливаете на сервер (или на свой компьютер) программу-анализатор журнала сервера и "скармливаете" ей журнал сервера. Журнал сервера (он еще называется протоколом доступа) можно получить в панели управления хостингом или скачать по FTP.

Метод маркировки страниц заключается в установке кода счетчика на каждой странице сайта (поскольку мы используем PHP и шаблоны, то код счетчика достаточно установить один раз — в TPL-шаблоне). Затем вам нужно зайти на сайт системы статистики (которая вам и предоставила код счетчика) и просмотреть статистику сайта.

А теперь поговорим о преимуществах и недостатках этих способов. Конечно, для многих пользователей предпочтительнее второй вариант — он более простой. Вы размещаете код счетчика, а затем лишь наведываетесь на страницу статистики и следите за посещаемостью вашего сайта. Но бесплатный сыр бывает лишь в мышеловке! Вы никогда не задумывались, почему в Сети так много бесплатных систем статистики — ведь вы не платите ни копейки за предоставляемые вам услуги?

Представим, что вы владелец магазина компьютерной техники. Даже не интернет-магазина, а самого обычного, с витринами и продавцами-консультантами. У вас есть список постоянных клиентов, которые несколько раз в месяц обязательно делают в вашем магазине крупные покупки. Вы регистрируете каждую покупку и спустя некоторое время уже знаете, что интересует того или иного клиента, и сразу предлагаете ему этот товар. Например, Иванову нужны картриджи для разных принтеров Epson и пачек десять бумаги, а Сидорову — пара сотен DVD-дисков. Есть у вас и постоянный оптовый покупатель ноутбуков. Хотели ли бы вы, чтобы информация о ваших покупателях попала в руки конкурентам? Я думаю, что нет, иначе велика вероятность, что вы потеряете своих покупателей и, следовательно, свой бизнес.

Размещая счетчик посторонней системы статистики на своем сайте, вы бесплатно отдаете ей всю информацию о пользователях сайта, об их интересах и предпочтениях. Если у вас сугубо информационный сайт — например, как у меня, то это еще полбеда. А вот если у вас сайт коммерческий (например, магазина или интернет-магазина), то это равноценно тому, что вы дарите конкурентам своих покупателей.

41.2. Программы-анализаторы журналов веб-сервера

Лучшей программой-анализатором журналов является Webalizer (рис. 41.1). Достаточно часто эта программа устанавливается на хостинг по умолчанию, и получить доступ к ней можно через панель управления хостингом.

Если на вашем хостинге не установлен Webalizer, вы можете бесплатно скачать программу по адресу www.webalizer.org. Программа доступна как для Windows, так и для Linux. Скачав программу, вы можете передать ей журнал сервера, а в результате получите анализ посещаемости сайта в формате HTML:

```
webalizer access.log
```

Кроме программы Webalizer, в Интернете вы можете найти следующие программы, которые основаны на Webalizer:

- Wephpalizer — "улучшенный" Webalizer. Программа обладает лучшей, чем у "предка", визуализацией, а также использует базу данных MySQL для хранения

статистики. Программа доступна по адресу <http://sourceforge.net/projects/wephpalizer/>;

- LogMiner — еще один анализатор журналов серверов Apache и Internet Information Server (вообще — любых серверов, поддерживающих формат журналов Apache или W3C). Для хранения статистики использует базу данных PostgreSQL. На основе собранных данных может строить много различных отчетов. Программа доступна по адресу <http://logminer.sourceforge.net/>;
- AWFFull — использует для работы Webalizer, но добавляет дополнительные отчеты. С помощью AWFFull вы можете создавать пользовательские отчеты. Про-

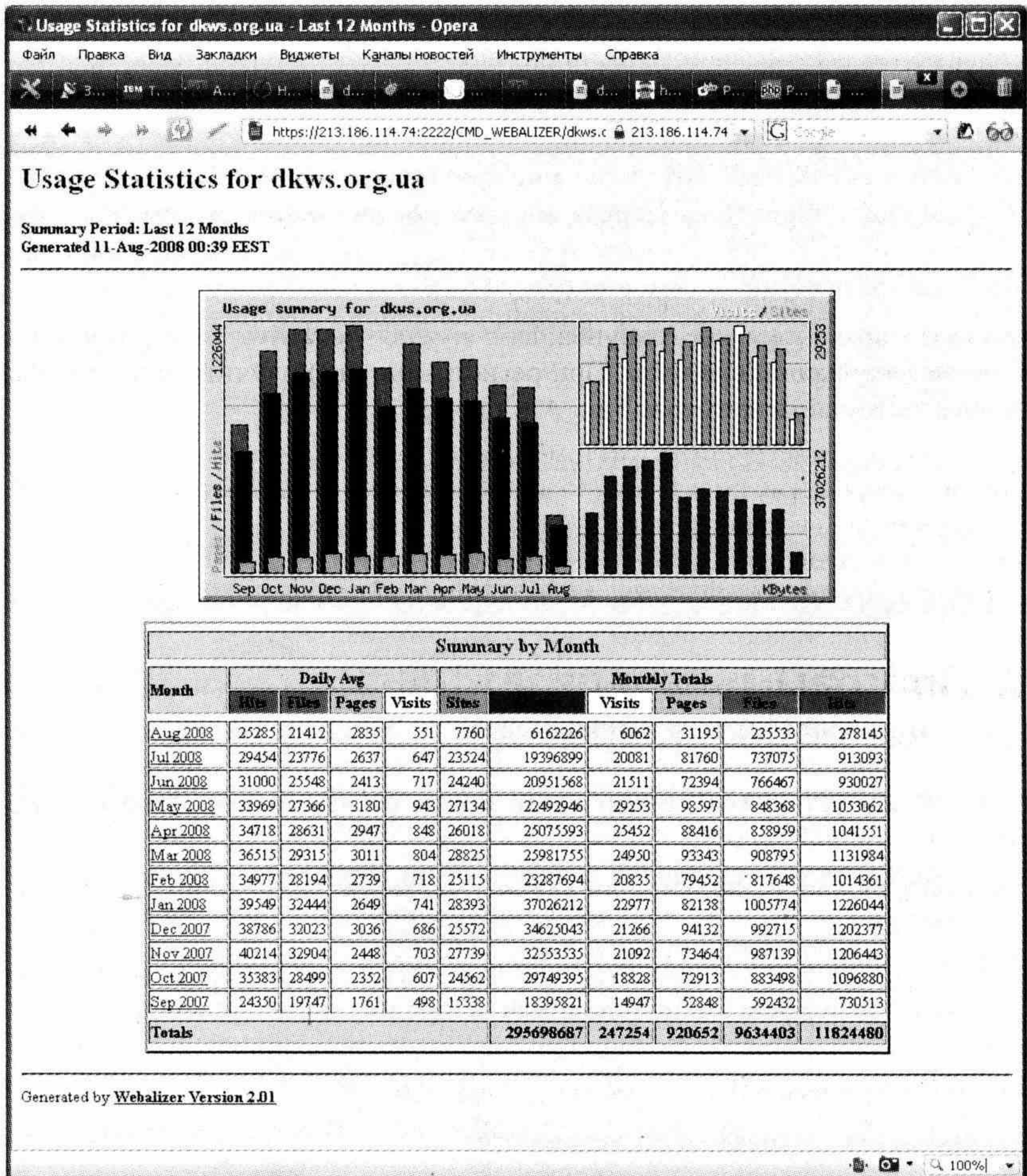


Рис. 41.1. Программа Webalizer

грамма поддерживает GeoIP для определения страны посетителя. Скачать программу можно по адресу www.stedee.id.au/awffull;

- ❑ **AWStats** — полноценный анализатор, написанный на Perl (и поэтому может запускаться в большинстве случаев только на UNIX/Linux). Программа поддерживает следующие форматы журналов: Apache Custom Log format (NCSA combined/XLF/ELF Log Format или common/CLF Log Format), WebStar, W3C Extended Log File Format (для Microsoft IIS). Программа умеет распознавать посещения человека и поискового "паука", а также использует особую технику вычисления статистики использования Flash-, QuickTime-, RealPlayer-роликов и других мультимедиафайлов сайта. Скачать программу можно здесь: <http://awstats.sourceforge.net/>.

41.3. Системы статистики

Из всех систем статистики, которыми мне приходилось пользоваться, я выбрал две: Google Analytics и bigmir)net. Первая система предоставляет огромное количество различных отчетов, которые, может, и не всегда нужны. Но для расширенного анализа посещаемости сайта — самое то (рис. 41.2)!

Для регистрации в Google Analytics вам нужно посетить сайт этой системы: <https://www.google.com/analytics/>. При регистрации вы получите код счетчика, который будет выглядеть примерно так:

```
<script src="http://www.google-analytics.com/urchin.js"
type="text/javascript">
</script>
<script type="text/javascript">
uacct = "UA-xxx-xxx";
urchinTracker();
</script>
```

xxx-xxx — это идентификатор вашего сайта, он будет другим. Заметьте, что код Google, в отличие от других счетчиков, не отображается на странице сайта. Поэтому после его установки дизайн сайта никак не изменится. Довольно удобно, кстати.

Код счетчика нужно установить на каждой странице сайта или один раз в TPL-шаблоне сайта.

Что же касается системы статистики bigmir)net (<http://top.bigmir.net/>), то она обладает удобным интерфейсом пользователя и предоставляет все основные отчеты (рис. 41.3):

- ❑ **Общая статистика сайта** — общая информация;
- ❑ **Посещаемость** — графики посещаемости;
- ❑ **Посетители** — доменные имена или IP-адреса посетителей;
- ❑ **Страницы** — наиболее популярные страницы сайта;

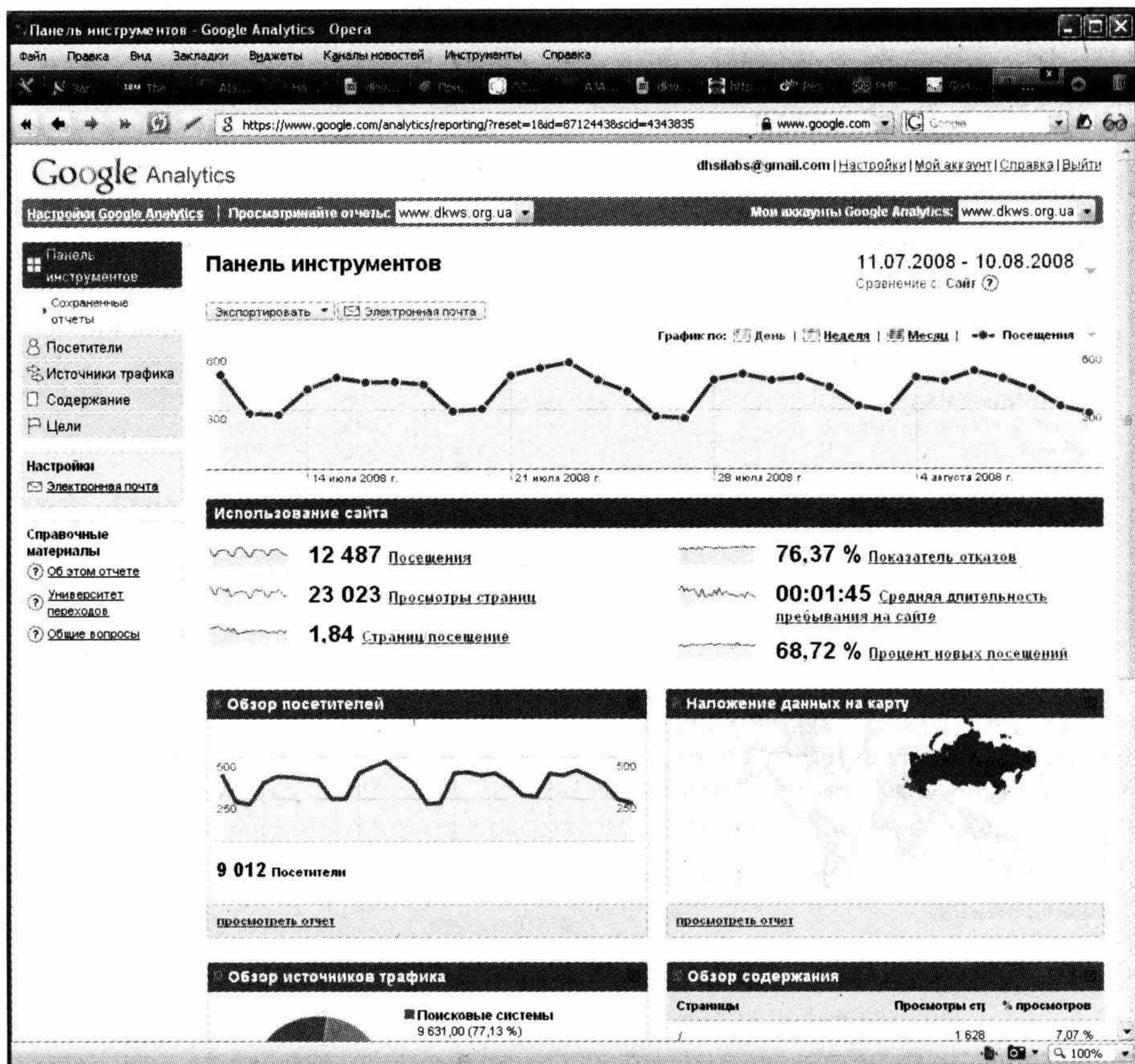


Рис. 41.2. Статистика сайта dkws.org.ua

- ❑ **Откуда пришли** — с каких сайтов пользователи перешли на ваш сайт (наверняка лидерами будут поисковые машины, что вполне закономерно);
- ❑ **О посетителях** — различная информация о посетителях (какие браузеры и какие операционные системы они используют и др.);
- ❑ **География** — информация о географическом местонахождении посетителей сайта;
- ❑ **Сессии** — информация о рабочих сессиях (сколько времени в среднем пользователи проводят на вашем сайте);
- ❑ **Аудитория** — анализ аудитории сайта;
- ❑ **Сравнение аудиторий** — сравнивает аудиторию с другим сайтом;
- ❑ **Сравнение посещаемости** — сравнивает посещаемость с другим сайтом.

Рейтинг сайтов bigmir.net - Opera

Файл Правка Вид Закладки Виджеты Каналы новостей Инструменты Справка

http://top.bigmir.net/stat/127906/

Сделать bigmir.net стартовой | Все сервисы bigmir.net

Почта Войти

bigmir.net Рейтинг сайтов

Начало ТОП-25 Статистика Добавить Изменить Удалить Счетчики Настройки Правила Помощь

Статистика сайта DKWS: настройка Linux

Общая статистика сайта [показать](#)

Статистика посещаемости сайта на 11.08.2008 13:16

Всего запросов к счетчику: 635 445

Всего посещений сайта: 247 956

Номер ID сайта в рейтинге bigmir.net: 127906

Дата регистрации: 26.02.2006

Время последнего вызова счетчика: 11.08.2008 13:15:04

Позиция в рейтинге Компьютеры: 72

Подробная статистика

Показатель	Сегодня	Вчера	За 7 дней	За 30 дней	Всего
Посетителей (хостов)	157	224	2 102	9 883	247 956
Посетителей (cookies)	165	222	2 145	10 267	264 538
Просмотров страниц (хитов)	376	527	4 161	20 042	635 445
Отношение хитов/хостов	2.3	2.3	1.9	2	2.5
Переходов из сайта на рейтинг	n/a	n/a	n/a	1	134
Переходов из рейтинга на сайт	n/a	n/a	n/a	1	48
Сейчас на сайте	2				

Незасчитанные вызовы счетчика

Всего	Сегодня	Вчера	За 7 дней	За 30 дней	Всего
Всего	18	19	133	785	37 803
В том числе:					
Через публичные проху	n/a	2	12	67	2 132
Без поля referer	12	14	86	538	29 693
С чужих страниц	6	3	35	175	5 379
Без указания User-Agent	n/a	n/a	n/a	5	599

Отчеты

Для более полных данных - перейдите на новые отчеты [new](#)

- общая статистика сайта
- посещаемость
- посетители
- страницы
- откуда пришли
- о посетителях
- география
- сессии
- аудитория
- сравнение аудиторий
- сравнение посещаемости

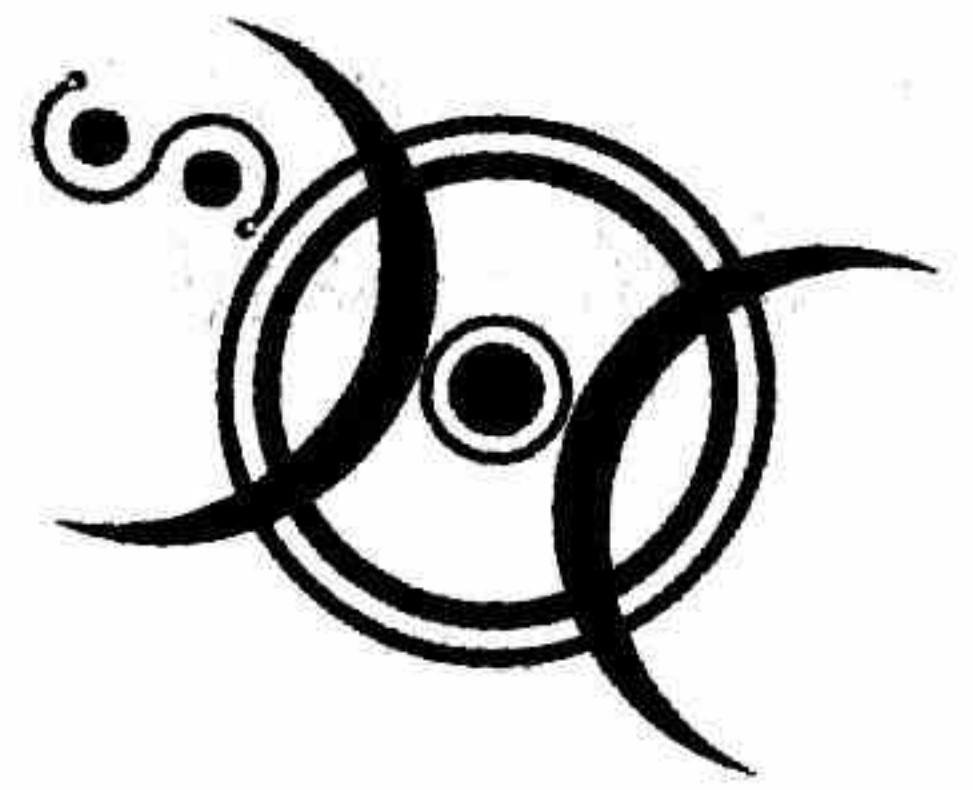
Смотрите также

Код счетчика

Реклама в поиске

100%

Рис. 41.3. Система bigmir.net



ГЛАВА 42

Голосования и комментарии для сайта

42.1. Разработка собственной системы голосования

42.1.1. Общая структура системы

Системы опросов давно стали популярными, и сейчас голосование можно встретить практически на любом сайте. В Интернете можно найти много уже готовых систем голосования — вот одна из них: Advanced Poll. С использованием этих систем, как правило, не возникает проблем, и вы всегда сможете разобраться в них самостоятельно. Здесь же мы поговорим о разработке собственной системы голосования.

Наша система голосований будет состоять из следующих файлов:

- poll_form.php — отображает голосование;
- poll_process.php — производит обработку голосования;
- poll_results.php — отображает результаты голосования пользователю;
- config.php — конфигурационный файл, содержащий параметры базы данных.

Перед тем как приступить к написанию PHP-кода, нужно спроектировать базу данных. В ней предполагаются три таблицы:

- polls — будет содержать вопросы голосований;
- answers — таблица вариантов ответов на вопросы голосований;
- votes — ответы пользователей на вопросы голосований.

Создают эти таблицы следующие SQL-операторы:

```
CREATE TABLE polls (  
ID INT NOT NULL AUTO_INCREMENT,  
question MEDIUMTEXT NOT NULL,  
PRIMARY KEY ( ID )  
);
```

```
CREATE TABLE answers (  
answer_ID INT NOT NULL AUTO_INCREMENT,
```



```
ID INT NOT NULL,
answer MEDIUMTEXT NOT NULL,
PRIMARY KEY ( poll_ID )
);
```

```
CREATE TABLE votes (
ID INT NOT NULL,
answer_ID INT NOT NULL,
INDEX ( ID )
);
```

Поле `answer_ID` таблиц `answers` и `votes` задает номер варианта ответа, а `ID` — номер голосования.

Теперь заполним наши таблицы начальными данными, а именно — добавим голосование:

```
INSERT INTO polls (question) VALUES ("Ваша операционная система?");
INSERT INTO answers (ID, answer) VALUES (1, "Windows");
INSERT INTO answers (ID, answer) VALUES (1, "Linux");
INSERT INTO answers (ID, answer) VALUES (1, "MacOS");
INSERT INTO answers (ID, answer) VALUES (1, "Windows + Linux");
```

Сначала мы добавили текст вопроса в таблицу `polls`. Его номер задается автоматически и будет равен 1. Затем мы добавили четыре ответа для опроса с номером 1. Если вам нужно будет добавить второе голосование, то вы можете сделать это аналогично:

```
INSERT INTO polls (question) VALUES ("Текст второго опроса");
INSERT INTO answers (ID, answer) VALUES (2, "Вариант 1");
INSERT INTO answers (ID, answer) VALUES (2, "Вариант 2");
```

Сценарий `config.php` возьмем тот же, что мы использовали в главах 36 и 40, а именно:

```
<?
```

```
$dbhost = 'localhost';           // имя сервера БД
$dbname = 'db';                  // имя БД
$dbuser = 'user';                // имя пользователя БД
$dbpasswd = 'password';          // пароль пользователя
```

```
// информация о сайте
$title = 'CMS v0.01';
$description = 'Описание сайта';
$info = '<p>&copy; Денис Колисниченко';
```

```
?>
```

42.1.2. Разработка сценария `poll_form.php`

Разработку нашей системы опросов начнем со сценария `poll_form.php`. Этот сценарий довольно прост — ему нужно передать идентификатор голосования, и он отобразит указанный опрос.

Прежде всего подключимся к базе данных и получим идентификатор голосования:

```
<?php

require_once "config.php";

@mysql_connect($dbhost, $dbuser, $dbpasswd) or
die("Не могу подключиться к БД");

@mysql_select_db($dbname) or die("Не могу выбрать БД");

// получаем номер голосования
$poll = $_GET['poll'];
if (!is_numeric($poll)) die("Номер голосования указан неверно!");
```

После этого нужно отобразить варианты ответа или, если пользователь уже проголосовал, перенаправить его на результаты голосования. Если пользователь уже проголосовал, то мы установим Cookies с именем `vote_ID`, где `ID` — идентификатор голосования:

```
$q = "SELECT polls.question, answers.answer, answers.answer_ID
FROM polls, answers
WHERE polls.ID = $poll
AND answers.ID = polls.ID";

$r = mysql_query($q);

if (mysql_num_rows($r) == 0) {
die('Ошибка!');
}

// пользователь уже голосовал
if ($_COOKIE["vote_$poll"]) {
header("Location: poll_results.php?poll=$poll");
exit;
}

// пользователь не голосовал, поэтому выводим форму опроса
$questions = "";
```

```
while($row = mysql_fetch_array($r)) {
$question = $row['question'];
$questions .= '<li><input name="answer" type="radio" value="' .
$row['answer_ID'] . '"> ' . $row['answer'] . '</li>';
}
```

Все, что нам осталось, — вывести HTML-код:

```
echo "<h1>$question</h1>";
echo "<form action=poll_process.php method=post>";
echo "<p><ul style=\"list-style-type: none;\">
```



```
$questions</ul>";
echo "<input name=poll type=hidden value=$poll>
<input type=submit value=\"Голосовать!\">
</form>";
?>
```

Этот сценарий только выводит голосование (рис. 42.1). Обработкой голосования занимается наш следующий сценарий — `poll_process.php`.

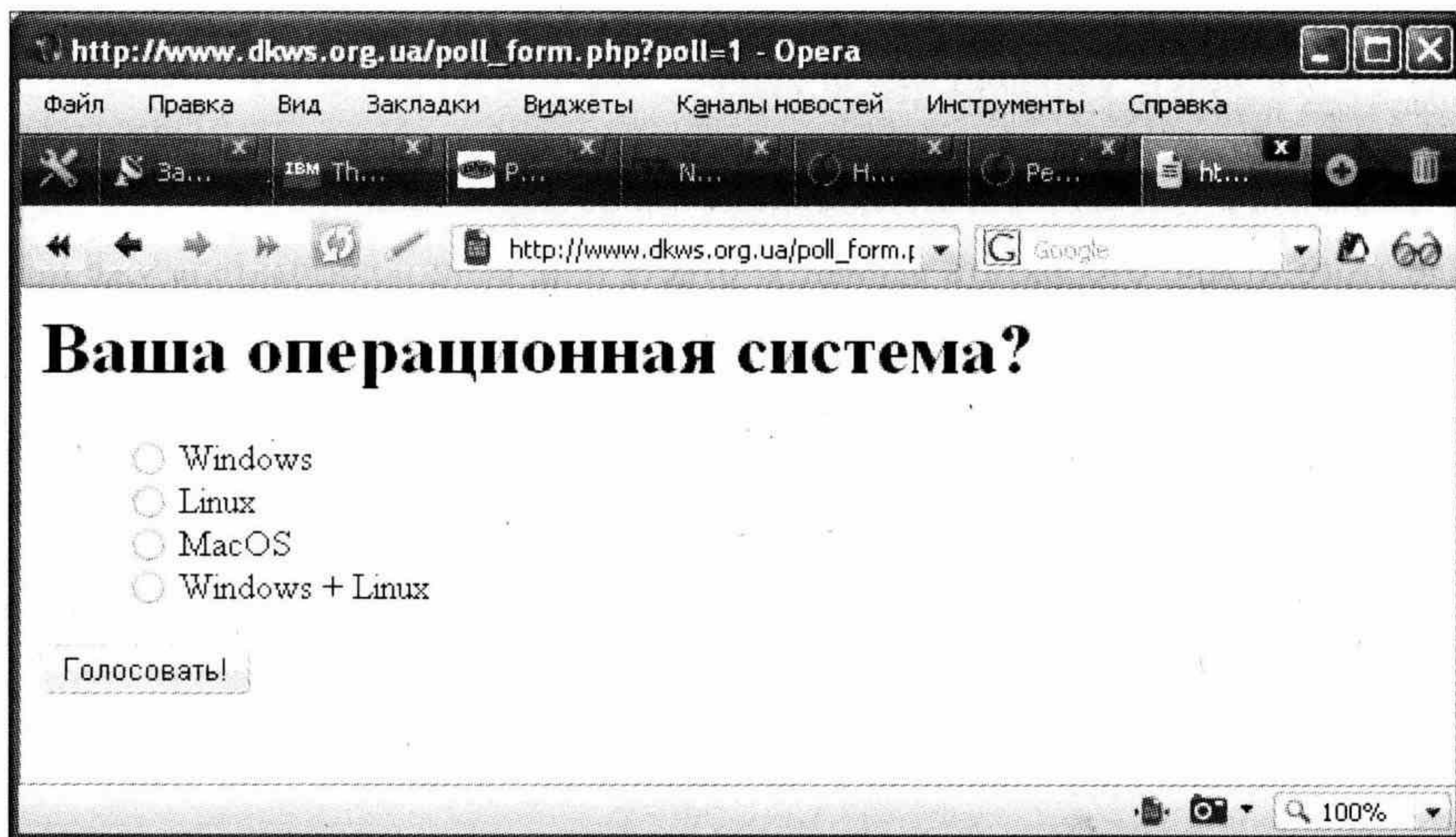


Рис. 42.1. Форма голосования

42.1.3. Сценарий `poll_process.php`

Задача этого сценария — добавление ответа пользователя в базу данных. Сценарию передаются два параметра: номер голосования и номер ответа. Нужно убедиться, что эти параметры являются числами, а не чем-либо еще:

```
<?php

require_once "config.php";
@mysql_connect($dbhost, $dbuser, $dbpasswd) or
die("Не могу подключиться к БД");

@mysql_select_db($dbname) or die("Не могу выбрать БД");
// получаем номер голосования
$poll = $_POST['poll'];
if (!is_numeric($poll)) die("Номер голосования указан неверно!");

$answer = $_POST['answer'];
if (!is_numeric($answer)) die("Номер ответа указан неверно!");
```


Но такой проверки явно маловато. Например, злоумышленник может передать параметры `poll=100` и `answer=50`. Но в нашей базе данных нет голосования с номером 100, да и ответы для каждого вопроса не превышают 10. Поэтому мы должны просмотреть базу данных и проверить, есть ли в ней объекты с указанными идентификаторами:

```
$q = "SELECT answers.answer_ID
FROM polls, answers
WHERE polls.ID = answers.ID
AND polls.ID = $poll
AND answers.answer_ID = $answer";

$r = @mysql_query($q) or die (mysql_error());

if (mysql_num_rows($r) == 0) {
die('Неправильный номер вопроса или ответа');
}
```

Если пользователь еще не голосовал, то нам нужно добавить в базу данных его ответ и установить Cookie-переменную, которая не позволит пользователю неделю (7 дней) участвовать в опросах сайта:

```
if (!$_COOKIE["vote_$poll"]) {
// добавляем информацию в базу данных
$q = "INSERT INTO votes ( answer_ID , ID)
VALUES ($answer, $poll)";
$r = @mysql_query($q);

    setcookie("vote_$poll", "1", time() + (3600*24 * 7));
}
```

После этого мы отправляем пользователя на результаты голосования:

```
header("Location: poll_results.php?poll=$poll");
?>
```

42.1.4. Сценарий `poll_results.php`

Теперь осталось вывести результаты голосования. Ради упрощения сценария для построения графиков мы не будем использовать возможности GD, а обратимся к HTML:

```
<?php

require_once "config.php";

@mysql_connect($dbhost, $dbuser, $dbpasswd) or
die("Не могу подключиться к БД");

@mysql_select_db($dbname) or die("Не могу выбрать БД");
```



```
// получаем номер голосования
$poll = $_GET['poll'];
if (!is_numeric($poll)) die("Номер голосования указан неверно!");

// ищем вопрос голосования
$q = "SELECT question
FROM polls
WHERE ID = $poll";
$r = @mysql_query($q);

if (mysql_num_rows($r) != 1) {
die('Ошибка!');
}
$row = mysql_fetch_array($r);
$question = $row["question"];
```

Далее нужно получить общее количество проголосовавших, чтобы вычислить процентное соотношение между вариантами ответов опроса:

```
$q = "SELECT count(*) AS total_votes
FROM votes
WHERE votes.ID = $poll";

$r = @mysql_query($q);
$row = mysql_fetch_array($r);

$total_votes = $row["total_votes"];
```

Затем получим счетчики каждого голоса. Для этого воспользуемся следующим сложным запросом. Если вы не понимаете, что он делает, то прочитайте внимательнее главу 18 или просто не вникайте в него:

```
$q = "SELECT answers.answer, answers.answer_ID, count(votes.answer_ID)
as num_votes
FROM answers
LEFT JOIN votes
ON votes.ID = answers.ID
AND votes.answer_ID = answers.answer_ID
WHERE answers.ID = $poll
GROUP BY answers.answer_ID
ORDER BY num_votes DESC, answers.answer ASC
";

$r = @mysql_query($q);
```

Теперь нам осталось только вывести HTML-код с результатами голосования:

```
echo "<html><head><title>$question</title></head><body>";
echo '<ul style="list-style-type: none; font-size: 12px;">';
echo '<li style="font-weight: bold; padding-bottom: 10px;">';
echo "$question";
echo '</li>';
```



```

while ($row = mysql_fetch_array($r)) {
if ($total_votes != 0) {
$p = sprintf("%.2f", 100.0 * $row["num_votes"] / $total_votes);
} else {
$p = "0";
}
$width = strval(1 + intval($p)) . "px";
echo '<li style="clear: left;">';
echo "$row[answer]";
echo "</li>";
echo '<li style="clear: left; padding-bottom: 7px;">';
echo '<div style="width: ' . $width . '; height: 15px; ' .
'; background: black; margin-right: 5px; float: left;">' .
"</div>$p%";
echo '</li>';
}

echo '<li style="clear: left;">';
echo "Всего проголосовало: $total_votes";
echo '</li>';
echo '</ul>';
echo '</body></html>';
?>

```

Наша система голосований готова. В качестве домашнего задания вам предстоит разработать панель управления для созданной нами системы. А пока, зная структуру базы данных, вы можете добавлять новые опросы с помощью программы phpMyAdmin. Для вывода результатов голосования (рис. 42.2) можно использовать библиотеку GD.

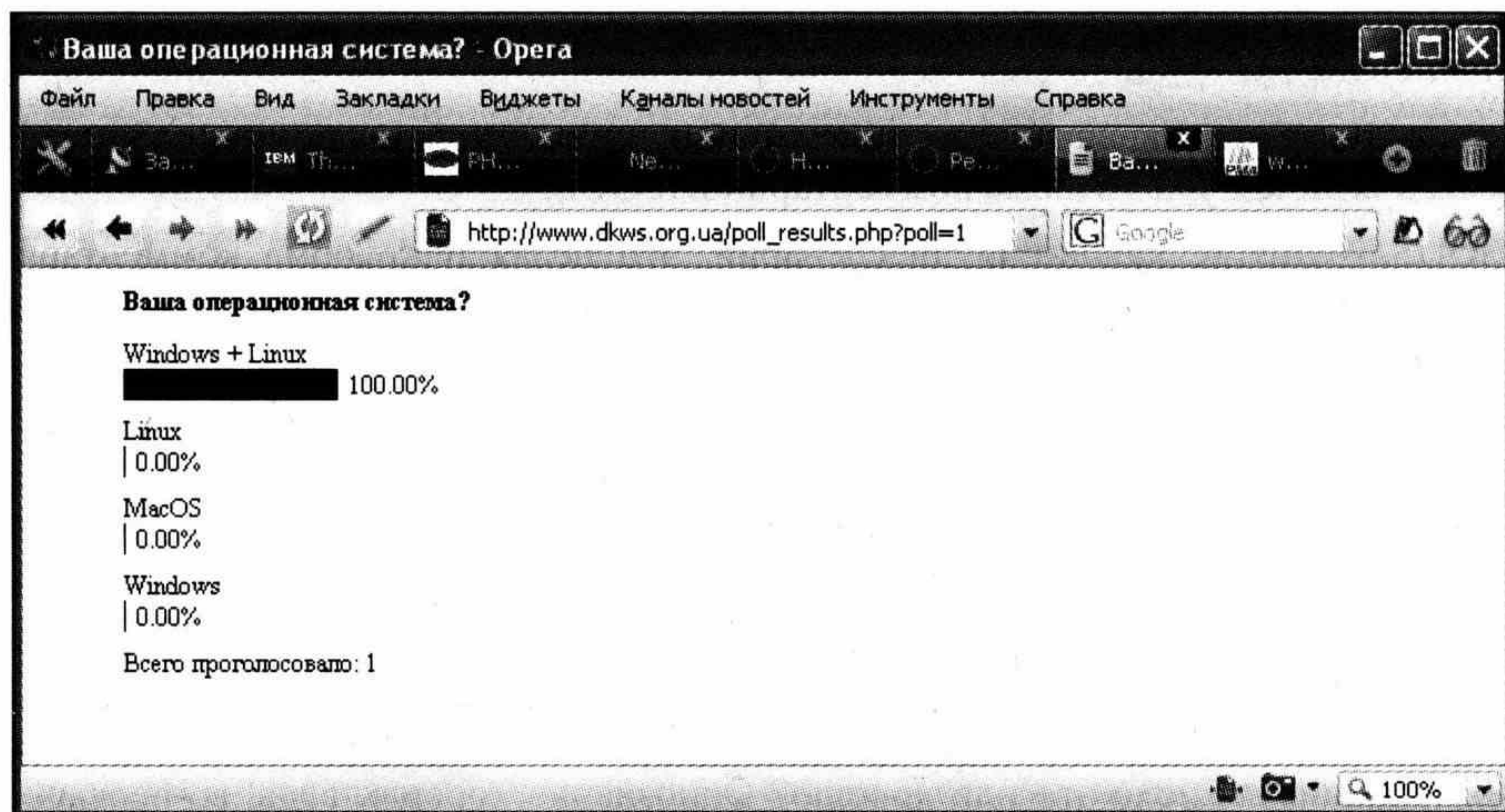


Рис. 42.2. Результаты голосования

ПРИМЕЧАНИЕ

Файлы всех приведенных здесь листингов вы можете найти в каталоге *Glava_42* сопровождающего книгу электронного архива (см. приложение 4). Описание программы *phpMyAdmin* приведено в приложении 1.

42.2. Комментарии DISQUS

42.2.1. Описание платформы

Комментарии — важнейший компонент любого сайта, позволяющий эффективно удерживать аудиторию. Вы можете часто не обновлять свой сайт, но пользователи все равно будут заходить на него в поисках новых комментариев, ответов на свои комментарии и т. д. Вспомните, сколько раз за день вы заходите на один и тот же сайт, если ведется интересная дискуссия? Некоторые пользователи проводят на сайте полдня, читая комментарии и отвечая на них.

Система комментариев — более сложная, чем система опросов. С опросами все значительно проще — отобразил список вариантов ответов, сохранил выбор пользователя. При этом нужно только проверить, что пользователь еще не участвовал в этом опросе, чтобы избежать "накрутки" одного из вариантов. Конечно, в нашем случае накрутку осуществить легко — достаточно использовать другой браузер или очистить Cookies, и можно голосовать повторно. Впрочем, при желании можно добавить еще и проверку по IP-адресу — что не так уж и трудно.

С комментариями сложнее. Ведь нужно оберегать свой сайт от многочисленных спамеров, надо предусмотреть аутентификацию пользователя (чтобы можно было установить, кто именно написал комментарий) и т. д.

В предыдущем разделе мы разработали собственную систему опросов, хотя могли бы воспользоваться одной из многочисленных уже готовых систем, — например, той же DRBPoll.

В случае с комментариями мы пойдем иным путем, чем сэкономим собственное время, и возьмем уже готовые комментарии DISQUS.

DISQUS — это не какой-нибудь сценарий, это целая платформа комментариев. Рассмотрим основные особенности этой платформы и вам станет понятно, почему я выбрал именно ее.

- ❑ Интеграция с любым сайтом. Вы можете интегрировать DISQUS в свой сайт, написанный на PHP, на обычном HTML, а также добавить комментарии DISQUS к любому готовому движку, — например, Joomla или WordPress. Ну, у WordPress достаточно мощная собственная система комментариев (не знаю даже, зачем устанавливать сторонние комментарии для WordPress), а вот для Joomla комментарии будут весьма кстати.
- ❑ Комментарии хранятся на удаленном сервере DISQUS, чем не забивают вашу базу данных. Плохо это или хорошо? Однозначно хорошо, ведь, во-первых, не расходуется место на вашем сайте. Во-вторых, вам не нужно создавать отдель-

ную базу данных, настраивать систему комментариев. Все, что придется сделать, — добавить код системы на ваш сайт. При этом у вас может быть самый дешевый хостинг вообще без поддержки баз данных! Такое в наше время встречается редко, но все же встречается.

- ❑ Для управления комментариями предоставляется удобная панель управления.
- ❑ Автоматическая публикация комментариев. Но можно также включить режим модерирования, когда вы или другой модератор будете одобрять написанные пользователями комментарии, — как, например, в WordPress.
- ❑ Как уже было отмечено, простота установки.
- ❑ Поддержка сервиса Akismet для защиты комментариев от спама.
- ❑ Интеграция с социальными сетями Facebook и Twitter.
- ❑ Различные способы аутентификации пользователя, включая OpenID, Google (gmail), Yahoo, Facebook, Twitter.

Используя DISQUS, вы получите универсальную систему комментариев. Поэтому для своего сайта я выбрал именно ее (рис. 42.3).



Рис. 42.3. Комментарии DISQUS на моем сайте dkws.org.ua

42.2.2. Установка комментариев на свой сайт

Прежде всего на сайте DISQUS: <https://disqus.com/admin/register/> — надо зарегистрироваться. Предлагается ввести адрес своего сайта, его название и короткое имя сайта (рис. 42.4). Короткое имя очень важно — по сути, это ваш идентификатор

Рис. 42.4. Регистрация на сайте DISQUS

в системе DISQUS. Далее (на рисунке не показано) нужно будет ввести e-mail и пароль для доступа.

Далее вам будет предложено установить основные параметры комментариев (рис. 42.5):

- ☐ **Language** — язык пользовательского интерфейса (панель управления комментариями всегда будет на английском);
- ☐ **Twitter @Replies** — можете указать имя пользователя Twitter для отправки комментариев;
- ☐ **Enable Like Buttons** — включить кнопку **Like** (Нравится).
- ☐ **Media Attachments** — пользователям будет разрешено добавлять мультимедиа-вложения к своим комментариям;
- ☐ **Mentions** — разрешает пользователям уведомлять других пользователей о комментариях.

Остальные параметры, в том числе Akismet, пока не включайте, а просто нажмите кнопку **Continue**, и вы перейдете в панель управления комментариями (рис. 42.6). Понятно, что пока комментариев у вас не будет, т. к. вы только что зарегистрировались.

Теперь получим код для установки комментариев на свой сайт. Нажмите ссылку **Install** в верхнем правом углу вкладки **Moderate** (см. рис. 42.6) и в открывшемся окне выберите движок вашего сайта. Поскольку движок у нас собственной разработки, то следует выбрать **Universal Code** (рис. 42.7).



Рис. 42.5. Основные параметры

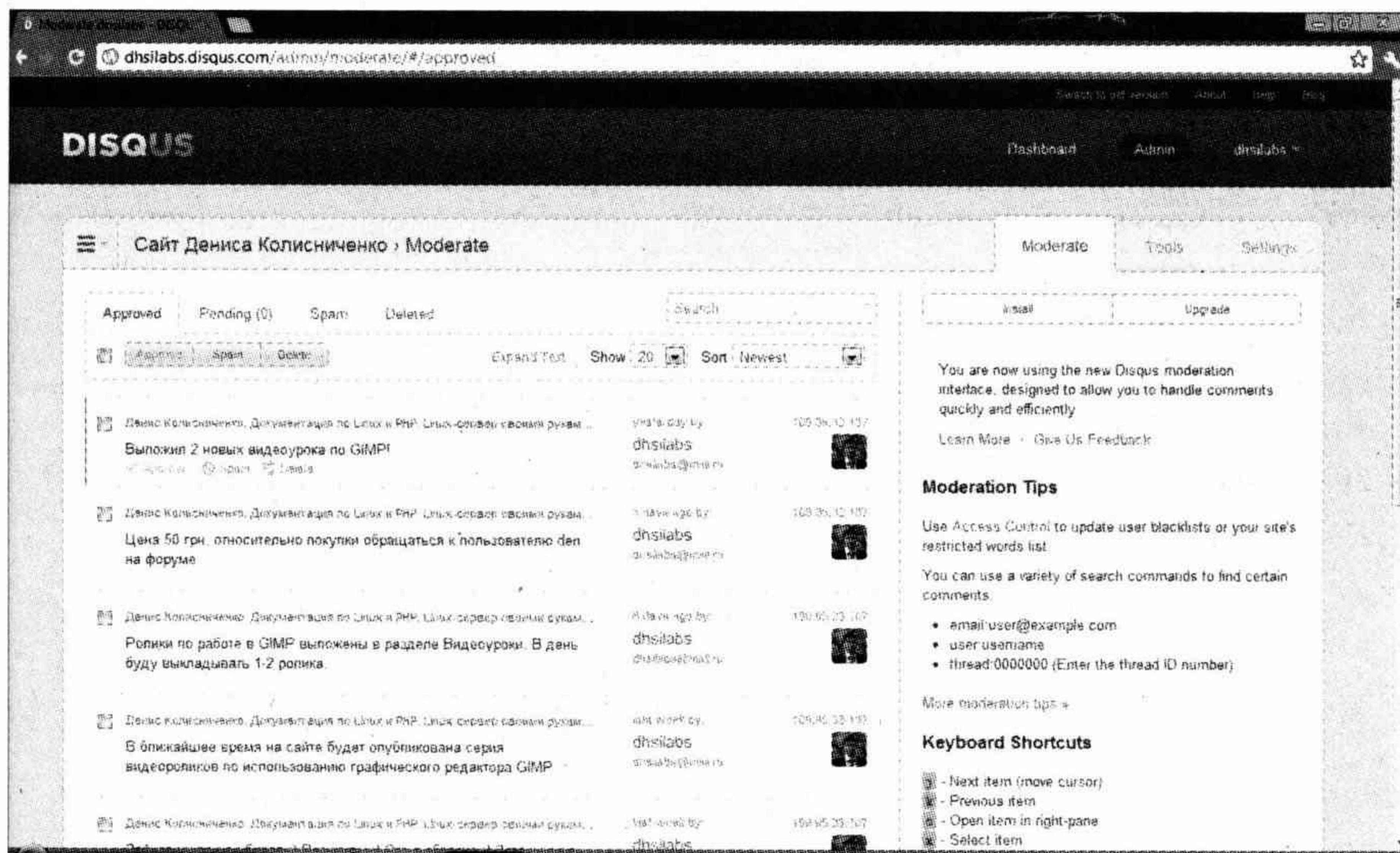


Рис. 42.6. Панель управления комментариями

Далее вы увидите код (рис. 42.8), который нужно скопировать и вставить в то место, где должны быть комментарии. Если вы используете шаблоны, то просто скопируйте код в шаблон без изменений. Если вы предпочитаете выводить код через PHP, тогда используйте одинарные кавычки для вывода кода:

```
echo 'код';
```

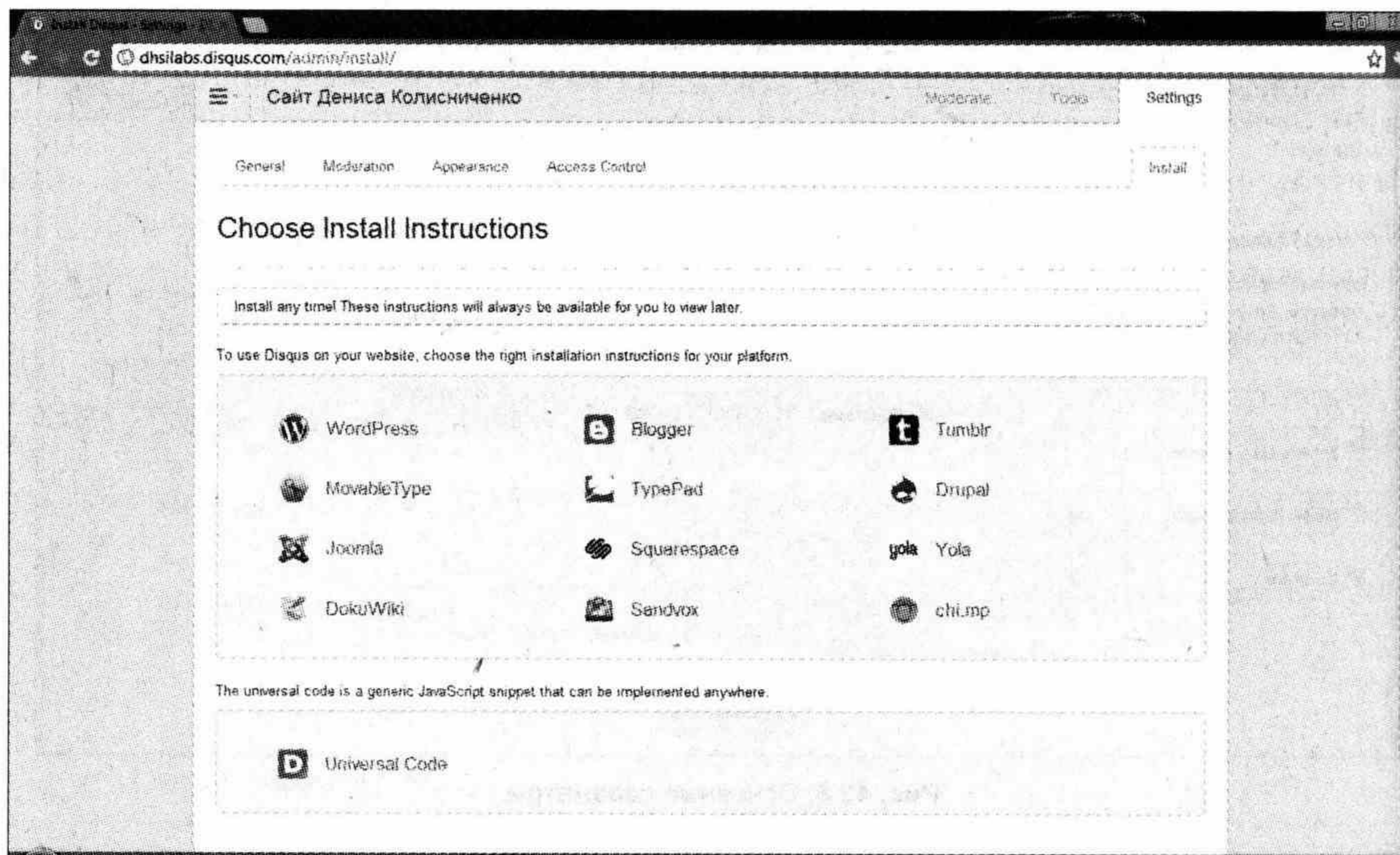



Рис. 42.7. Выберите тип кода DISQUS

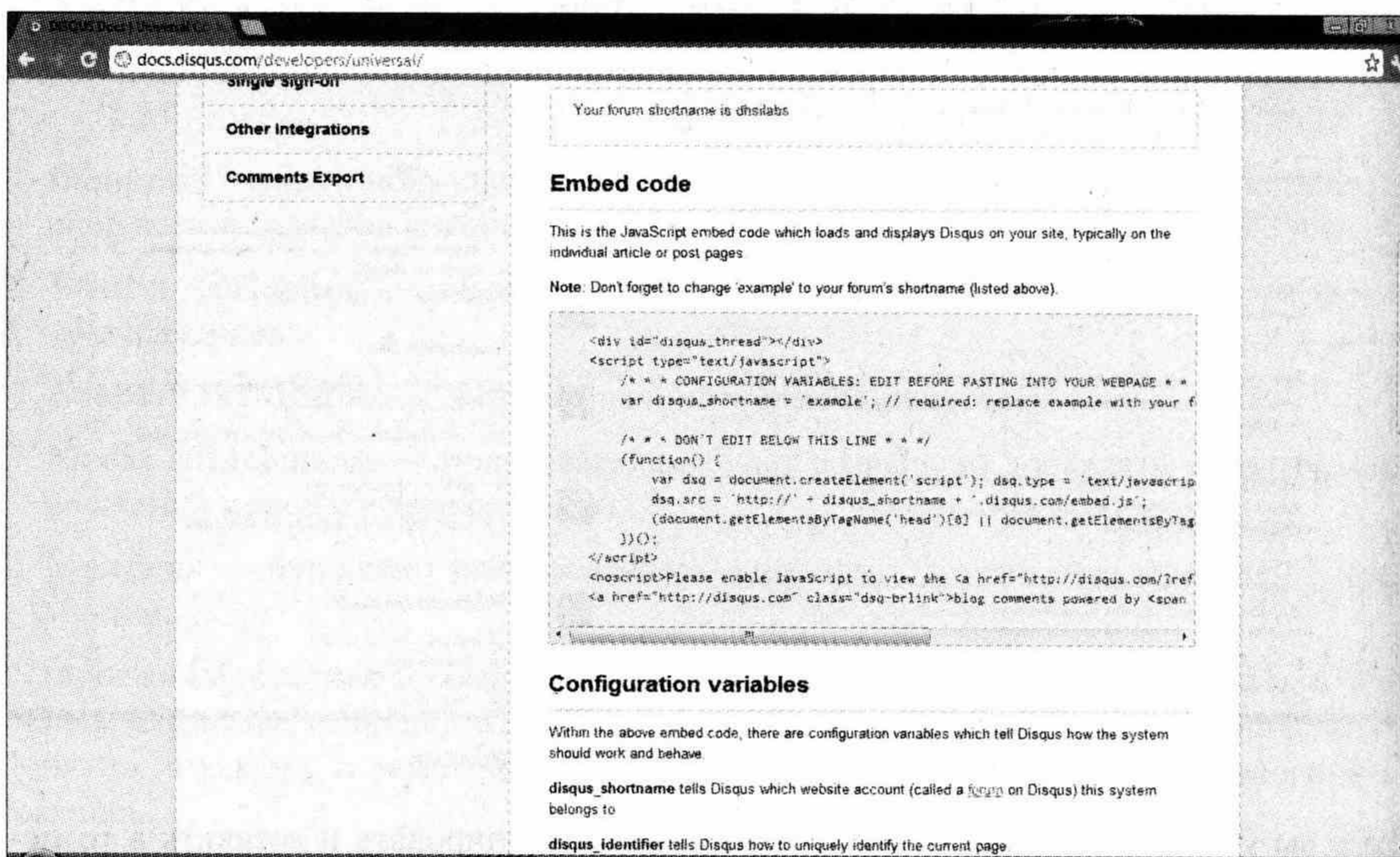


Рис. 42.8. Код для установки на сайт

Сам код здесь я не привожу, поскольку он привязан к короткому имени сайта. Если вы скопируете мой код на свой сайт, то увидите комментарии к моему сайту на своем, а это не то, что вам нужно.

На некоторых сайтах размещают счетчики комментариев. Обычно такие счетчики расположены где-то внизу страницы, до тега `</body>`. Пролистайте страницу размещения кода и вы найдете код **Comment count** — это как раз и есть счетчик комментариев (рис. 42.9).

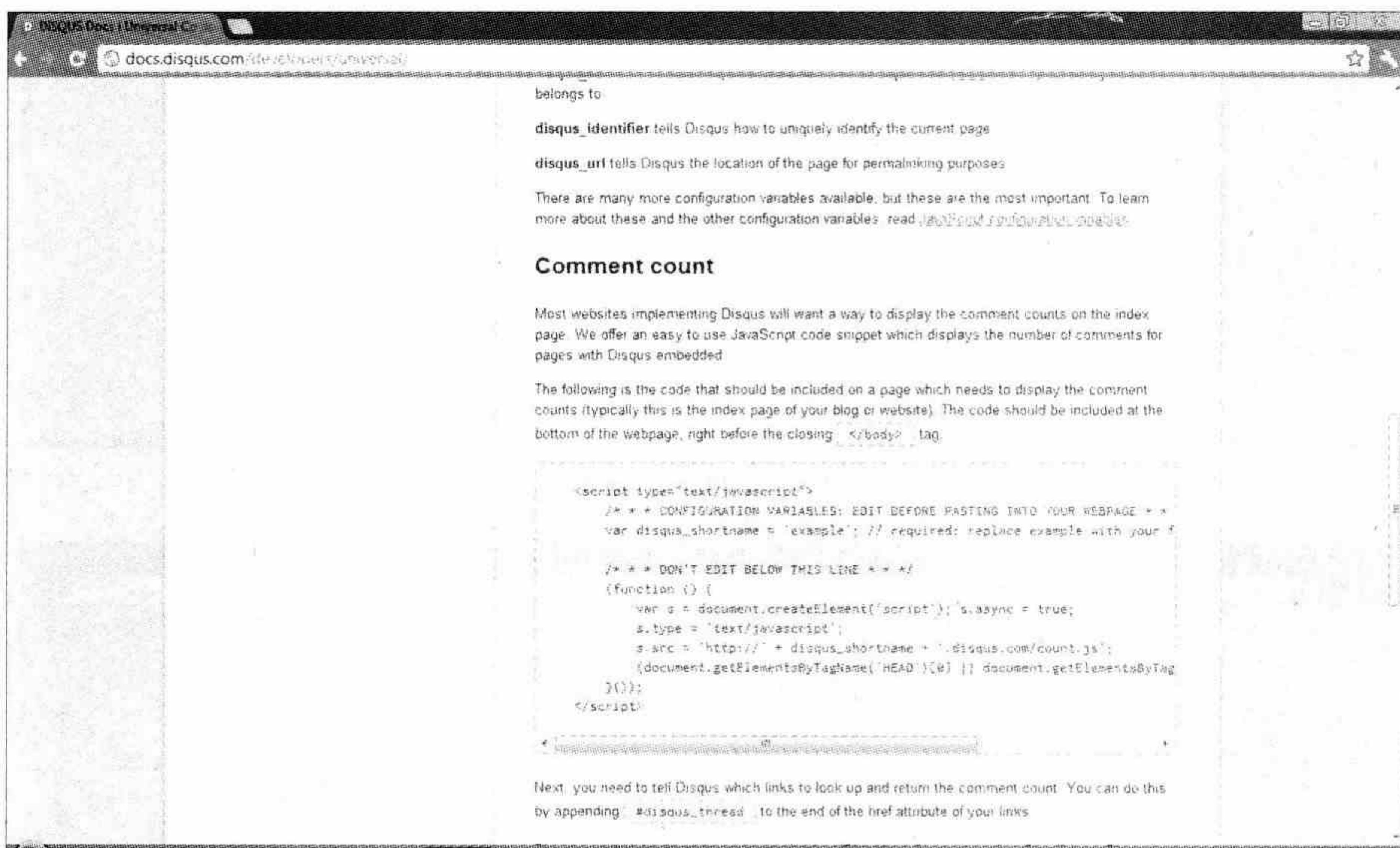


Рис. 42.9. Код счетчика комментариев

Но и это еще не все. На выбор вы можете установить несколько виджетов для своего сайта. Перейдите на вкладку **Tools**, а на ней — на вкладку **Widgets** (рис. 42.10). Здесь вы можете установить следующие виджеты:

- ❑ **Combination** — комбинированный виджет, состоит из трех следующих;
- ❑ **Recent Comments** — отображает последние комментарии;
- ❑ **Popular Threads** — популярные потоки (наиболее комментируемые страницы сайта);
- ❑ **Top Commenters** — лучшие комментаторы.

Собственно, практически все. С панелью управления, думаю, разберетесь сами. Я только подскажу, как активировать Akismet для защиты от спамеров. Перейдите на вкладку **Settings | General**, пролистайте страничку параметров вниз и включите параметр **Enable Akismet** (рис. 42.11). Затем нужно будет получить на сайте <http://akismet.com/> API-ключ (абсолютно бесплатно) и ввести его в поле **API Key**.

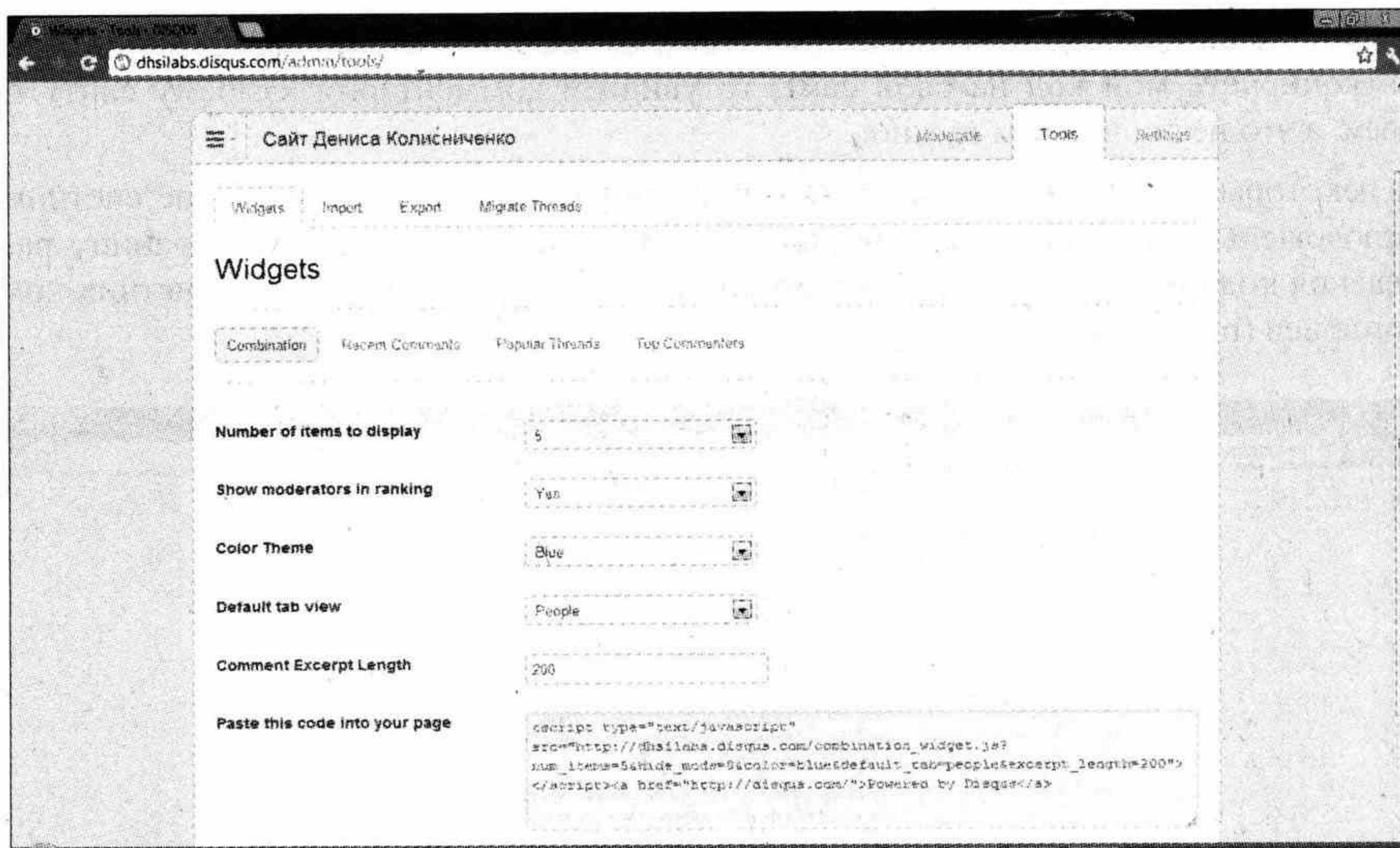


Рис. 42.10. Виджеты

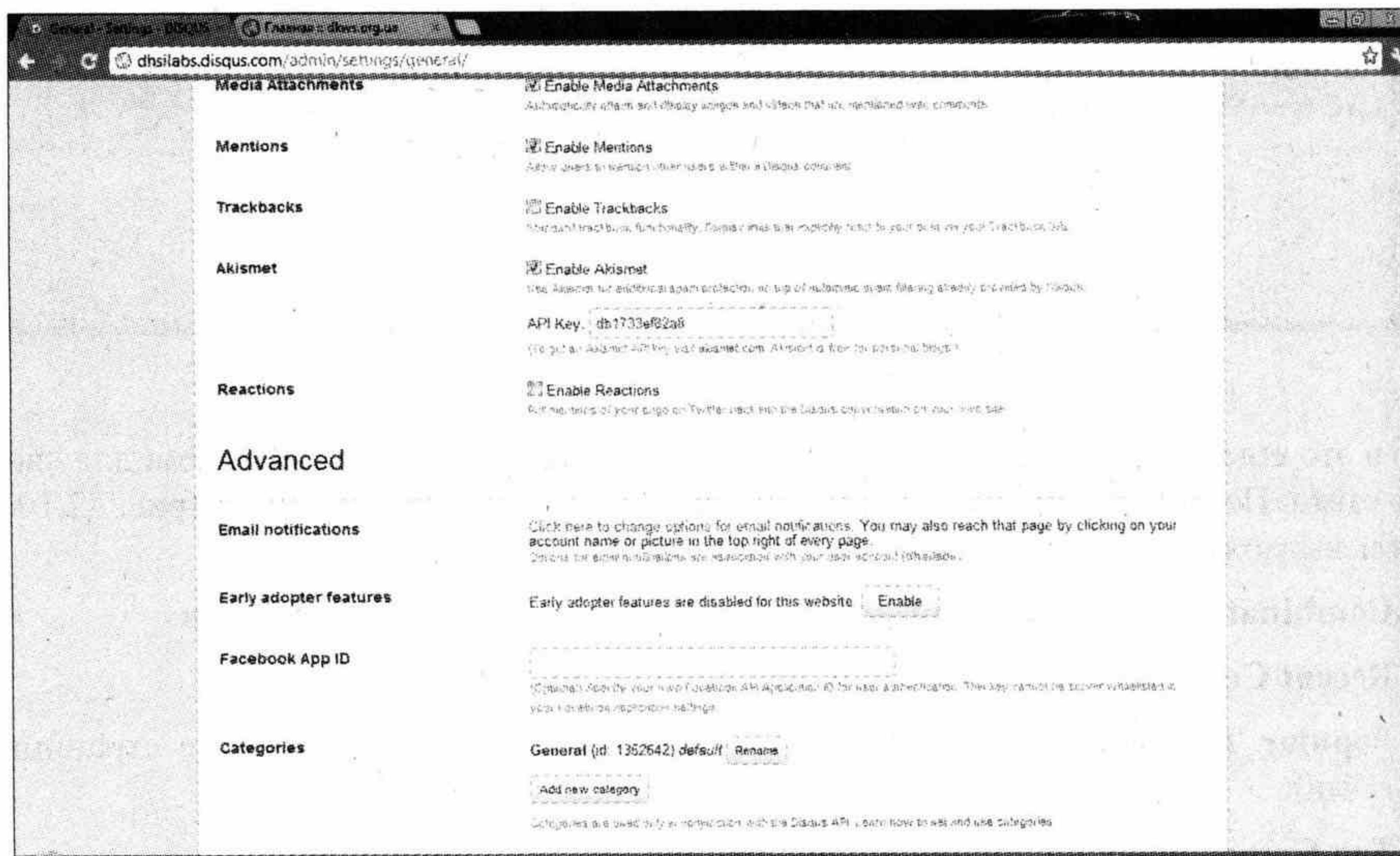
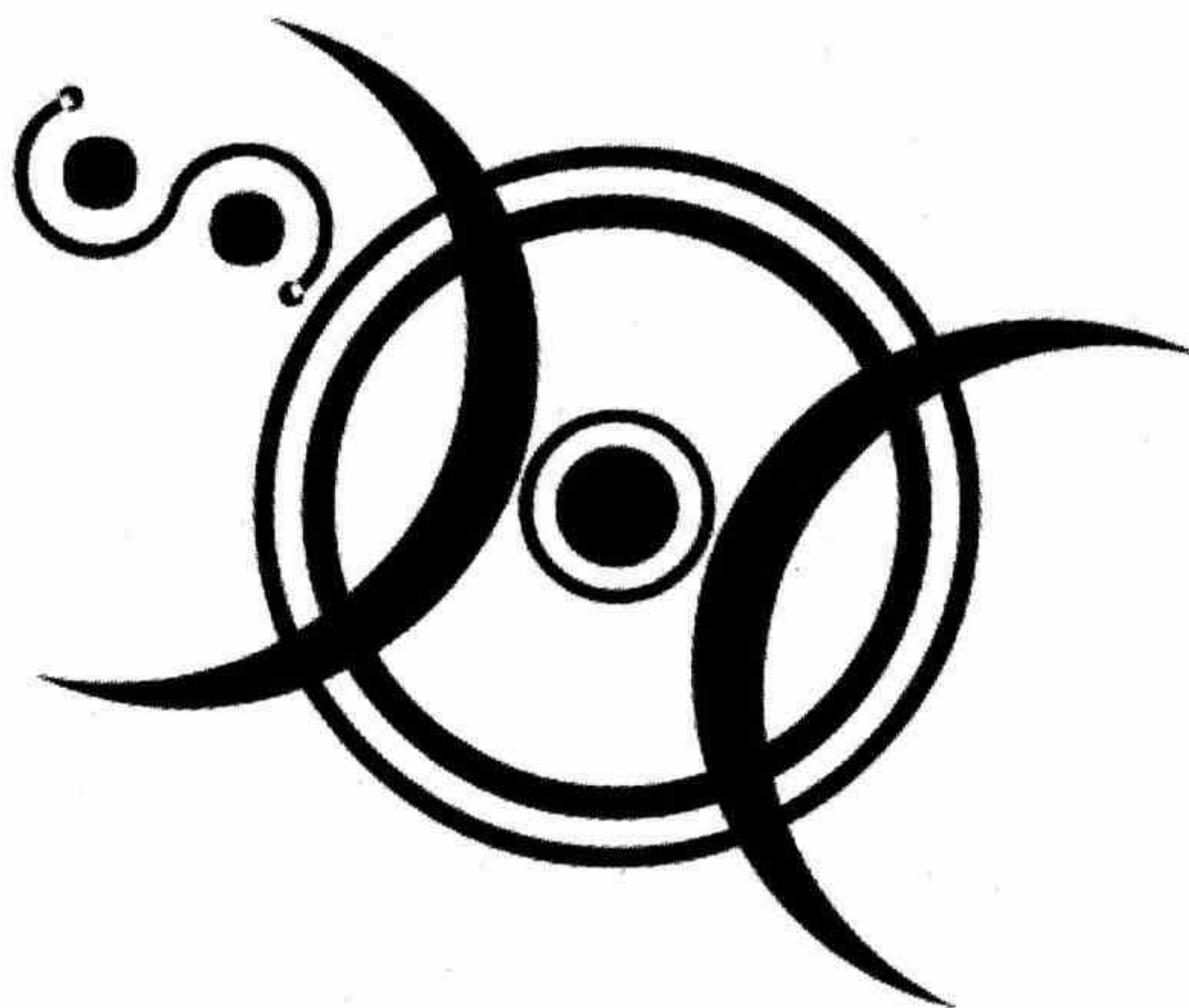


Рис. 42.11. Активация Akismet

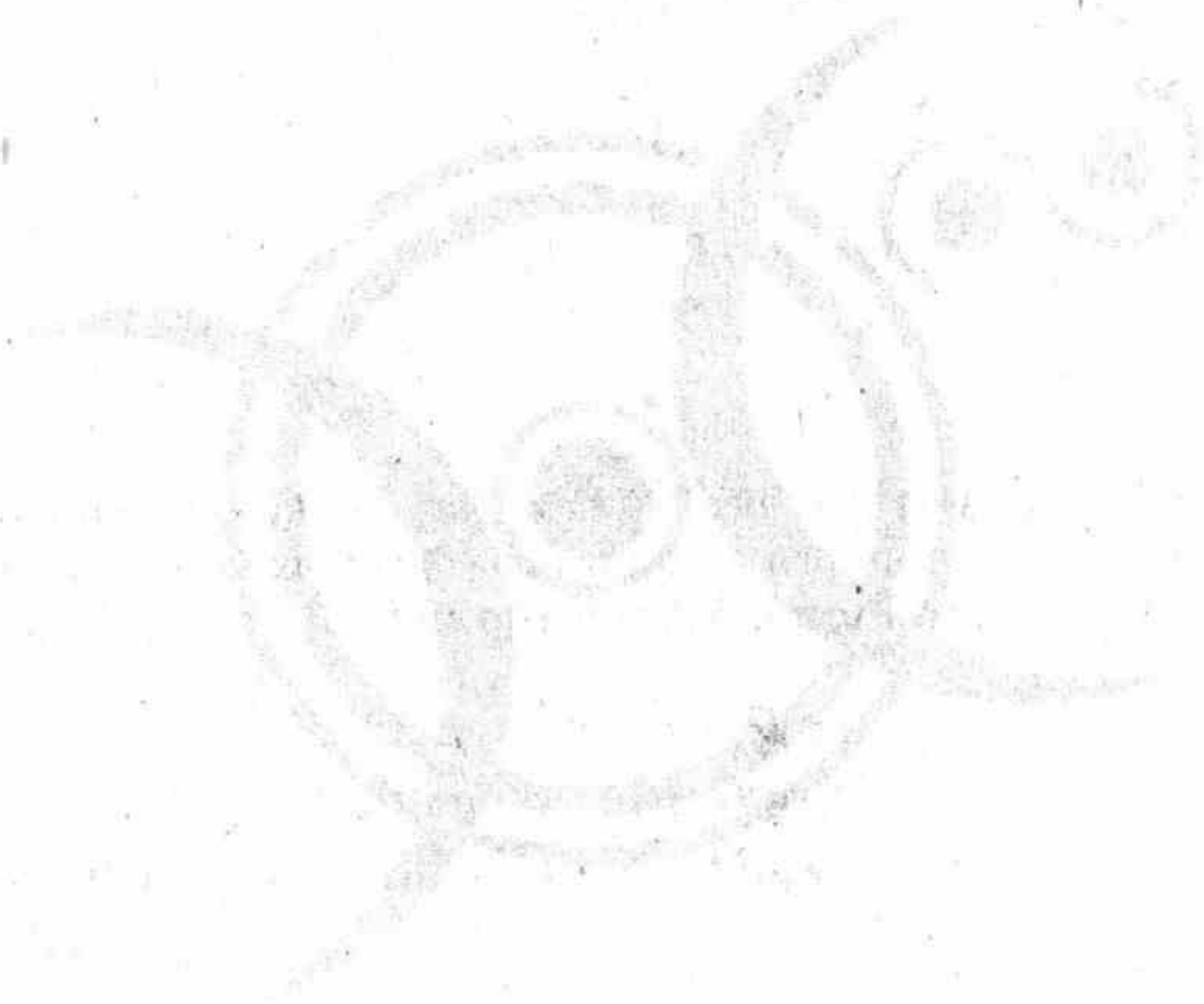
Вот теперь, действительно, все. Когда вы закроете окно браузера, заново получить доступ к панели управления можно будет по адресу: **http://<короткое имя>.disqus.com**.



II. РАЗДЕЛ 9

Сложный проект

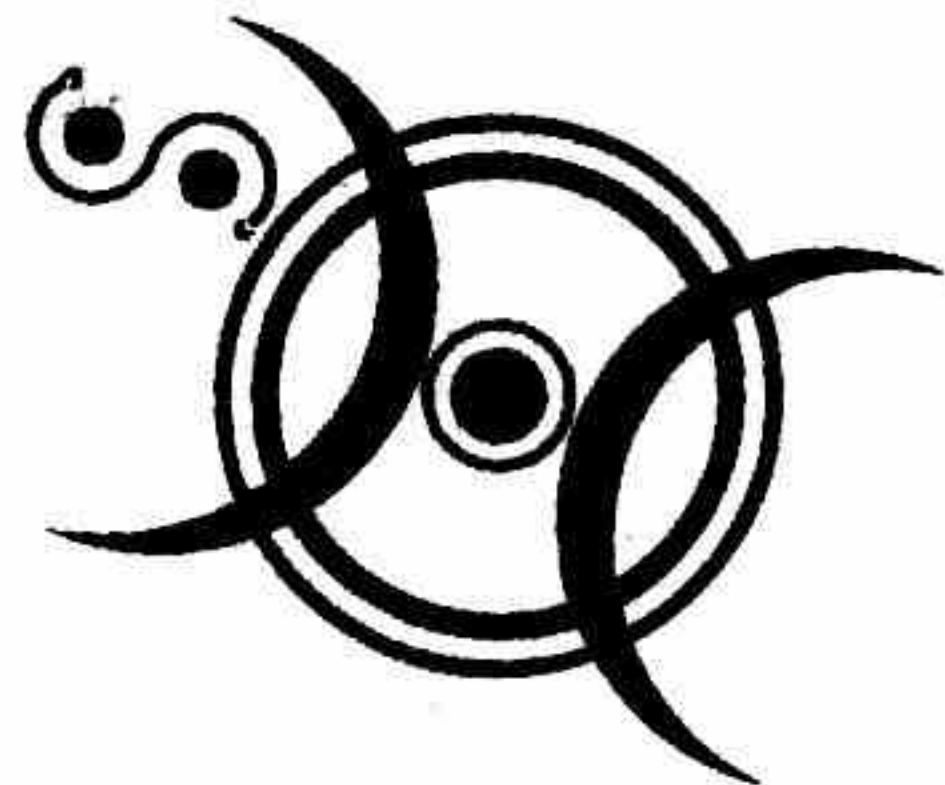
Глава 43.	Постановка задачи
Глава 44.	Разработка основной части сайта
Глава 45.	Панель управления WordPress



О. И. Д. А. 9 . II

С. И. Д. А. 9 . II

С. И. Д. А. 9 . II



ГЛАВА 43

Постановка задачи

43.1. Концепция проекта

У всех нас есть свои цели: кто-то хочет сдать экзамен, кто-то — бросить курить, кто-то копит на машину или квартиру, а кому-то нужно выучить английский. К сожалению, мы часто забываем о своих целях и начинаем двигаться в другом направлении. Чтобы человек не забывал о поставленных целях, и был создан проект ВЯблоко (www.vyabloko.ru), рассматриваемый в этом разделе книги вместо системы продажи недвижимости, которой он был посвящен в 3-м издании.

ПРИМЕЧАНИЕ

В каталоге *Edition_3* сопровождающего книгу электронного архива (см. приложение 4) вы найдете файлы глав 3-го издания книги с описанием системы продажи недвижимости.

Идея проекта принадлежит моему другу, а я, как программист, помогал ее реализовать. В этой главе мы рассмотрим только саму концепцию проекта, а в следующих главах раздела его код, но, понятное дело, не весь, а только его основные моменты. Проект весьма масштабный и рассмотреть весь его код в одной книге просто физически невозможно (ну, или нужно написать отдельную книгу об этом проекте, но не знаю, кто ее будет покупать).

Проект ВЯблоко даже на данном этапе находится на стадии тестирования и постоянно развивается, поэтому его реальный код будет немного отличаться от представленного здесь.

Разберемся, в чем заключается концепция проекта. ВЯблоко — это не обычная "напоминалка", которая станет слать e-mail'ы до тех пор, пока человек не достигнет поставленной цели. Нет, проект работает иначе. При регистрации пользователь указывает свою цель, выбирает дату ее достижения и указывает друга, который и будет напоминать пользователю о его цели (рис. 43.1)! Уведомления о цели будут отправляться на e-mail друга, указанный при регистрации, а уже он станет всеми возможными способами напоминать другу о его цели. Согласитесь, это гораздо эффективнее, чем простая "напоминалка". Когда звонит друг и спрашивает, не за-

курили ли вы снова, это намного лучше. Друг же и будет уведомлять систему о вашем продвижении и о том, что цель выполнена или, наоборот, провалена.

Конечно, на плечи друга возложено много работы, но ведь для этого и существуют друзья?

Рис. 43.1. Фрагмент главной страницы ВЯблоко (www.vyabloko.ru)

43.2. Алгоритм работы

Рассмотрим подробнее алгоритм работы проекта, начиная с добавления цели. Не советую пропускать этот раздел, иначе следующая глава, в которой будет приведен код проекта, окажется совсем непонятна.

После регистрации на сайте человек добавляет цель. При этом выбирается категория цели, есть и возможность включения переключателя **радикальная цель**. Категория нужна сугубо для администрирования целей, чтобы в будущем можно было добавить к сайту каталог целей.

Цели бывают обычными, которые достигаются небольшими шагами, и радикальными. Например, вы поставили перед собой цель изучить иностранный язык. Вы учите его, и каждый день ваши познания совершенствуются, и так до тех пор, пока вы не выучите язык до приемлемого уровня.

Представим, что вы хотите выучить язык за один месяц, и вы попросили проект напоминать о вашей цели раз в неделю. Это означает, что раз в неделю ваш друг будет получать напоминание о вашей цели и о вашем прогрессе в ее достижении. Всего ваш друг получит четыре уведомления, в каждом из которых система спросит друга, насколько успешно вы продвигаетесь к цели. Друг, общаясь с вами, знает, учите ли вы иностранный язык или, наоборот, забросили обучение. Он и должен сообщить системе, каковы ваши успехи в продвижении к цели. Поскольку уведомлений четыре, то каждый положительный ответ будет добавлять к этому продвижению 25 %. Если друг проигнорировал сообщение системы или сообщил, что вы не достигаете цели, индикатор продвижения увеличиваться не будет.

Допустим, три раза друг сообщил системе, что вы достигаете цели, а один раз проигнорировал это сообщение (или же вы забросили обучение) — тогда общий про-

гресс будет равен 75 %. Сообщить системе, что человек достиг цели, может или сам "владелец" цели или друг, положительно ответив на все уведомления системы. Да, вы сами можете сообщить, что достигли цели, — это делается на случай, если ваш друг проигнорировал некоторые уведомления системы.

Понятно, что "владелец" цели может солгать и сообщить о достижении цели, хотя на самом деле это не так. На этот случай страница просмотра цели снабжена комментариями, с помощью которых другие пользователи (в том числе и ваш друг) смогут сообщить, что вы не достигли цели.

Думаю, алгоритм понятен. А теперь представим, что вы поставили перед собой цель бросить курить. Это радикальная цель, ее нельзя достигать небольшими шагами, тут все просто: вы либо курите, либо — нет. Следовательно, если друг хотя бы раз сообщит, что вы не достигаете цели, цель считается проваленной.

Наш проект требует хостинга с поддержкой демона расписаний `crond` (или аналогичного), поскольку без него нельзя реализовать функцию напоминаний. Рассылка ежедневных, еженедельных, ежемесячных уведомлений осуществляется отдельными сценариями.

43.3. Структура базы данных

Изначально для работы проекта были нужны три таблицы:

- `users` — таблица, содержащая сведения о регистрации пользователей;
- `targets` — цели;
- `comments` — комментарии целей.

Все бы хорошо, но нам не хотелось разрабатывать панель управления, поэтому было решено интегрировать проект с WordPress. Во-первых, мы получали удобную панель управления. А во-вторых, — возможность использования всех плагинов, доступных для WordPress, в том числе и плагинов защиты от спама.

Но поскольку интеграцию с WordPress пришлось произвести уже после того, как сам проект был практически готов, для интеграции с базой данных WordPress потребовалось изменить его исходный код, но это уже дело техники. Что нужно было сделать, вы узнаете в следующей главе.

Мы параллельно установили WordPress, а потом удалили все, что касается пользовательской части сайта. Пользовательскую часть популярного блога заменил наш движок, а вот панель управления осталась без изменений.

ПРИМЕЧАНИЕ

Напомню, что познакомиться с платформой WordPress, зарегистрироваться и начать с ней работать можно на ее сайте <http://ru.wordpress.com>. Описание панели управления WordPress приведено в главе 45.

Информацию о пользователях WordPress хранит в таблице `wp_users`. Мы тоже возьмем эту таблицу без каких-либо изменений. Структура таблицы `wp_users` представлена в листинге 43.1.

Листинг 43.1. Структура таблицы wp_users

```

CREATE TABLE IF NOT EXISTS `wp_users` (
  `ID` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `user_login` varchar(60) NOT NULL DEFAULT '',
  `user_pass` varchar(64) NOT NULL DEFAULT '',
  `user_nicename` varchar(50) NOT NULL DEFAULT '',
  `user_email` varchar(100) NOT NULL DEFAULT '',
  `user_url` varchar(100) NOT NULL DEFAULT '',
  `user_registered` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  `user_activation_key` varchar(60) NOT NULL DEFAULT '',
  `user_status` int(11) NOT NULL DEFAULT '0',
  `display_name` varchar(250) NOT NULL DEFAULT '',
  PRIMARY KEY (`ID`),
  KEY `user_login_key` (`user_login`),
  KEY `user_nicename` (`user_nicename`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ;

```

Аутентификация пользователя будет производиться средствами WordPress, поэтому заботиться об обработке пользователей не придется. Нам лишь нужно знать назначение четырех полей: `user_login` (логин пользователя), `user_pass` (пароль), `user_email` (электронный адрес) и `display_name` (отображаемое имя).

Цели будут храниться в таблице `wp_posts`. Да, для WordPress мы приравняли цель и пост. Однако стандартная таблица `wp_posts` нам немного не подходила, поэтому мы ее модифицировали, добавив дополнительные поля (листинг 43.2).

Листинг 43.2. Таблица wp_posts

```

CREATE TABLE IF NOT EXISTS `wp_posts` (
/* стандартные поля WordPress */
/* ID цели */
  `ID` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
/* автор цели */
  `post_author` bigint(20) unsigned NOT NULL DEFAULT '0',
/* стандартное поле, дата поста */
  `post_date` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  `post_date_gmt` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
/* текст цели */
  `post_content` longtext NOT NULL,
/* заголовок, можно не заполнять */
  `post_title` text NOT NULL,
/* остальные поля нас не интересуют */
  `post_excerpt` text NOT NULL,
  `post_status` varchar(20) NOT NULL DEFAULT 'publish',
  `comment_status` varchar(20) NOT NULL DEFAULT 'open',
  `ping_status` varchar(20) NOT NULL DEFAULT 'open',

```



```
`post_password` varchar(20) NOT NULL DEFAULT '',
`post_name` varchar(200) NOT NULL DEFAULT '',
`to_ping` text NOT NULL,
`pinged` text NOT NULL,
`post_modified` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
`post_modified_gmt` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
`post_content_filtered` text NOT NULL,
`post_parent` bigint(20) unsigned NOT NULL DEFAULT '0',
`guid` varchar(255) NOT NULL DEFAULT '',
`menu_order` int(11) NOT NULL DEFAULT '0',
`post_type` varchar(20) NOT NULL DEFAULT 'post',
`post_mime_type` varchar(100) NOT NULL DEFAULT '',
/* счетчик комментариев */
`comment_count` bigint(20) NOT NULL DEFAULT '0',
/* категория цели, категория хранится в wp_terms */
`cid` int(11) DEFAULT '1',
/* электронный адрес пользователя, от этого поля можно отказаться,
  оно было в первой версии проекта в таблице targets, потом я оставил
  его для совместимости */
`email` varchar(250),
/* день достижения цели */
`day` int(11) DEFAULT '1',
/* месяц достижения цели */
`month` int(11) DEFAULT '1',
/* год достижения цели */
`year` int(11) DEFAULT '2012',
/* email-друга */
`friend` varchar(250),
/* имя друга */
`friendname` varchar(250),
/* пол друга, нужен для корректной отправки уведомлений */
`friendsex` int(11) DEFAULT '1',
/* интервал уведомлений о цели */
`inter` int(11) DEFAULT '1',
/* признак радикальной цели, 1 — цель радикальная */
`radical` int(11) DEFAULT '0',
/* количество положительных отметок о достижении цели */
`positive` int(11) DEFAULT '0',
/* количество отрицательных отметок о достижении цели */
`negative` int(11) DEFAULT '0',
/* если 1 — цель достигнута */
`isdone` int(11) DEFAULT '0',
/* используется в служебных целях */
`h` varchar(128) DEFAULT NULL,
/* дата постановки цели в формате int-timestamp */
`dt` int(11) DEFAULT NULL,
```



```
PRIMARY KEY (`ID`),  
KEY `post_name` (`post_name`),  
KEY `type_status_date` (`post_type`, `post_status`, `post_date`, `ID`),  
KEY `post_parent` (`post_parent`),  
KEY `post_author` (`post_author`)  
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ;
```

Комментарии хранятся в таблице `wp_comments`, ее структура неизменна — в ней мы и будем хранить комментарии. За добавление комментариев тоже отвечает WordPress, но, чтобы вызвать форму добавления комментария, придется немного потрудиться (об этом — в следующей главе).

Категории целей будут храниться в таблице `wp_terms`. В ней есть два основных поля: `term_id` и `name`. Первое хранит ID категории, второе — ее название (листинг 43.3).

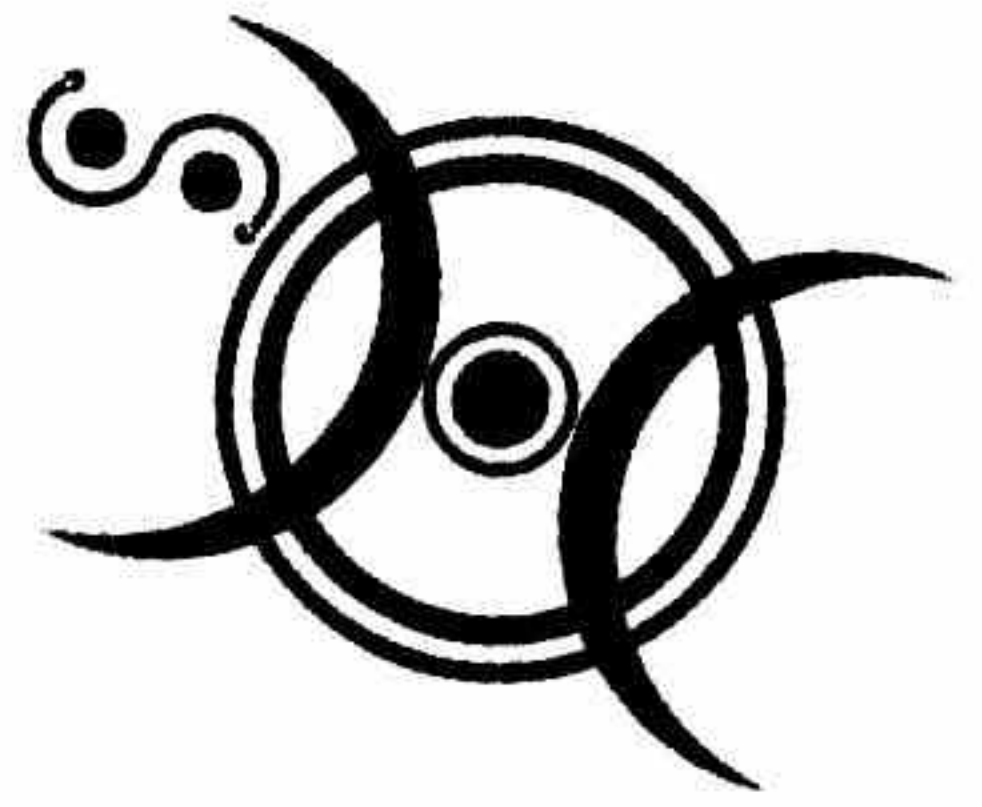
Листинг 43.3. Структура таблицы `wp_terms`

```
CREATE TABLE IF NOT EXISTS `wp_terms` (  
  `term_id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,  
  `name` varchar(200) NOT NULL DEFAULT '',  
  `slug` varchar(200) NOT NULL DEFAULT '',  
  `term_group` bigint(10) NOT NULL DEFAULT '0',  
  PRIMARY KEY (`term_id`),  
  UNIQUE KEY `slug` (`slug`),  
  KEY `name` (`name`)  
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ;
```

ПРИМЕЧАНИЕ

Файлы всех приведенных здесь листингов вы можете найти в каталоге *Glava_43* сопровождающего книгу электронного архива (см. приложение 4).

Итак, разобравшись с концепцией проекта и структурой таблиц, можно приступать к написанию кода проекта.



ГЛАВА 44

Разработка основной части сайта

44.1. Структура проекта

Проект хоть и сложный, но чтобы не усложнять его до бесконечности, практически весь функционал проекта сосредоточен в файле `index.php`. В табл. 44.1 приводится описание основных файлов проекта (за исключением дизайна).

Таблица 44.1. Структура проекта

Файл/каталог	Описание
<code>config.php</code>	Параметры доступа к БД и другие настройки
<code>index.php</code>	Основной сценарий проекта
<code>library.php</code>	Содержит некоторые вспомогательные функции
<code>no.php</code>	Сценарий, делающий "отрицательную" отметку (пользователь не достигает цели)
<code>refusal.php</code>	Друг отказался помочь пользователю в достижении цели (отказ приема приглашения)
<code>v.php</code>	Просмотр цели
<code>yes.php</code>	Сценарий, делающий "положительную" отметку (пользователь достигает цели)
каталог <code>mailers</code>	В этом каталоге хранятся сценарии отправки уведомлений
каталог <code>messages</code>	Здесь содержатся шаблоны уведомлений, которые будут отправлены сценариями из каталога <code>mailers</code>

ПРИМЕЧАНИЕ

Файлы всех приведенных в этой главе листингов вы можете найти в каталоге *Glava_44* сопровождающего книгу электронного архива (см. приложение 4).

Но сначала мы рассмотрим сценарий `config.php` (листинг 44.1), а уже затем — сценарий `index.php`. Остальные файлы будут рассмотрены по мере необходимости.

Листинг 44.1. Сценарий config.php

```
<?php
// параметры сервера БД
$server = 'localhost';
$db = 'database';
$user = 'user';
$pass = 'password';
?>
```

44.2. Основной сценарий index.php

В основном сценарии содержится 606 строк, и можно было бы, конечно, привести весь сценарий в одном большом листинге, прокомментировав каждую строку. Но такой листинг окажется сложен для восприятия, поэтому мы его разобьем на части и рассмотрим каждую часть отдельно.

44.2.1. Подключение механизмов WordPress

Начнем с самого начала, а именно с подключения механизмов WordPress (ru.wordpress.com) и подключения к базе данных (листинг 44.2).

Листинг 44.2. Фрагмент index.php (начало)

```
<?php

// мы не используем темы WordPress
define('WP_USE_THEMES', false);
// подключаем WordPress
include('./wp-blog-header.php');

// запускаем новую сессию
session_name('vyabloko');
session_start();

// получаем переменную z, в ней хранится имя
// страницы, которую нужно отобразить
$z = $_REQUEST['z'];

// подключаем настройки и библиотеку функций
include "config.php";
include "library.php";

// выводим заголовок (обычный HTML-файл) страницы
echo join('', file('header.html'));

// получаем служебные переменные
$current_user = wp_get_current_user();
```



```

$username = $current_user->display_name;
$userID = $current_user->ID;
$email = $current_user->user_email;

// подключаемся к серверу БД
mysql_connect($server, $user, $pass);
mysql_select_db($db);

```

44.2.2. Аутентификация пользователя

Заботиться об аутентификации пользователя нам не требуется — об этом позаботится сам WordPress, надо только в заголовок страницы добавить ссылки на следующие сценарии:

- ❑ `wp-login.php` — отображает форму входа на сайт (рис. 44.1);
- ❑ `wp-login.php?action=register` — отображает форму регистрации нового пользователя (рис. 44.2).

Рис. 44.1. Стандартная форма входа на сайт

Рис. 44.2. Форма регистрации на сайте

В самом сценарии нам нужно удостовериться, что пользователь аутентифицирован перед выполнением каких-либо действий. Это можно сделать с помощью функции `is_user_logged_in()`. Например, перед выводом меню действий мы проверяем, аутентифицирован ли пользователь (листинг 44.3).

Листинг 44.3. Проверка аутентификации пользователя

```

if ( is_user_logged_in() ) {

echo "
<a href=\"index.php\">Главная</a>
<a href=\"wp-admin/profile.php\">Профиль</a>

```



```

<a href=\"index.php?z=my\">Мои цели</a>
|
Привет, $username
|
<a href=\"wp-login.php?action=logout\">Выход</a>";
}

```

Вообще-то HTML-код отделен от PHP-кода и хранится в TPL-шаблонах, но для большей простоты здесь и далее будем считать, что сценарий выводит HTML-код непосредственно в браузер.

Как видите, в меню есть всего две ссылки, которые обрабатываются нашим сценарием: ссылка на главную страницу и ссылка на страницу управления целями пользователя.

44.2.3. Форма добавления цели

Пока все просто и как-то особо не отличается от любого другого обычного сайта. Разве что использование стандартных функций WordPress чуть необычно. Самое время продолжить углубление в наш сценарий. Начнем с формы добавления цели.

Посмотрим еще раз, как она выглядит (рис. 44.3, а). Форма подразумевает выбор категории цели (рис. 44.3, б), а также предполагаемой даты достижения цели (рис. 44.3, в).

Рис. 44.3, а. Форма добавления цели в состоянии "спокойствия"

Рис. 44.3, б. Выбор категории

Рис. 44.3, в. Выбор даты

Реализовать выбор даты проще всего. Для этого можно взять любой готовый календарь. В этом проекте использовался jQuery UI Datepicker (<http://jqueryui.com/demos/datepicker/>). Чуть позже мы поговорим, в каком формате должен передавать дату календарь.

Список категорий целей мы получаем с помощью стандартных функций WordPress `get_query_var()` и `get_categories()` — листинг 44.4.

Листинг 44.4. Код вывода категорий целей

```
echo ' <fieldset>
    <label for="cid">Тематика цели:</label>
    <select name="cid" id="cid"> ';
$cat = get_query_var('cat');
$categories=get_categories('orderby=id&order=ASC&hide_empty=0&child_of='.$cat);
if ($categories) {
    foreach($categories as $term) {
        echo '<option value="' . $term->term_id . '">' . $term->cat_name . ' - '
        . $term->description . '</option>';
    }
}
echo ' </select>
    </fieldset> ';
```

С остальными полями формы вы справитесь без особых проблем — там остались только текстовые поля и поля типа `checkbox`.

44.2.4. Добавление цели

Формат добавления цели передает информацию сценарию `index.php`, вызванному с параметром `z=set`:

```
<form method="post" action="index.php" name="aim">
<input type="hidden" name="z" value="set">
```


Обработкой добавления цели занимается следующий огромный фрагмент кода (листинг 44.5):

Листинг 44.5. Добавление цели

```
$z = $_REQUEST['z'];
...
if ($z=="set") {
// если пользователь аутентифицирован...
if ( is_user_logged_in() ) {

// получаем категорию цели
$cid = $_POST[cid];
// текст цели
$target = $_POST[target];
// электронный адрес друга
$friend = $_POST[friend];
// имя друга
$friendname = $_POST[friendname];

/* с помощью фильтра проверяем корректность электронного
адреса. Для упрощения кода мы просто прерываем работу сценария,
если введен некорректный адрес */
if (!filter_var($friend, FILTER_VALIDATE_EMAIL))
    die('Некорректный e-mail помощника!');

// получаем день, месяц и год достижения цели
// эти данные передаются нашим календарем
$day = $_POST[day]; $month = $_POST[month]; $year = $_POST[year];

/*          Обработка ошибок          */

// если переменная $cid не является числовой, прекращаем
// работу сценария
if(!is_numeric($cid)) die("Попытка взлома? $cid");

// если не указано явно, что цель радикальная, тогда
// цель считается обычной
if (isset($_POST[radical])) $radical = 1; else $radical = 0;

// переменная
$interval = $_POST[interval];

// получаем текущую дату в виде строки и timestamp
$dt = date('Y-m-d h:i:s');
$timestamp = time();

// статус записи, нужно для WordPress
$pub = 'publish';
```



```
// получаем пол друга, желательно также проверить
// эту переменную на наличие SQL Injection, но для простоты
// кода обходимся без такой проверки
$sex = $_POST[sex];

// заголовок, нужен для WordPress
$title = strtolower(ctl_sanitize_title($target));

// получаем ID последней записи таблицы wp_posts
$last = $wpdb->get_row("SHOW TABLE STATUS LIKE 'wp_posts'");
$lastid = $last->Auto_increment;

// формируем длинный запрос, согласно структуре таблицы
// wp_posts (см. главу 43)
$q = "insert into wp_posts (ID, post_author, post_date, post_date_gmt,
post_content, post_title, post_excerpt, post_status, comment_status,
ping_status, post_password, post_name, to_ping, pinged, post_modified,
post_modified_gmt, post_content_filtered, post_parent, guid, menu_order,
post_type, post_mime_type, comment_count, cid, email, day, month, year, friend,
friendname, friendsex, inter, radical, positive, negative, isdone, h, dt)
VALUES
(0, $userID, '$dt', '$dt', '$target', '$target', '', '$pub', 'open', 'open',
'', '$title', '', '', '2011-02-10 08:18:41', '2011-02-10 08:18:41', '', 0,
'http://vyabloko.ru/?p=$lastid.html', 0, 'post', '', 0, '$cid', '$email',
'$day', '$month', '$year', '$friend', '$friendname', '$sex', '$interval',
$radical, 0, 0, 0, 0, $timestamp)";

// выполняем запрос
mysql_query($q) or die(mysql_error());

// формируем запрос для таблицы wp_term_relationships (соответствие
// цели категории)
$q2 = "INSERT INTO wp_term_relationships (object_id,term_taxonomy_id) VALUES
('$lastid','$cid')";

// выполняем второй запрос
mysql_query($q2) or die(mysql_error());

// отправляем электронное письмо помощнику
$last = mysql_insert_id();
$n = getusername($_SESSION[session_email]);
$d = "$day / $month / $year";
$goal = "http://www.vyabloko.ru/v.php?t=$last";
$refusal = "http://www.vyabloko.ru/refusal.php?t=$last";

$headers = 'MIME-Version: 1.0' . "\r\n";
$headers .= 'Content-type: text/html; charset=utf-8' . "\r\n";
$headers .= "From: ВЯблоко <admin@vyabloko.ru>" . "\r\n";
```



```

$msg = join('', file('messages/to_helper.msg'));
if ($sex==1) $s = "Уважаемый"; else $s="Уважаемая";
$msg = str_replace('{hello}', $s, $msg);
$msg = str_replace('{Name}', $n, $msg);
$msg = str_replace('{helper-name}', $friendname, $msg);
$msg = str_replace('{date}', $d, $msg);
$msg = str_replace('{goallink}', $goal, $msg);
$msg = str_replace('{no}', $refusal, $msg);
$msg = str_replace('{goal}', $target, $msg);

if (!mail($friend, "Ваш друг просит помочь ему в достижении цели", $msg,
$headers))
    echo "<p>Ошибка при отправке e-mail вашему помощнику. Скорее всего, он не
получил письмо, поэтому свяжитесь с ним другим способом или
    проверьте его e-mail (возможно, вы допустили ошибку) и в разделе Мои цели
выберите функцию Изменить помощника: снова укажите этого же
    человека, но введите правильный e-mail";

echo "<p><p>Ваша цель успешно добавлена! Просмотреть цель и изменить помощника
можно в разделе Мои цели";
}
else
    echo "Цель не добавлена! <p>Войдите или <a
href=register.php>зарегистрируйтесь</a>!";

} // if

```

Рассмотрим этот фрагмент кода подробнее.

Сначала мы проверяем, аутентифицирован ли пользователь. Если нет, сразу выводим сообщение о необходимости входа на сайт.

Далее мы получаем данные от формы цели и устраиваем их минимальную проверку. Корректность e-mail мы проверяем с помощью фильтра, а не регулярных выражений.

Затем мы формируем огромный запрос (в нем нет ничего сложного, это обычный INSERT-запрос) и выполняем его.

После добавления цели надо отправить письмо помощнику. Для этого нужно сначала сформировать письмо. Шаблон письма хранится в файле `messages/to_helper.msg` (листинг 44.6).

Листинг 44.6. Файл `messages/to_helper.msg`

{hello} {helper-name}, ваш друг {Name} просит вас помочь ему достичь цель: {goal}.

Он собирается выполнять это обещание до {date}.

Если вы согласны ему в этом помочь, перейдите по ссылке:

{goallink}

Если вы не согласны, перейдите по этой ссылке:

```
{no}
```

С уважением,
команда ВЯблоко.Ру

Письмо формируется путем замены переменных шаблона:

```
$msg = join('', file('messages/to_helper.msg'));
if ($sex==1) $s = "Уважаемый"; else $s="Уважаемая";
$msg = str_replace('{hello}', $s, $msg);    // приветствие
$msg = str_replace('{Name}', $n, $msg);     // имя
// имя помощника
$msg = str_replace('{helper-name}', $friendname, $msg);
// дата достижения цели
$msg = str_replace('{date}', $d, $msg);
// ссылка на цель
$msg = str_replace('{goallink}', $goal, $msg);
// ссылка для отказа в участии
$msg = str_replace('{no}', $refusal, $msg);
// сама цель
$msg = str_replace('{goal}', $target, $msg);
```

Обратите внимание на заголовки письма:

```
$headers = 'MIME-Version: 1.0' . "\r\n";
$headers .= 'Content-type: text/html; charset=utf-8' . "\r\n";
$headers .= "From: ВЯблоко <admin@vyabloko.ru>" . "\r\n";
```

Осталось отправить письмо. Мы это делаем с помощью стандартной функции mail():

```
if (!mail($friend, "Ваш друг просит помочь ему в достижении цели", $msg,
$headers))
    echo "<p>Ошибка при отправке e-mail вашему помощнику. Скорее всего, он не
получил письмо, поэтому свяжитесь с ним другим способом или
проверьте его e-mail (возможно, вы допустили ошибку) и в разделе Мои цели
выберите функцию Изменить помощника: снова укажите этого же
человека, но введите правильный e-mail";
```

```
echo "<p><p>Ваша цель успешно добавлена! Просмотреть цель и изменить помощника
можно в разделе Мои цели";
```

44.2.5. Просмотр всех целей

Пользователь может просмотреть список своих целей. Вывод списка целей осуществляется кодом из листинга 44.7. Как обычно, код упрощен (вместо использования шаблонизатора просто выводим HTML-код в браузер).

Листинг 44.7. Вывод целей пользователя

```
if ($z === "my") {

    echo "<h1>Мои цели</h1>";

    // если пользователь не аутентифицирован, мы просто
    // завершаем работу сценария, т. к. ссылку на вывод
    // целей можно получить только после входа на сайт, а раз
    // сценарий был вызван непосредственно (без входа на сайт),
    // можно просто прервать его работу без вывода красивого
    // сообщения
    if ( !is_user_logged_in() ) {
        die("Вы не зарегистрированы на сайте. Войдите или
зарегистрируйтесь<p>"); }

    // выбираем все цели, автор которых — зарегистрированный
    // в данный момент пользователь
    $q = "select * from wp_posts where post_author=$userID";
    $r = mysql_query($q) or die(mysql_error());
    // если есть цели
    if (mysql_num_rows($r)>0) {
    // то выводим заголовок таблицы
    echo "<table><tr>
        <td>Цель</td>
        <td>Дата остановки</td>
        <td>Помощник</td>
        <td>Дата завершения</td>
        <td>Достигнута?</td>
        <td>+</td>
        <td>-</td>
        <td>%</td>
        <td>Действия</td>";

    // выводим сами цели:
    while ($row = mysql_fetch_array($r)) {

        if ($row[isdone]==1) $done = "Ура! Человек слова!";
        else $done = "Пока нет";

        $dt = $row[post_date];

        if ($row[isdone]==0) {
            $progress = 0;
            // вычисляем прогресс в %
            $positive = $row[positive];
            $negative = $row[negative];
```



```

// нужно вычислить, сколько попыток контроля
// должно быть до часа X (времени достижения цели)
if ($row[inter] == 1) {
// попыток ровно столько, сколько дней от даты
// постановки до даты достижения
    $start = $row[post_date];
    $end = mktime(0, 0, 0, $row[month], $row[day], $row[year]);
    $raznica = $end - $start;
    // в сутках 24 часа * 3600 = 86400
    $qty = $raznica / 86400;
    // считаем только положительные попытки,
    // отрицательные отнимаются от положительных
    $positive = $positive - $negative;
    // $qty = 100%, $positive = X%
    $progress = ($positive * 100) / $qty;
}
} else $progress = 100;

$end = mktime(0, 0, 0, $row[month], $row[day], $row[year]);
// разница между временем отрицательна, значит время вышло
if ($row[isdone]==0) if ($end - time() < 0) { $done = "Провалена";
$progress = 0; };

echo "<tr>
<td>$row[post_content]</td>
<td>$dt</td>
<td>$row[friendname]</td>
<td>$row[day] / $row[month] / $row[year]</td>
<td>$done</td>
<td>$row[positive]</td>
<td>$row[negative]</td>
<td>" . number_format($progress, 2, ',', '') . "</td>
<td><a href=index.php?z=change&t=$row[ID]>Сменить помощника</a>
<a href=v.php?t=$row[ID]>Подробнее</a>
<a href=index.php?z=done&t=$row[ID]>Достигнута!</a>
</td></tr>";

}
echo "</table>";
}
else echo "<p>У вас пока нет целей";
}

```

При выводе целей учитывается:

- ☐ выполнена цель или нет (поле isdone);
- ☐ процент достижения цели;
- ☐ провалена цель или нет (если время достижения цели уже вышло).

Процент достижения цели вычисляется довольно просто. Представим, что вы желаете достичь цели за 10 дней. Интервал напоминания о цели — каждый день. Следовательно, должно быть сделано 10 напоминаний.

Итак, прошло четыре дня, друг за это время поставил три положительные отметки (сообщив о том, что вы достигаете цели) и одну отрицательную. Общий прогресс достижения цели в этом случае окажется равным всего 20 %, т. к. одна попытка равна 10 % (в нашем случае), а количество отрицательных отметок вычитается из положительных.

Если же в течение четырех дней друг/помощник поставил три положительных отметки, а одну вообще пропустил, то общий прогресс будет равен 30 %.

В меню действий есть три действия:

- ☐ **Сменить помощника** — позволяет назначить другого помощника;
- ☐ **Подробнее** — просмотр подробной информации о цели;
- ☐ **Достигнута** — цель отмечается как достигнутая самим пользователем.

Изменение помощника — предельно простое действие, которое вы можете реализовать самостоятельно. Отображается форма, позволяющая ввести имя и электронный адрес помощника. Далее помощнику, как и при постановке цели, отправляется письмо с просьбой принять участие в проекте.

Когда пользователь отмечает цель как достигнутую, выполняется код, указанный в листинге 44.8.

Листинг 44.8. Отмечаем цель как достигнутую

```
if ( !is_user_logged_in() ) {
    die("Вы не зарегистрированы на сайте. Войдите или зарегистрируйтесь"); }

if (!isset($_GET[t])) die('Цель не выбрана');

$target = $_GET[t];
$r = mysql_query("select * from wp_posts where ID=$target limit 1");

if (mysql_num_rows($r)>0) {

    $row = mysql_fetch_array($r);
    if ($userID == $row[post_author]) {

        mysql_query("update wp_posts set isdone=1 where ID=$target limit 1");
        echo "<p>Ура! Вы достигли поставленной цели. Вы — человек слова!";

    }
    else echo "<p>Вы не можете достичь цель другого человека";

}
else echo "<p>Цели с заданным ID не существует";
```

Просмотр цели — это сценарий v.php, который будет рассмотрен далее.

44.3. Сценарий v.php — просмотр цели

Сценарий v.php отображает подробную информацию о цели. Сценарию передается номер цели, а он отображает саму цель, имя автора, дату достижения цели и имя помощника. Зная структуру таблицы wp_posts, вы напишете такой сценарий примерно минут за пять. В нем нет ничего сложного, если бы не одно но — для каждой цели нужно отображать комментарии пользователей и форму добавления комментариев.

Комментарии вывести очень просто. Для этого служит код, приведенный в листинге 44.9.

Листинг 44.9. Вывод комментариев (фрагмент v.php)

```
// выбираем все комментарии к цели, заданной переменной
// $target
$q = "select * from wp_comments where comment_post_ID=$target";
$r = mysql_query($q) or die(mysql_error());
if (mysql_num_rows($r)==0) echo "<p>Комментариев пока нет";
else while ($row = mysql_fetch_array($r)) {

// автор комментария
$username = $row[comment_author];

$dt = $row[comment_date];

echo "<div id=incom>
    <div id=comment>
    <div>$dt</div>
    <img src=img/ava_r.gif border=0><span>
    <p><a href=#>$username</a>
        <br>$row[comment_content]</p></span>
    </div>
</div>";
}
```

Вывести форму добавления комментариев позволяет код, приведенный в листинге 44.10. Если вы воспользуетесь им в своем проекте, обратите внимание, что вам нужно обязательно указать номер поста (записи), к которому относится добавляемый комментарий (то, что нужно изменить, в листинге выделено полужирным).

Листинг 44.10. Форма добавления комментариев WordPress

```
define('WP_USE_THEMES', false);
include('../wp-blog-header.php');

require( dirname(__FILE__) . '/wp-load.php' );
...
```



```

require('wp-includes/comment.php');
<?php if(comments_open()) : ?>
    <?php if(get_option('comment_registration') && !$user_ID) : ?>
        <p>Вам нужно <a href="<?php echo get_option('siteurl'); ?>/wp-
login.php?redirect_to=<?php echo urlencode(get_permalink()); ?>">войти</a> для
отправки комментария.</p><?php else : ?>
        <form action="<?php echo get_option('siteurl'); ?>/wp-comments-
post.php" method="post" id="commentform">
            <?php if($user_ID) : ?>
                <p>Вы вошли как <a href="<?php echo get_option('siteurl');
?>/wp-admin/profile.php"><?php echo $user_identity; ?></a>. <a href="<?php echo
get_option('siteurl'); ?>/wp-login.php?action=logout" title="Log out of this
account">Выйти &raquo;</a></p>
            <?php else : ?>
                <p><input type="text" name="author" id="author" value="<?php
echo $comment_author; ?>" size="22" tabindex="1" />
                <label for="author"><small>Имя <?php if($req) echo
"(обязательно)"; ?></small></label></p>
                <p><input type="text" name="email" id="email" value="<?php echo
$comment_author_email; ?>" size="22" tabindex="2" />
                <label for="email"><small>E-mail (не будет опубликовано) <?php
if($req) echo "(required)"; ?></small></label></p>
                <p><input type="text" name="url" id="url" value="<?php echo
$comment_author_url; ?>" size="22" tabindex="3" />
                <label for="url"><small>Сайт</small></label></p>
            <?php endif; ?>
            <p><textarea name="comment" id="comment" cols="100%" rows="10"
tabindex="4"></textarea></p>
            <p><input name="submit" type="submit" id="submit" tabindex="5"
value="Добавить комментарий" />
            <input type="hidden" name="comment_post_ID" value="<?php echo
$target; ?>" /></p>
            <?php do_action('comment_form', $post->ID); ?>
        </form>
    <?php endif; ?>
<?php else : ?>
    <p>Комментарии закрыты.</p>
<?php endif; ?>

```

44.4. Сценарии отправки уведомлений. Сценарии yes.php и no.php

В каталоге mailers содержатся сценарии отправки уведомлений:

- ☐ everyday.php — будет выполняться каждый день;
- ☐ everyweek.php — выполняется раз в неделю;
- ☐ everymonth.php — выполняется раз в месяц.

Все эти сценарии идентичны, а различаются только SQL-запросами. Теоретически можно было бы обойтись одним сценарием, но поскольку нам надо будет прописывать вызов этих сценариев в расписании `crond`, то намного проще реализовать их так, чтобы не пришлось передавать им аргументы.

Прежде чем рассмотреть код этого сценария, рассмотрим алгоритм его работы. Казалось бы, все просто — сценарий выбирает все цели с нужным интервалом и отправляет всем помощникам соответствующие сообщения. Шаблон сообщения хранится в папке `messages`. Так, в файле `messages/notify.msg` (листинг 44.11) хранится шаблон сообщения, которое будет отправлено помощнику.

Листинг 44.11. Шаблон письма-уведомления (сокращенный)

```
<p>{hello} {friendname}!

<p>Вы обещали вашему другу {name} помочь в достижении его цели {target}

<p>Достигает ли ваш друг цели:

<p><a href={yes-link}>ДА</a>

<p><a href={no-link}>НЕТ</a>

...

<p>Помогайте вашим друзьям в достижении их целей! Используйте для этого все
средства (телефонный звонок, личная встреча), а не только общение по e-mail!

<p>С уважением,
<p>Команда ВЯблоко.Ру
```

Итак, сценарий отправит сообщения каждому помощнику. В сообщении содержатся две ссылки: переход по `yes-link` означает, что пользователь достигает цели, а переход по `no-link` равносителен тому, что пользователь забыл о своей цели.

Но, что если помощник окажется чрезмерно усердным, т. е. решит помочь своему другу накрутить счетчики, несколько раз нажав ссылку `yes-link`? Для защиты от подобного поведения мы используем служебное поле `h` в таблице целей. Когда `crond` запускает сценарий, он генерирует уникальный хэш для каждой цели. Этот хэш добавляется к каждой ссылке `yes-link` и `no-link`, т. е. ссылки выглядят примерно так:

```
http://vyabloko.ru/yes.php?t=22&h=05c53e753f844126ee0b097cadf30a3a
http://vyabloko.ru/no.php?t=22&h=05c53e753f844126ee0b097cadf30a3a
```

Здесь аргумент `t` — задает номер цели, а `h` — тот самый хэш.

Хэш вычисляется с помощью встроенной SQL-функции `MD5` на основании номера цели и текущего времени (`timestamp`).

При запуске сценарии `yes.php` и `no.php` сравнивают значение параметра `h` с содержимым поля `h`. Если оно отличается, то отметка не учитывается. Если же значение переменной `h` равно значению поля `h`, отметка засчитывается, а значение поля `h` обнуляется.

Вот теперь можно рассмотреть сценарий `everyday.php`, отправляющий уведомление помощнику (листинг 44.12). Из-за проблем с отправкой сообщений в кодировке UTF-8 стандартной функцией `mail()` здесь мы используем класс `PHPMailer`, корректно выполняющий отправку сообщения. Почему-то в случае с `mail()` текст сообщения неправильно читался в некоторых почтовых клиентах (несмотря на правильность указания заголовков). Разбираться не было времени, поэтому для отправки сообщений был "прикручен" класс `PHPMailer`.

Листинг 44.12. Сценарий `everyday.php`

```
<?php
// для корректной работы с UTF-8
header ("Charset: utf-8");
include "../config.php";
include "../library.php";

mysql_connect($server, $user, $pass) or die('Error');
mysql_select_db($db);
mysql_query('SET NAMES utf8');

// первым делом генерируем хэши
// WHERE нужен, чтобы запрос не трогал "чужие" цели –
// установкой хэша для целей с другим временным интервалом
// пусть занимается другой сценарий
// хэш – это MD5(текущее время + номер цели)
$q = 'update wp_posts set h = MD5( UNIX_TIMESTAMP( ) + wp_posts.ID ) where
(inter=1) ';
mysql_query($q) or die(mysql_error());

// выбираем цели: интервал = 1 (каждый день) и со статусом 0
// (не достигнута)
$q = "select * from wp_posts where (inter=1) and (isdone=0)";

// нужно для отладки, переменная $DEBUG определяется в
// файле config.php
if ($DEBUG==1) echo $q . "<br>\n";

$r = mysql_query($q);

// подключаем PHPMailer
require_once 'mailer/class.phpmailer.php';
```



```
while ($row = mysql_fetch_array($r)) {

// вывод на английском, чтобы при проблемах с кодировкой
// видеть все, что выводит сценарий
if ($DEBUG==1) echo "<p>Sending message to $row[email]: ";

$number_days = ceil((mktime(0,0,0, $row[month], $row[day], $row[year]) -
time())/86400);

// пропускаем просроченные цели
if ($number_days<0) {
    echo " Цель просрочена, переход к следующей\n";
    continue;
}

// пропускаем цели с неправильно указанным адресом
// помощника. Теоретически таких не должно быть, но лучше
// перестраховаться
if (!filter_var($row[friend], FILTER_VALIDATE_EMAIL)) {
    echo "Invalid e-mail: $row[friend] SKIP";
    continue;
}

// используем PHPMailer
$mail = new PHPMailer(true);
// загружаем шаблон сообщения
$msg = join('',file('/full/path/messages/notify.msg'));
// заменяем переменные шаблона
if ($row[friendsex]==1) $s = "Уважаемый"; else $s="Уважаемая";
$msg = str_replace('{hello}', $s, $msg);
$msg = str_replace('{friendname}', $row[friendname], $msg);
$username = getusername($row[email]);
$msg = str_replace('{name}', $username, $msg);
$msg = str_replace('{target}', $row[post_content], $msg);
$dt = $row[day] . "/" . $row[month] . "/" . $row[year];
$msg = str_replace('{date}', $dt, $msg);

$yeslink = "http://vyabloko.ru/yes.php?t=$row[ID]&h=$row[h]";
$noink = "http://vyabloko.ru/no.php?t=$row[ID]&h=$row[h]";

$msg = str_replace('{yes-link}', $yeslink, $msg);
$msg = str_replace('{no-link}', $noink, $msg);

// заполняем поля сообщения и отправляем его
try {
    $mail->AddReplyTo('admin@vyabloko.ru', 'ВЯблоко');
    $mail->AddAddress($row[friend], $row[friendname]);
    $mail->SetFrom('admin@vyabloko.ru', 'ВЯблоко');
    $mail->AddReplyTo('admin@vyabloko.ru', 'ВЯблоко');
```



```

$mail->Subject = 'Мониторинг цели друга';
$mail->MsgHTML($msg);
$mail->Send();
echo "Message Sent OK\n";
} catch (phpmailerException $e) {
    echo $e->errorMessage();
} catch (Exception $e) {
    echo $e->getMessage();
}

unset($mail);
}

?>

```

Вроде бы все должно быть понятно...

ПРИМЕЧАНИЕ

К классу `PHPMailer` мы уже обращались в главе 31. Последняя на момент написания этих строк версия класса `PHPMailer` (файл `PHPMailer_v5.1.zip`) находится в каталоге `Glava_31` сопровождающего книгу электронного архива. В дальнейшем свежие версии этого класса вы всегда сможете скачать по адресу: <http://phpmailer.worxware.com/>.

Теперь реализуем сценарии `yes.php` и `no.php`. Сценарии предельно просты. Сценариям передаются два аргумента: номер цели и хэш. Далее сценарии выбирают цель с заданным номером и увеличивают ее поля `positive` (`yes.php`) или `negative` (`no.php`). В листинге 44.13 приводится сценарий `yes.php`. Сценарий `no.php` отличается только изменением поля `negative`.

Листинг 44.13. Сценарий `yes.php`

```

<?
// стандартное начало для нашего проекта
@session_name('apple');
@session_start();

define('WP_USE_THEMES', false);
include('./wp-blog-header.php');

include "config.php";
include "library.php";

mysql_connect($server, $user, $pass);
mysql_select_db($db);

// получаем номер цели и хэш
$t = $_GET[t];          // номер цели
$h = $_GET[h];          // хэш

```



```
echo join('', file('header.html'));

// получаем информацию о цели
$q = "select positive, h from wp_posts where ID = $t limit 1";
$r = mysql_query($q);
$row = mysql_fetch_array($r);
$positive = $row[positive] + 1;

// значение 0 — служебное, если человек его указал в качестве
// значения аргумента h, то он точно пытается "накрутить"
// счетчик
if ($h === "0") die();

if ($row[h] === $h) {

    $q = "update wp_posts set positive = $positive, h = 0 where ID = $t limit 1";
    mysql_query($q) or die(mysql_error());
    echo "<p>Спасибо за ваше участие в нашем проекте!";

} else
    echo "<p>Ссылка недействительна, ваш голос не учтен. Возможно, ссылка
просрочена — если вы не успели вовремя проголосовать. Срок действия ссылки
зависит от интервала оповещения цели — 1 день, 1 неделя, 2 недели, 1 месяц,
полгода, год";

if ( !is_user_logged_in() ) {

    echo "<p>Чтобы ваша работа с проектом была более эффективной, <a
href=http://vyabloko.ru/wp-login.php?action=register>зарегистрируйтесь на
сайте</a>.";

}

?>
```

Мы рассмотрели все основные сценарии проекта. Чтобы сценарии уведомления заработали как следует, нужно добавить в расписание планировщика задач сценарии рассылки уведомлений. Далее я покажу, как это сделать на примере панели управления DirectAdmin.

ПРИМЕЧАНИЕ

Познакомиться с DirectAdmin можно по ссылке: <http://www.directadmin.com/>. Работа с панелью управления DirectAdmin описана в главе 46.

Зайдите в панель управления и выберите команду **Планировщик задач** (рис. 44.4). Укажите периодичность запуска сценария (рис. 44.5). Например, если сценарий должен запускаться каждый день, то вам нужно изменить только поля **Минута** и **Час**, указав точное время запуска сценария, а все остальные поля не трогать (оставить значение *).

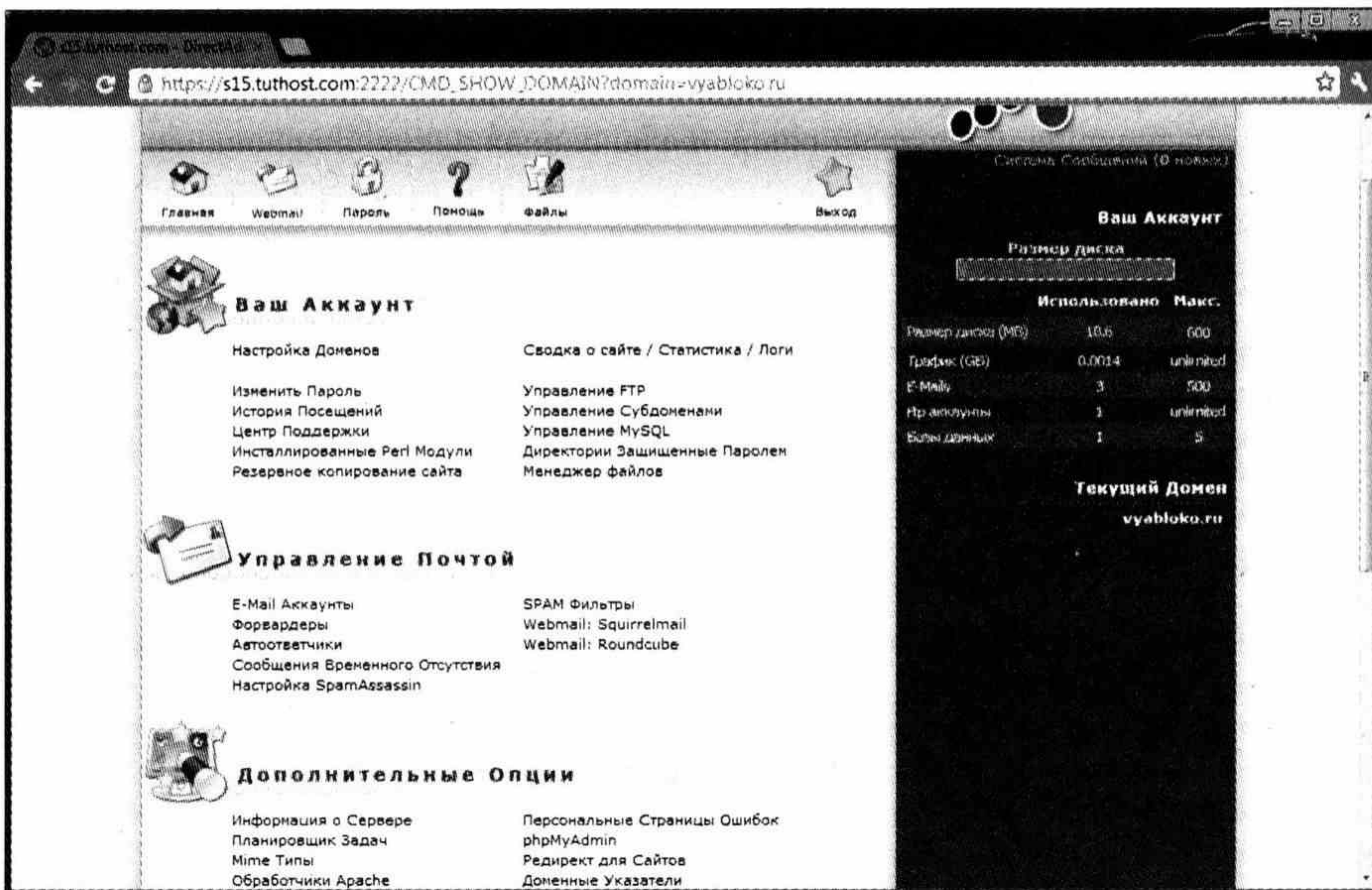


Рис. 44.4. Панель управления DirectAdmin

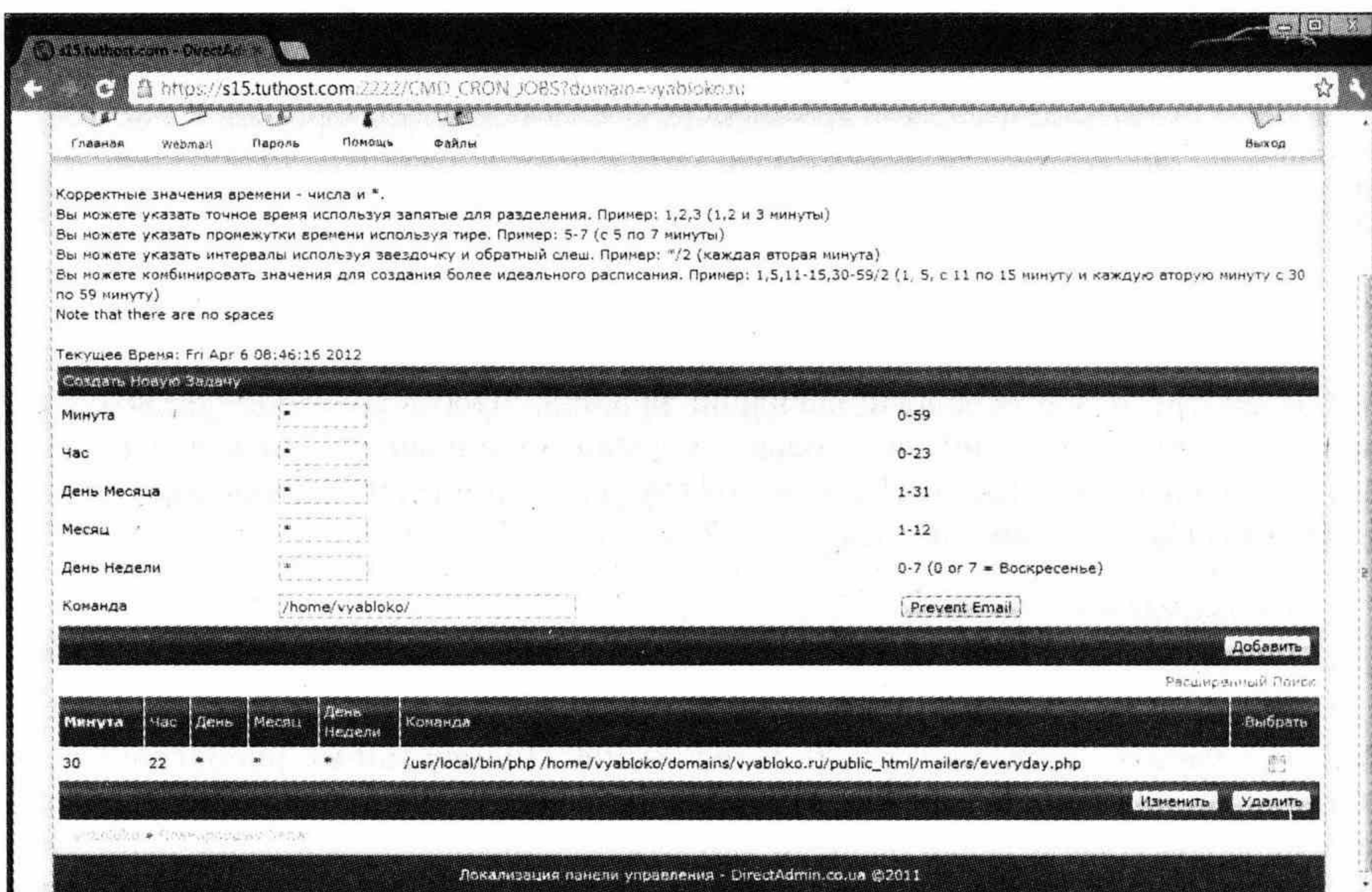


Рис. 44.5. Редактирование расписания планировщика задач

Чтобы сценарий запускался раз в неделю, заполните поля **Минута**, **Час** и **День недели**. Например, чтобы сценарий запускался в 8 утра каждое воскресенье, заполните поля так:

- ☐ **Минута** — 0;
- ☐ **Час** — 8;
- ☐ **День недели** — 7.

Для запуска сценария раз в месяц поля нужно заполнить так:

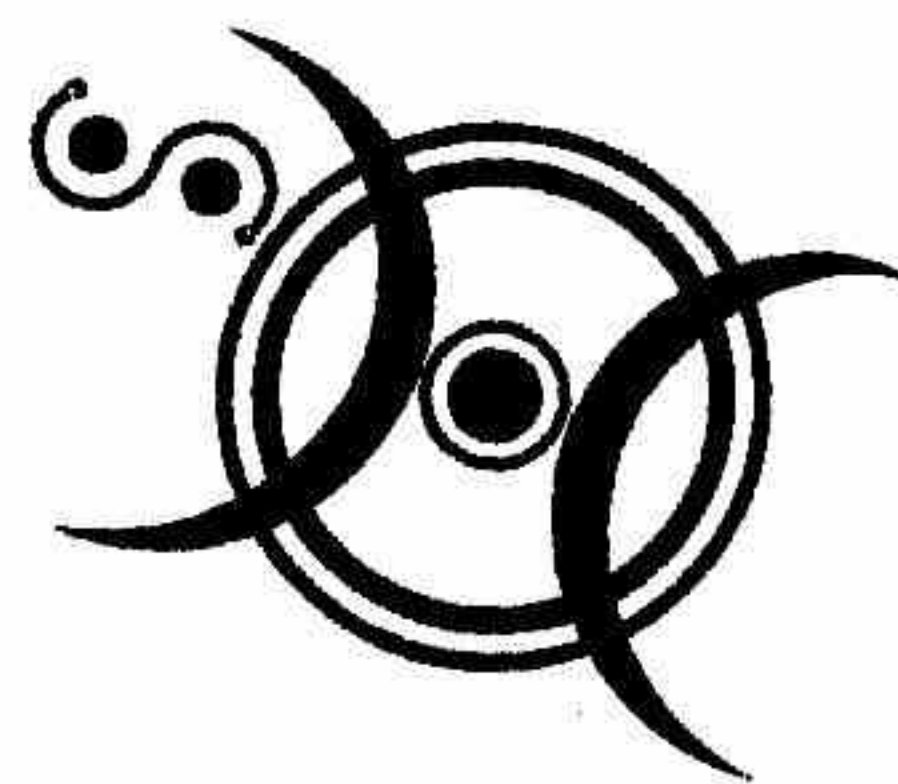
- ☐ **Минута** — 0;
- ☐ **Час** — 9;
- ☐ **День месяца** — 1.

В этом случае сценарий будет запускаться в 9:00 первого числа каждого месяца.

Перед указанием времени убедитесь, что ваше текущее время совпадает с временем сервера. Если это не так, при указании времени делайте поправку на разницу во времени.

Обратите внимание, как на рис. 44.5 прописана команда запуска сценария. Мы не можем просто так запустить PHP-сценарий — это обычный текстовый файл. Зато мы можем запустить интерпретатор `/usr/local/bin/php` и передать сценарий ему, то есть требуется указать полный путь к интерпретатору PHP и к самому сценарию.

ГЛАВА 45



Панель управления WordPress

45.1. О чем эта глава?

По сути, как интегрировать любой движок с панелью управления WordPress (ru.wordpress.com), было рассказано в *главе 43*. Для этого достаточно установить WordPress, удалить все, что касается пользовательской части, оставив только панель управления. А данные хранить в таблицах WordPress, при этом можно добавлять новые поля, что мы и сделали.

Так о чем же эта глава? Здесь вам ничего не нужно будет создавать. Просто посмотрите на уже готовый результат интеграции, чтобы понять, что получится на "выходе".

45.2. Вход в панель управления

Для входа в панель управления пройдите по адресу <http://caim/wp-admin/>. Вы увидите обычную, без всяких доработок панель управления WordPress (рис. 45.1). По сути, она и не нуждается в каких-либо доработках.

45.3. Просмотр целей

Чтобы просмотреть цели пользователей, перейдите в раздел **Записи** — ведь записи у нас соответствуют целям (рис. 45.2).

Категории целей можно добавить/удалить в разделе **Рубрики** (рис. 45.3).

Нужно отметить, что не стоит пытаться добавить новую цель с помощью панели управления, поскольку, добавляя необходимые нам поля, мы модифицировали таблицу `wp_posts`. Соответственно админка (панель администратора) WordPress ничего не знает об этих новых полях. Вывод один: или модифицировать админку, или добавлять цели исключительно только через пользовательский интерфейс (что проще и правильнее — ведь цели должны добавлять пользователи, а не администратор).

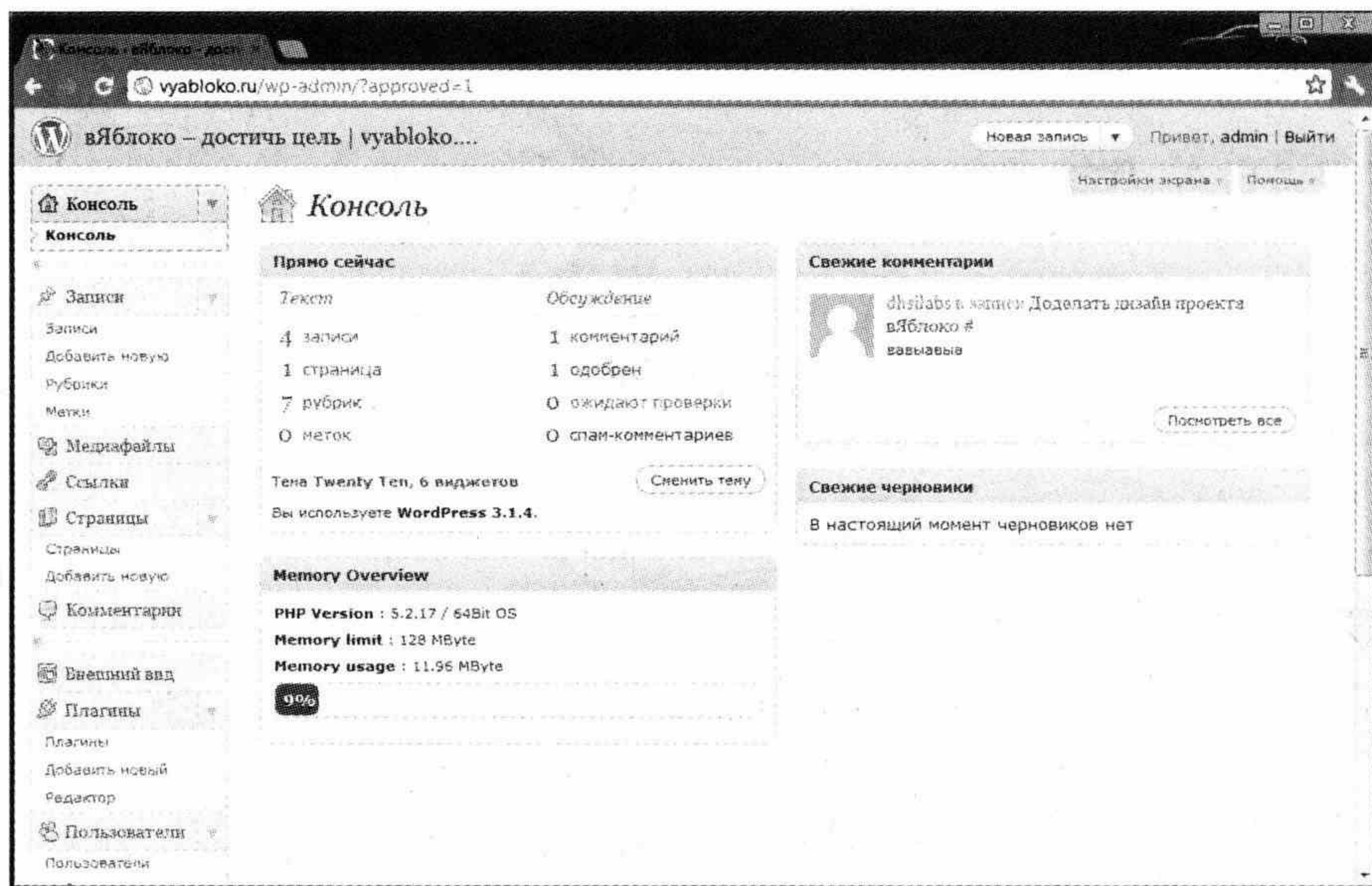


Рис. 45.1. Панель управления WordPress

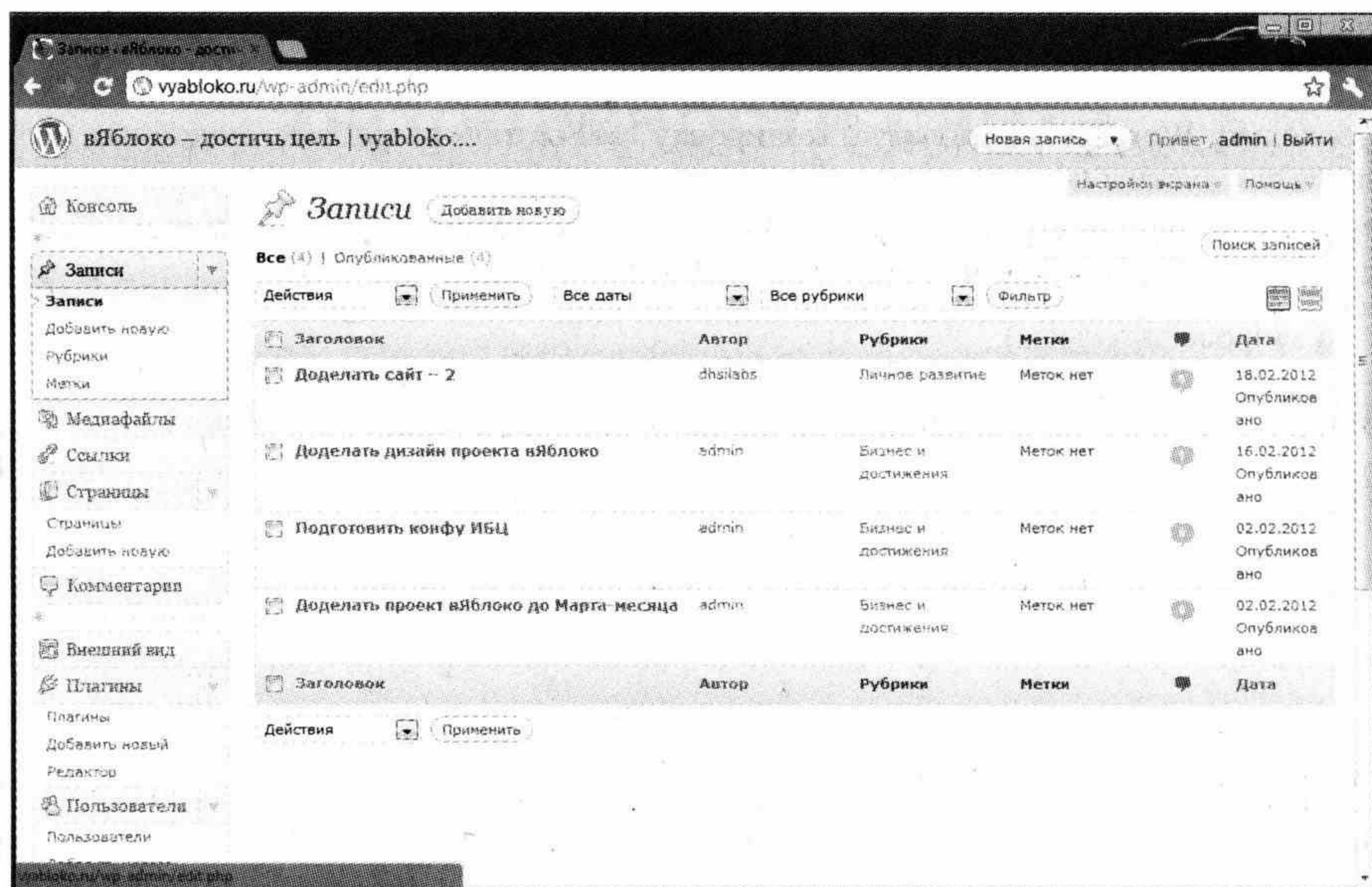


Рис. 45.2. Цели пользователей

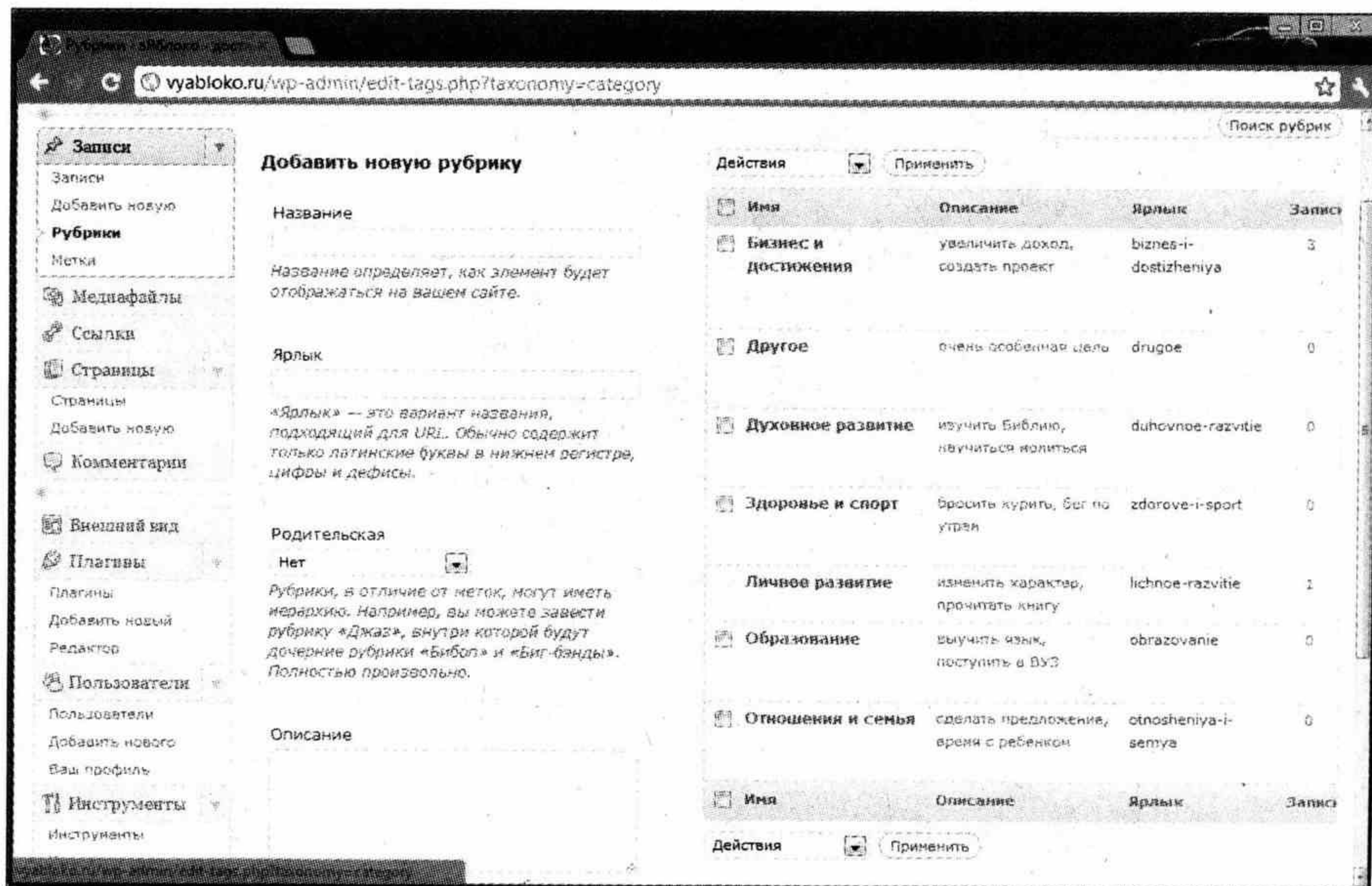
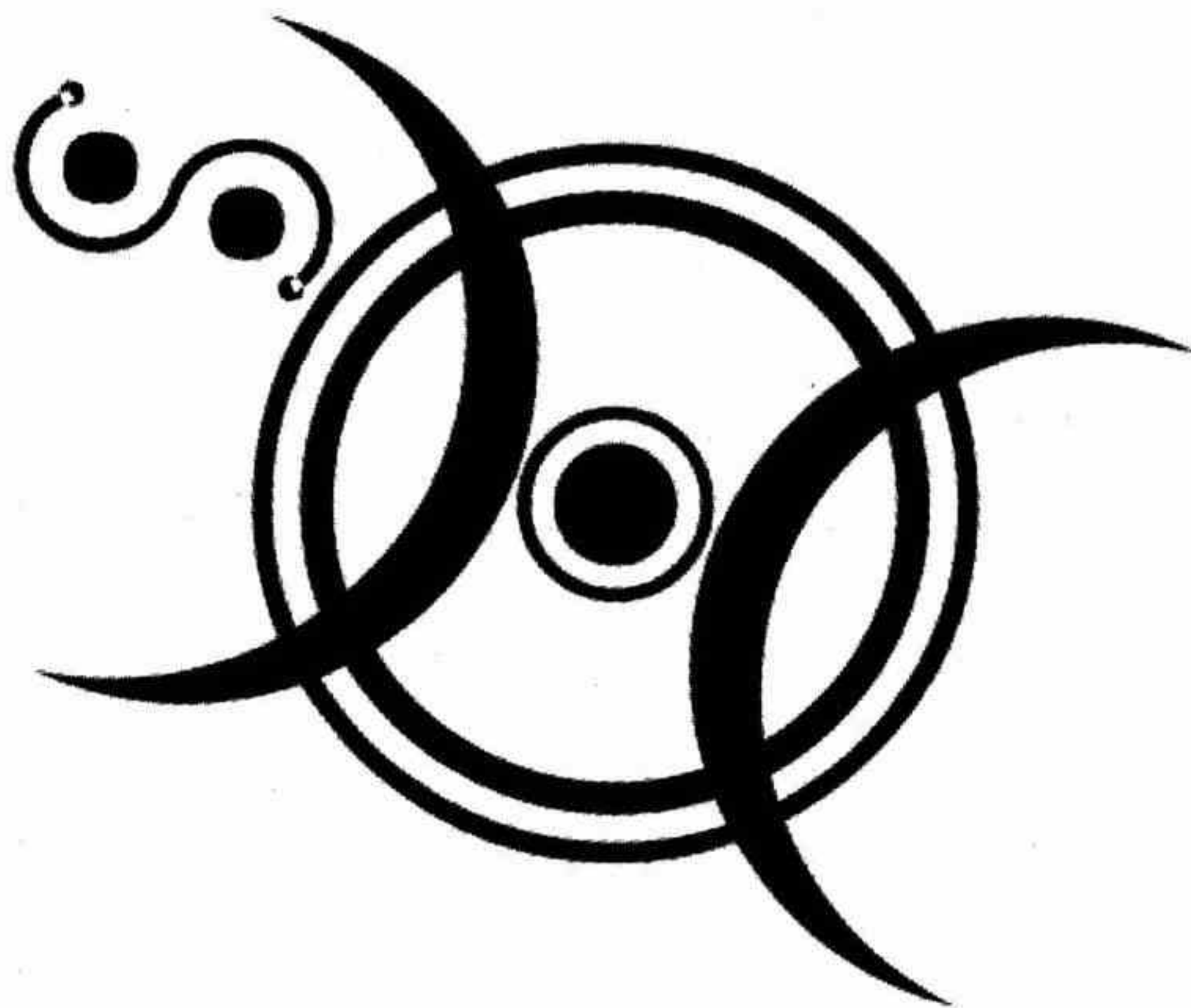


Рис. 45.3. Категории целей

45.4. Не забывайте о UTF-8

Поскольку WordPress использует кодировку UTF-8, при разработке подобных проектов вы должны помнить, что:

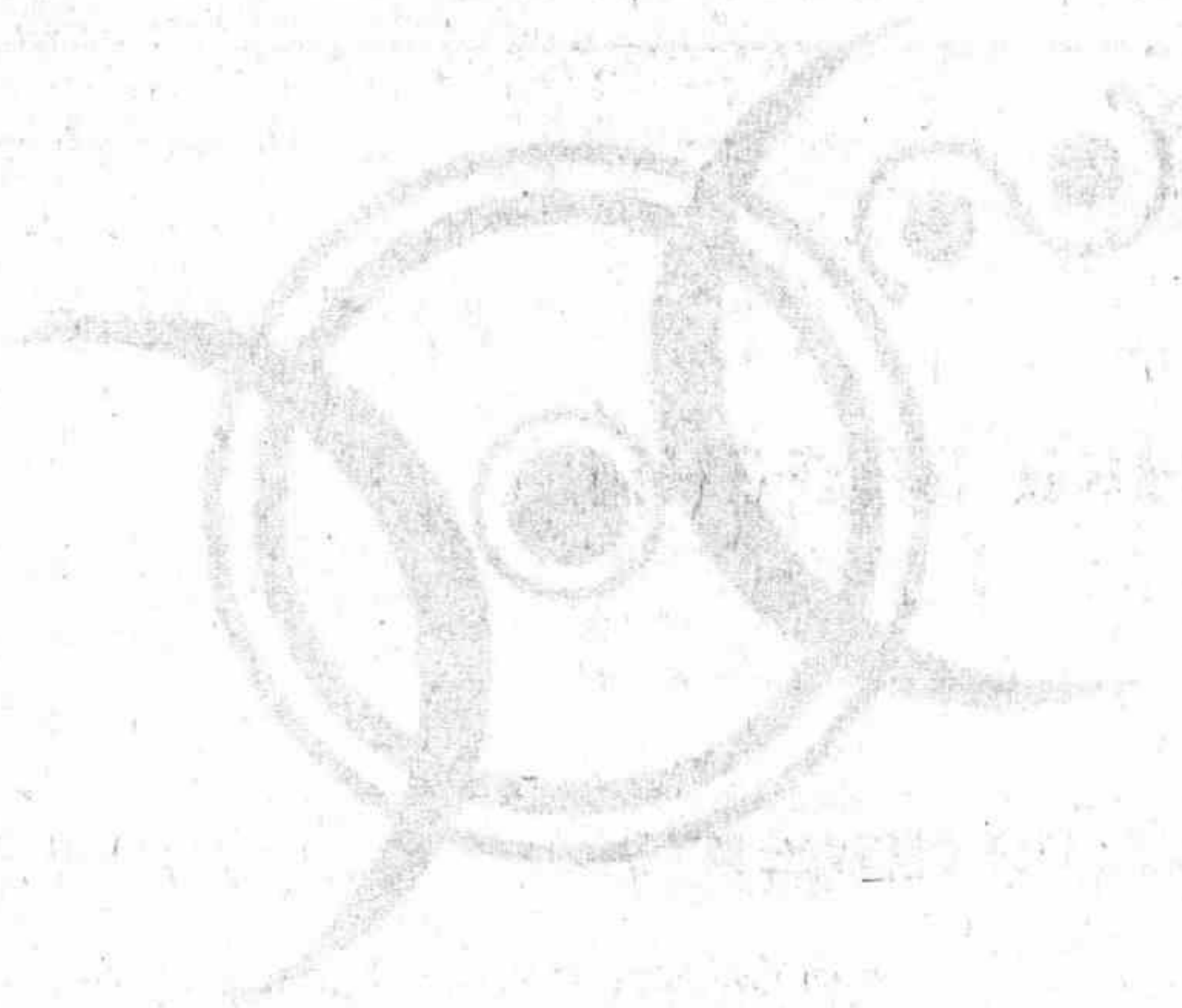
- ☐ все сценарии должны быть написаны в кодировке UTF-8;
- ☐ все, что передается в базу данных, должно передаваться в UTF-8 (не забывайте о запросе SET NAMES);
- ☐ при выводе информации нужно указать кодировку UTF-8 в HTML-коде и/или с помощью Header("Charset: utf-8").



II. РАЗДЕЛ 10

Обеспечение безопасности сайта

Глава 46.	SSL-соединения
Глава 47.	Защита PHP с помощью конфигурационного файла
Глава 48.	Защита сайта от атак



ALL PAGES 10

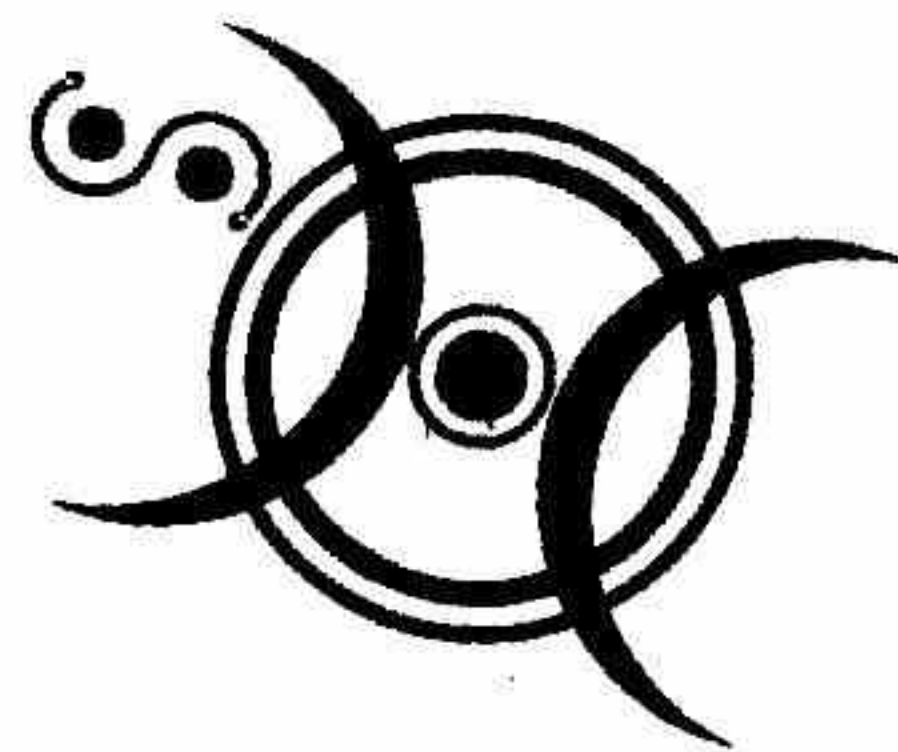
OPERATIONS

GENERAL INFORMATION

THE FOLLOWING INFORMATION IS FOR YOUR INFORMATION ONLY

PLEASE REFER TO THE ATTACHED DOCUMENTS

ГЛАВА 46



SSL-соединения

46.1. Защищаем передаваемые данные

Предположим, что вы разрабатываете сценарий интернет-магазина. При регистрации пользователь должен указать номер кредитной карточки и другие конфиденциальные данные. У вас самый защищенный сервер, который практически невозможно взломать, и вы не беспокоитесь о сохранности данных. Да, с вашего сервера украсть информацию невозможно. Но от клиента до сервера информация передается в абсолютно открытом виде, следовательно, по пути к серверу она может быть перехвачена! Для перехвата информации злоумышленнику не нужна особая квалификация — ему достаточно запустить сетевой сниффер, каких в Интернете очень много, — на любой вкус.

Защитить данные, передаваемые вашими клиентами, можно с помощью технологии SSL (Secure Socket Layer). Обратите внимание на имя протокола в строке браузера. Обычно это **http://**, но на некоторых сайтах, где нужно указать конфиденциальную информацию, используется другой протокол — **https://**. Это защищенная версия протокола HTTP. Протокол HTTPS подразумевает шифрование передаваемых данных, поэтому данными клиента, если они и будут перехвачены, злоумышленник воспользоваться не сможет. Конечно, и в этом случае можно обойти защиту, но такой взлом требует от злоумышленника очень высокой квалификации.

Технология SSL была разработана компанией Netscape Communications. Если вам приходилось когда-либо настраивать SSL на сервере Apache, то вы наверняка сталкивались с аббревиатурой TLS (Transport Layer Security). TLS — это развитие протокола SSL 3.0. По сути TLS 1.0 — это то же самое, что и SSL 3.0. Чтобы не указывать номер версии SSL, часто используют аббревиатуру TLS.

Протокол SSL включает в себя три основных фазы:

1. Выбор алгоритма шифрования — сервер и клиент договариваются, какой алгоритм будут использовать.
2. Обмен ключами — если используется криптосистема с открытым ключом — или же аутентификация на основе сертификатов.
3. Передача зашифрованных данных.

Дополнительную информацию о SSL и TLS можно получить по адресам: <http://ru.wikipedia.org/wiki/SSL> и <http://ru.wikipedia.org/wiki/TLS>.

Здесь мы не станем рассматривать настройку Apache для работы с SSL. Пусть этим занимаются администраторы службы технической поддержки хостера. Но вы, как программист, должны знать, как настроить SSL на своем хостинге. Мы рассмотрим настройку на примере популярной панели управления хостингом DirectAdmin. Если вам все же интересно, как настраивается SSL на сервере Apache с нуля, прочитайте следующую статью:

<http://www.dkws.org.ua/index.php?page=show&file=a/servers/ssl-ssh>.

46.2. Настройка SSL в DirectAdmin

Зайдите в панель управления вашим хостингом (рис. 46.1). Надеюсь, вы знаете, как это сделать. Если нет, то обратитесь в службу поддержки. Обратите внимание на строку адреса на рис. 46.1 — DirectAdmin для своей работы использует защищенное соединение!

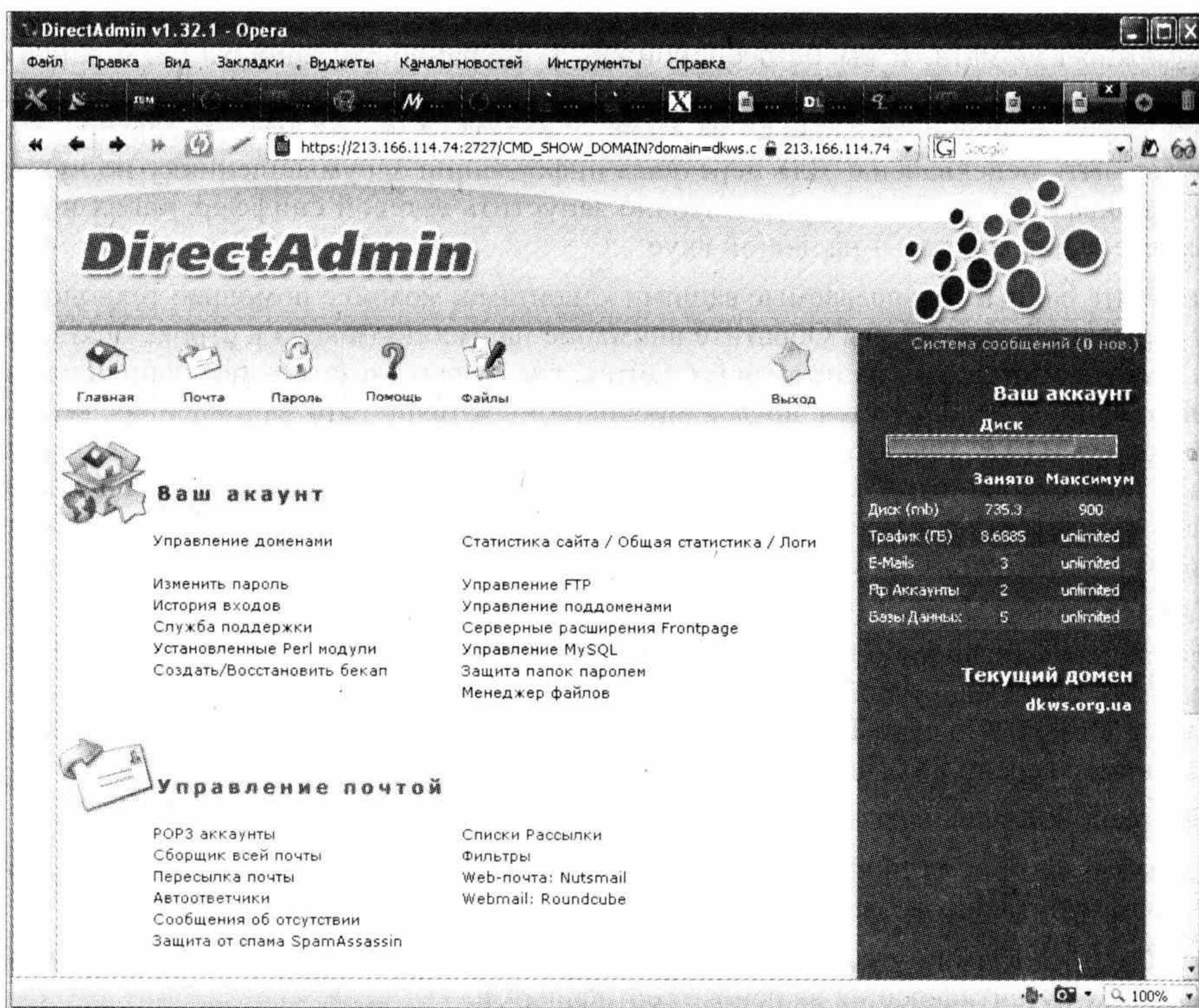


Рис. 46.1. Главная страница DirectAdmin

Перейдите по ссылке **Управление доменами** и выберите свой домен (у вас их может быть несколько). Если опция **Secure SSL** не включена, ее следует включить (рис. 46.2). Если же этой опции нет или она неактивна, ваш хостинг не поддерживает SSL. Обратитесь к службе поддержки. Возможно, для поддержки SSL нужно будет сменить тарифный план.

Обратите внимание на параметр **Настройки private_html**. По умолчанию все файлы, которые доступны веб-серверу, а значит, и всем пользователям Интернета, помещаются в каталог `public_html`. Доступ к этим файлам осуществляется по протоколу HTTP. После включения SSL сценарии, работающие с конфиденциальными данными (и формы ввода конфиденциальной информации), следует поместить в каталог `private_html`. После этого они станут доступны только по протоколу HTTPS. Например, имя моего узла **dkws.org.ua** я поместил в каталог `private_html` в сценарий `credit_card_info.php`. Получить доступ к нему теперь можно так: **https://dkws.org.ua/credit_card_info.php**.

По протоколу HTTP, понятное дело, этот сценарий доступен не будет, поскольку его нет в каталоге `public_html`.

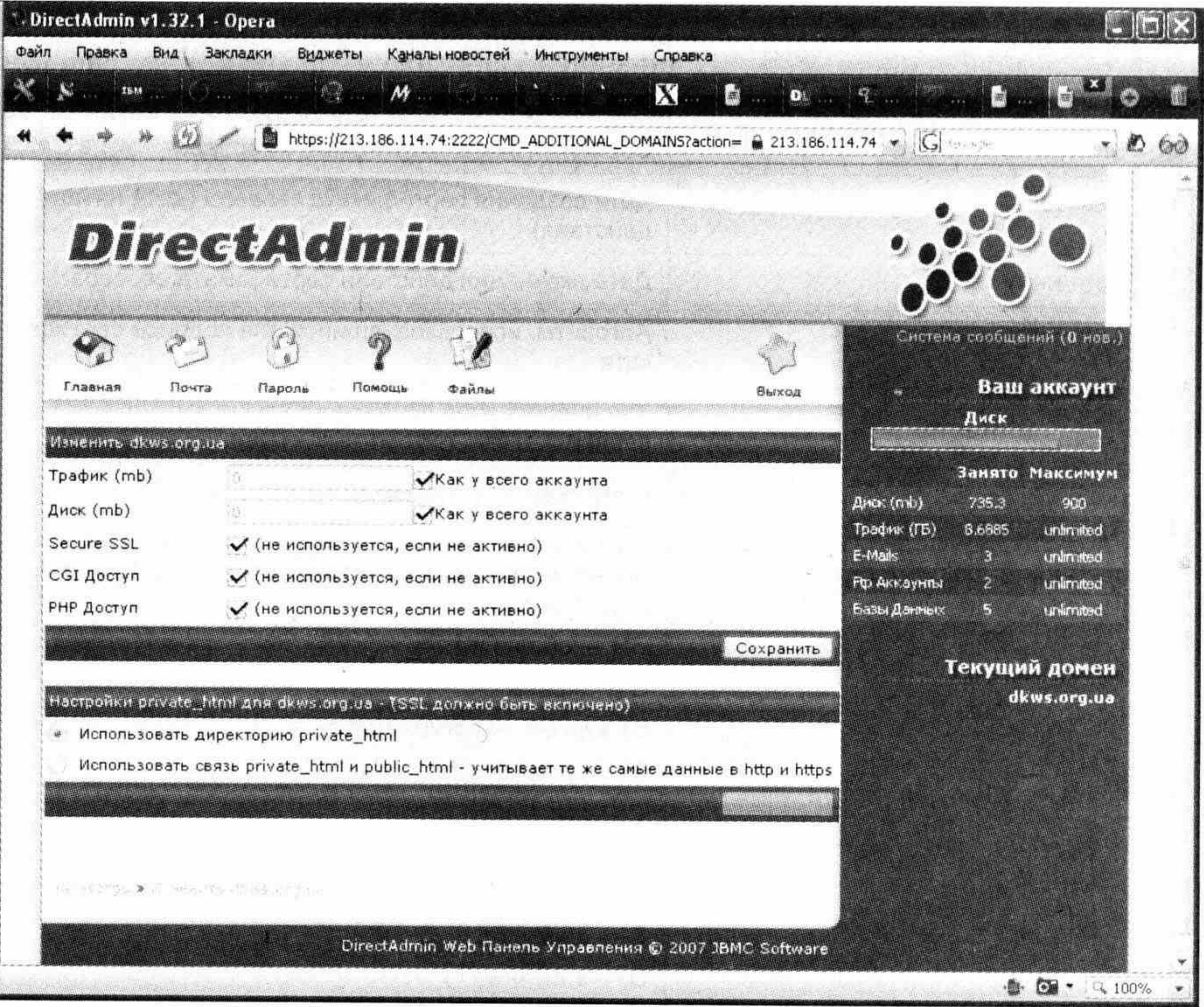


Рис. 46.2. Параметры домена

46.3. SSL-переменные

При разработке "защищенных" сценариев вы можете использовать CGI-переменные SSL-соединения, приведенные в табл. 46.1.

Таблица 46.1. CGI-переменные SSL-соединения

Переменная	Описание
SSL_PROTOCOL	Версия протокола SSL. Допустимые значения: SSLv2, SSLv3 или TLSv1
SSL_CIPHER и HTTPS_CIPHER	То же, что и SSL_PROTOCOL
SSL_CIPHER_EXPORT	Содержит true, если шифр соответствует действующим в США ограничениям на длину шифра (56 битов для симметричных ключей и 1024 для асимметричных)
SSL_CIPHER_USEKEYSIZE и HTTPS_SECRETKEYSIZE	Разрядность (в битах) текущего ключа
SSL_CIPHER_ALGKEYSIZE и HTTPS_KEYSIZE	Максимальная разрядность шифра
SSL_SERVER_M_VERSION	Версия сертификата сервера
SSL_SERVER_M_SERIAL	Серийный номер сертификата сервера
SSL_SERVER_V_START	Дата создания сертификата сервера (дата начала действия)
SSL_SERVER_V_END	Дата окончания действия сертификата сервера
SSL_SERVER_A_SIG	Алгоритм, использовавшийся для подписи сертификата
SSL_SERVER_A_KEY	Тип шифрования приватного ключа сертификата сервера
SSL_SERVER_S_DN и HTTPS_SERVER_SUBJECT	Кому выдан сертификат
SSL_SERVER_S_DN_компонент	Значение компонента SSL_SERVER_S_DN. Компонент может быть: CN — общее имя C — страна ST или SP — регион (штат) O — организация T — заголовок I — инициалы G — имя S — фамилия D — описание E-mail — электронный адрес для связи

Таблица 46.1 (окончание)

Переменная	Описание
SSL_SERVER_I_DN и HTTPS_SERVER_ISSUER	Кто выдал сертификат
SSL_SERVER_I_DN_компонент	То же самое, что и SSL_SERVER_S_DN_component, но для SSL_SERVER_I_DN

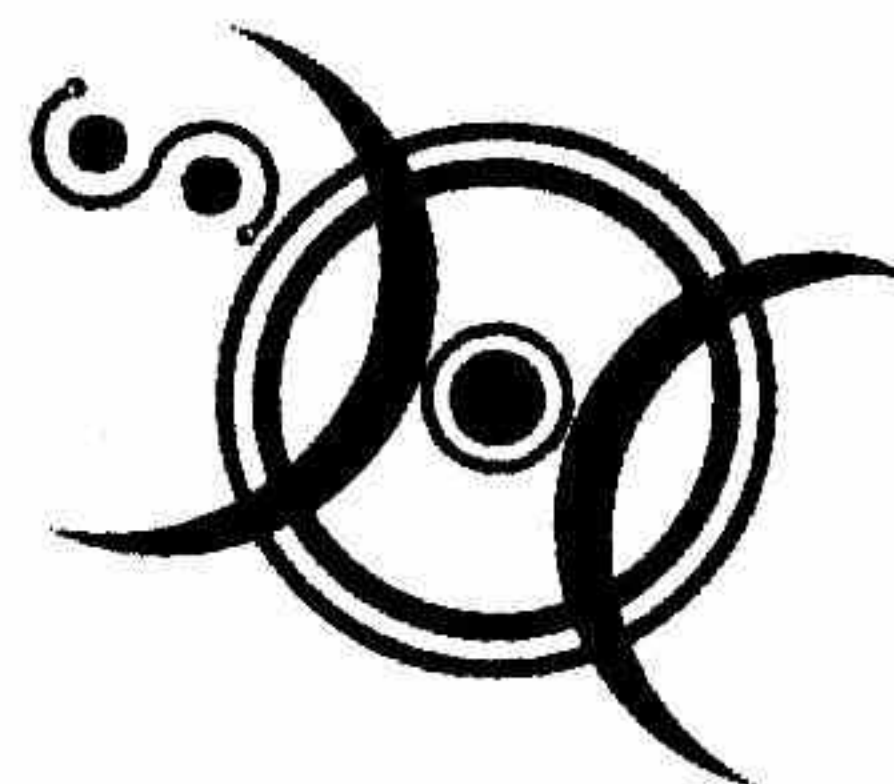
SSL-переменные устанавливаются только для SSL-соединения. Поэтому очень легко проверить тип соединения. Если переменная 'SSL_PROTOCOL' не установлена, то пользователь зашел на сайт по обычному протоколу HTTP. Вот пример перенаправления пользователя на безопасную версию сценария, работающую по протоколу HTTPS:

```
if (!$_SERVER['SSL_PROTOCOL']) {  
    header("Location: https://dkws.org.ua/index.php");  
    exit;  
}
```

Вот еще один метод проверки типа соединения:

```
$ssl=getenv("SSL_PROTOCOL");  
if ($ssl)  
    $http_proto="https";  
else  
    $http_proto="http";
```


ГЛАВА 47



Защита PHP с помощью конфигурационного файла

47.1. Конфигурационный файл php.ini

Конфигурационный файл `php.ini` содержит все настройки интерпретатора PHP. А здесь мы рассмотрим параметры PHP, относящиеся к безопасности. Но прежде разберемся, где этот файл находится.

В Linux/UNIX файл `php.ini` обычно лежит в каталоге `/etc`, но иногда его можно найти в каталогах `/usr/lib` или `/usr/local/lib`. В Windows файл `php.ini` находится обычно в каталоге `C:\php` (хотя это зависит от способа настройки связки Apache + PHP). Проще всего определить местонахождение этого файла — выполнить функцию `phpinfo()`. Ее вывод представлен на рис. 47.1. В поле `Loaded Configuration File` вы найдете полный путь к конфигурационному файлу. Как видно из рис. 47.1, это `/usr/local/lib/php.ini`.

Редактировать `php.ini` можно в любом текстовом редакторе. Вот пример директив этого файла:

```
max_execution_time = 30      ; Maximum execution time of each script,
                               in seconds
memory_limit = 8M           ; Maximum memory a script may consume
```

Если вас интересует какая-то директива, ее описание на русском языке можно найти на следующей странице: <http://ua2.php.net/manual/ru/ini.core.php>.

В Linux/UNIX для редактирования файла `php.ini` нужны права пользователя `root`. Другими словами, если у вас хостинг, а не выделенный сервер, вы его редактировать не сможете.

Все директивы конфигурационного файла `php.ini` мы рассматривать не будем. В этом нет смысла, поскольку в большинстве случаев вы не сможете их изменить.

Однако вы можете включить ту или иную директиву для своего сценария с помощью функции `ini_set()`:

```
string ini_set ( string $директива, string $значение )
```

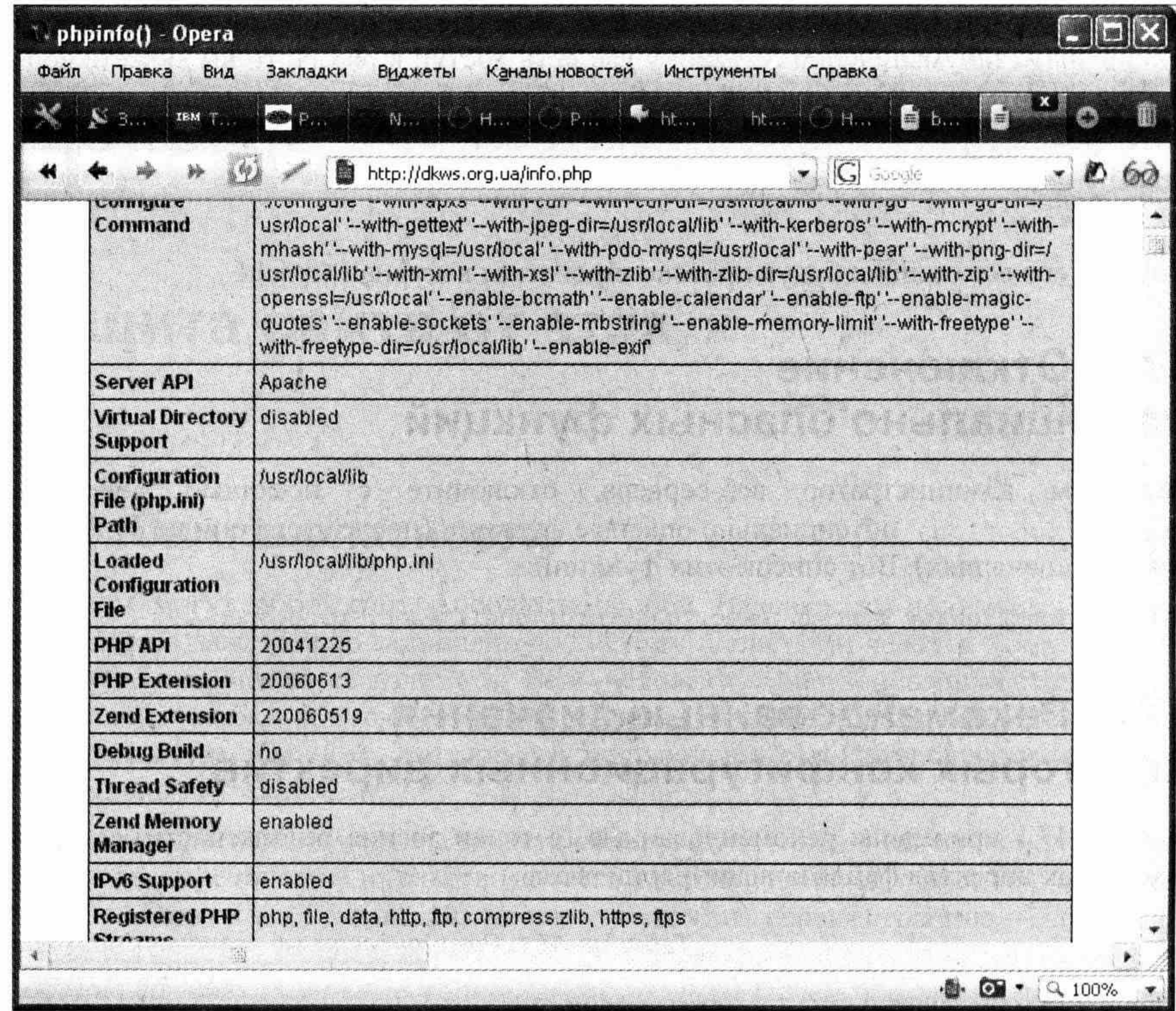



Рис. 47.1. Вывод phpinfo()

Например:

```
ini_set("max_execution_time", "30");
```

Для установки значения директивы конфигурации служит функция `ini_get()`, например:

```
echo "max_execution_time =" . ini_get('max_execution_time');
```

Одна из очень полезных директив файла конфигурации — `error_reporting`. Она управляет сообщениями об ошибках и предупреждениями, которые будут выводиться интерпретатором. Во время разработки сценария нужно включить вывод всех ошибок и других сообщений интерпретатора. Для этого служит функция:

```
error_reporting(E_ALL);
```

Вызов этой функции должен идти в сценарии самым первым. Когда сценарий отлажен и работает без ошибок, вывод всех ошибок следует выключить:

```
error_reporting(0);
```


Дело в том, что при выводе ошибки PHP показывает информацию, которую не нужно знать злоумышленнику. Если у вас есть доступ к `php.ini`, то вы можете вообще выключить вывод сообщений интерпретатора в браузер, но включить их протоколирование. Для этого служат две следующие директивы:

```
display_errors = Off
log_errors = On
```

После этого все ошибки будут протоколироваться в журнал Apache.

47.2. Отключение потенциально опасных функций

Если вы администратор веб-сервера, отключите с помощью директивы `disable_functions` потенциально опасные функции (после этого они не будут доступны в сценариях). Вот список этих функций:

```
disable_functions = system, exec, passthru, shell_exec, proc_open
```

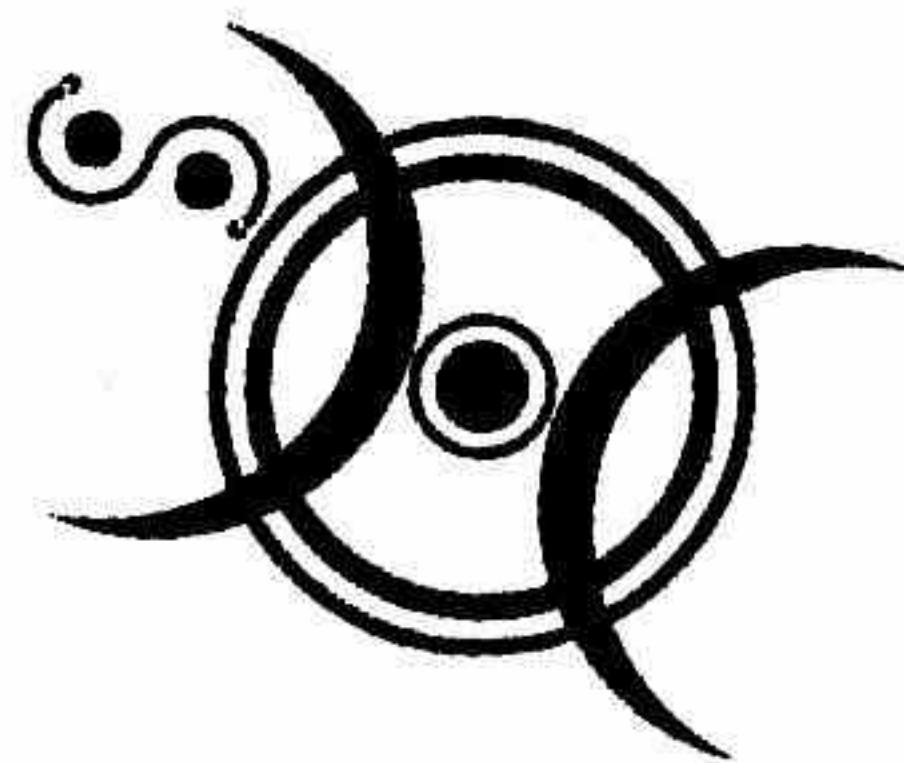
47.3. Рекомендованные значения некоторых конфигурационных директив

В табл. 47.1 приведены рекомендованные (с точки зрения безопасности) значения некоторых директив файла конфигурации `php.ini`.

Таблица 47.1. Рекомендованные значения директив

Директива и значение	Комментарий
<code>register_globals = off</code>	Данная директива исключена из PHP 6, но если у вас PHP 5, то ее нужно выключить
<code>safe_mode = off</code>	Безопасный режим на самом деле ничего не помогает обезопасить, поэтому эту директиву нужно выключить
<code>error_reporting = off</code>	Не выводить сообщения об ошибках. Чем меньше злоумышленник знает о нашей системе, тем лучше
<code>log_errors = on</code>	Включить протоколирование ошибок в файл журнала
<code>open_basedir = /tmp, ...</code>	Здесь нужно указать два каталога: корневой каталог веб-сервера и каталог <code>/tmp</code> . Не забудьте его указать, иначе сессии работать не будут
<code>expose_php = off</code>	Не добавлять подпись PHP к заголовкам Apache (чтобы никто не узнал вашу версию PHP)
<code>allow_url_open = off</code>	Запрещает обращаться к удаленным файлам по протоколу HTTP

ГЛАВА 48



Защита сайта от атак

48.1. Сайт в опасности

Взломать могут любой сайт. Абсолютно любой. Надеюсь, вы понимаете, что взломать можно даже самую защищенную систему. Если один человек защитил систему, то другой может ее взломать. Ведь человеку свойственно ошибаться. Кто-то совершает ошибки, а кто-то этим пользуется. От этого никуда не денешься. Все серверы, как и обычные компьютеры, подключенные к Интернету, являются потенциальными "жертвами".

В один прекрасный момент вы можете обнаружить, что ваш сайт взломан. Об этом будет красноречиво свидетельствовать его главная страница, оформленная в стиле "эТоТ сАЙт взлОмал ВаСя". Но то, что уничтожена первая страница — еще полбеды, ее легко восстановить, плохо если вы потеряете свою базу данных. Ведь не все хостеры создают резервные копии данных пользователей, а если и делают это, то не каждый день.

Почему взламываются сайты? Кто-то делает это для удовлетворения собственного "я". Как правило, это молодые люди, которым хочется доказать кому-то (например, владельцу сайта) свое превосходство. Кто-то взламывает сайты по заказу.

Как вы думаете, каких "крекеров" больше: любителей, которые взламывают сайты от нечего делать (для них это вроде хобби), или профессионалов? Понятно, что любителей больше. Теперь посмотрим на все это глазами взломщика-любителя. Какой бы сайт вы взломали — менее защищенный или хорошо защищенный? Правильно. Менее защищенный. Ведь ваша цель — это сам факт взлома. Главное взломать, а какой именно сайт — это уже второстепенный вопрос.

Здесь мы поговорим о том, как защитить свой сайт от Васи Пупкина, который в свои 15 лет возомнил себя великим кречером. Защиту от профессионального взлома мы рассматривать не будем, поскольку это потребует не только правильного написания кода, но и защиты самого сервера, на котором размещен ваш сайт, что возможно только при тесном сотрудничестве с хостинг-провайдером.

Прежде чем приступить к защите вашего сайта, хочется поговорить немного о кречерах и хакерах, ведь наверняка найдутся читатели, для которых слово "кре-

кер" ассоциируется, скорее, с пищей. Почему я не употребляю здесь слово "хакер"? Да потому, что хакер, по определению, — это программист высочайшей квалификации, способный создавать программы без предварительной разработки детальной спецификации, а также вносить изменения в работающие программы, в том числе и на уровне машинного кода. Как видите, о взломе систем не сказано ни слова. Но СМИ ошибочно навязали общественности другое понятие, заключающееся в том, что хакеры связаны со взломом компьютерных систем. Нет, наоборот, хакеры в компьютерном мире выполняют созидательную функцию, а не разрушительную. А вот взломом занимаются специалисты несколько другого профиля — крекеры (от англ. *cracker*). Чтобы это слово не резало слух, "крекеры" мы далее будем называть злоумышленниками.

48.2. Два самых распространенных метода взлома

При взломе сайтов чаще всего применяются два метода: так называемый *межсайтовый скриптинг* (XSS) и *SQL-инъекции* (SQL injection). Первый метод заключается в том, что злоумышленник внедряет свой HTML- или PHP-код в сценарий страницы сайта. Причем возможности этого практически ничем не ограничены — можно вставить зловредный код даже через якобы безопасный тег `img`, предназначенный для отображения картинки. Как правило, от этого страдают плохо защищенные гостевые книги или форумы, позволяющие использовать HTML-теги.

SQL-инъекции еще более опасны, поскольку внедрение SQL-кода может привести к краже информации и даже к полному уничтожению базы данных. Тут уже и не знаешь, что хуже — потеря или кража информации. Возможен и третий вариант — кража с потерей, т. е. базу данных сначала украли, а потом уничтожили.

Далее мы рассмотрим оба метода достаточно подробно.

48.3. Межсайтовый скриптинг

Проблема XSS-атак известна уже давно, но тем не менее она до сих пор остается актуальной, поскольку им подвержено очень много сайтов, да и сами эти атаки нельзя назвать безопасными. Особенность такой атаки заключается в том, что взламывается не сервер хостинг-провайдера (ведь в основном провайдеры уделяют огромное внимание безопасности собственных серверов), а плохо защищенный сценарий клиента. Естественно, далеко не все их авторы являются "специалистами с большой буквы", вот и получается, что каждый день появляются уязвимые сайты. Дело в том, что разработчик, который создавал сценарий, даже и не подозревал, что кто-то может вот так просто взломать его сайт. Да и отслеживать такие атаки сложно, потому что веб-серверы протоколируют далеко не все запросы (например, POST-запросы вообще редко протоколируются, следовательно, можно передать любой POST-запрос и остаться незамеченным).

Как осуществляется XSS-атака? Злоумышленник конструирует специальный URL, который передает своей жертве. Цель? Чаще всего вставка определенного HTML-кода осуществляется для рекламы или перенаправления на другой сайт (который, как правило, находится под контролем злоумышленника). Бывают и сложные XSS-атаки, когда злоумышленник перехватывает сессию, но это довольно редкий тип атаки, поскольку злоумышленнику нужен полный доступ к веб-серверу, на котором запущен сценарий, а имея такой доступ, можно навредить сайту и без всяких XSS-атак.

Рассмотрим небольшой пример уязвимого сценария:

```
<?php  
echo "Привет $name";  
?>
```

Сценарию нужно передать имя (параметр \$name), которое будет выведено в браузер:

`http://localhost/index.php?name=Denis.`

Если передать просто имя, тогда мы увидим то, что и ожидали:

Привет, Denis

Но что, если передать более сложный HTML-код — например:

`http://localhost/index.php?name=<img+src=http://host.ru/bannder.jpg>.`

Будет выведено слово **Привет** и указанная нами картинка!

В нашем случае ничего страшного не произойдет. Просто злоумышленник порадует, увидев свою картинку. Но ведь иногда информация передается сразу в базу данных, без предварительной проверки! А потом при ее чтении из базы данных будет выведен HTML-код, оставленный "на память" злоумышленником.

Но и это еще не самая большая беда. Ведь злоумышленник таким образом может передать не только HTML-код, но и JavaScript, а иногда даже и PHP-код. Вот небольшой пример передачи JavaScript-кода:

`http://localhost/index.php? name=<script>alert (document.cookie)</script>.`

Если просмотреть исходный код страницы, то он будет выглядеть так:

Привет `<script>alert (document.cookie)</script>`

Как же уберечься от XSS-атак? В PHP есть две функции, которые могут помочь делу:

- ❑ `strip_tags()` — удаляет из строки все HTML-теги, кроме разрешенных;
- ❑ `htmlspecialchars()` — заменяет все специальные символы на их HTML-эквиваленты (амперсанд заменяется `&`, символ `<` — `<` и т. д.).

Рекомендую сначала удалить все HTML-теги, а затем пропустить полученные извне данные через функцию `htmlspecialchars()` — для надежности:

```
$name = strip_tags($name);  
$name = htmlspecialchars($name);
```


Второй параметр функции `strip_tags()` позволяет указать список разрешенных HTML-тегов, например:

```
$name = strip_tags($name, '<p><b>');
```

Но я настоятельно не рекомендую разрешать какие-либо HTML-теги, даже самые безобидные вроде `<p>` и `<b>`.

В дополнение к этому желательно работать с переменными так, будто бы директива `register_globals` отключена. Так вы сможете контролировать способы поступления информации — например, только по методу GET:

```
$name = $_GET['name'];
```

Ведь при включенной директиве `register_globals` в PHP нет разницы, как был передан параметр: с помощью метода GET или POST. А злоумышленник может специально передать данные методом POST, чтобы его запрос не был внесен в журнал сервера. Если вы сами контролируете способ передачи данных, то такую ситуацию можно исключить.

48.4. SQL-инъекции

SQL-инъекции по своей природе очень похожи на XSS-атаки и различаются только деталями. Цель XSS-атаки — выход за пределы HTML-тега, а SQL-инъекции — внедрение SQL-кода, что может привести к краже информации и даже к полному уничтожению базы данных.

Рассмотрим тривиальный пример SQL-инъекции. Предположим, что у нас есть таблица `users`, содержащая информацию о пользователях:

```
id int(11) NOT NULL auto_increment,  
login tinytext,  
password tinytext,  
email tinytext,  
PRIMARY KEY (id_user)
```

Авторизацию пользователя выполняет сценарий `login.php` (листинг 48.1).

Листинг 48.1. Уязвимый сценарий

```
<?php  
// В файле config.php — параметры соединения с сервером MySQL  
include "config.php";  
  
// подключаемся к серверу  
mysql_connect($SERVER, $USER, $PASSWORD);  
mysql_select_db($DB);  
  
// сценарию через форму передаются две переменные: $login и $pass  
// если в таблице users будет запись, поле login которой содержит
```



```
// значение, равное значению переменной $login, и поле password
// содержит значение, равное значению переменной $pass, то результат
// данного запроса будет содержать одну строку, в противном случае
// число строк результата будет равно нулю $query = "SELECT * FROM users
        WHERE login = \"$login\" AND password = \"$pass\"";
// выводим для отладки запрос, чтобы вы видели, что происходит
echo $query;

$r = mysql_query($query);

if(mysql_num_rows($r)>0)
{
    echo "<p>аутентификация прошла успешно";
}
else echo "<p>access denied";

?>
```

ПРИМЕЧАНИЕ

Код этого сценария вы найдете в файле *43-1.php* каталога *Glava_43* сопровождающего книгу электронного архива (см. приложение 4).

Нашему сценарию нужно передать два параметра: *\$login* и *\$pass*. Форма для передачи этих параметров может выглядеть так:

```
<form action=sql.php method=get>
Login <input type=text name=login>
Pass <input type=text name=pass>
<input type=submit name=go>
</form>
```

Предположим, что в нашей таблице есть пользователь *admin* с паролем *123*. Если ввести эти имя пользователя и пароль, то мы увидим в окне браузера следующий текст:

SELECT * FROM users WHERE login = "admin" AND password = "123"

аутентификация прошла успешно

Все как бы работает правильно. Теперь вместо имени пользователя введем *admin"/**, а пароль вообще не будем указывать. В окне браузера увидим следующее (рис. 48.1):

SELECT * FROM users WHERE login = "admin"/*" AND password = ""

аутентификация прошла успешно

Получается интересная ситуация — вместо имени пользователя ввели строку, только отчасти похожую на нужную, пароль вообще не указали, а сценарий пишет, что авторизация прошла успешно.



Рис. 48.1. Результат SQL-инъекции

Оказывается, ничего сверхъестественного не произошло. Вы только взгляните на наш SQL-запрос:

```
SELECT * FROM users WHERE login = "admin"/*" AND password = ""
```

На самом деле он выглядит так:

```
SELECT * FROM users WHERE login = "admin"
```

Символы `/*` считаются началом комментария, поэтому все, что находится после них, просто не учитывается. В результате наш запрос просто запрашивает строку, содержащую информацию о пользователе `admin`. Поскольку у нас есть пользователь с таким именем, то число строк результата будет больше нуля, следовательно, сценарий "радостно" сообщит там, что:

аутентификация прошла успешно

Выходит, все, что необходимо знать злоумышленнику, — это имя пользователя, которого нужно "взломать".

Что же нам делать? Во-первых, следует изменить алгоритм проверки имени пользователя и пароля. Сначала нужно запросить всю запись, содержащую информацию о пользователе `admin`, а затем сравнить переданный пароль со значением поля `password`. Сказанное иллюстрирует листинг 48.2.

Листинг 48.2. Сценарий без уязвимости

```
<?php
// В файле config.php — параметры соединения с сервером MySQL
include "config.php";
// подключаемся к серверу
mysql_connect($SERVER, $USER, $PASSWORD);
mysql_select_db($DB);

$query = "SELECT * FROM users WHERE login = \"\$login\"";

echo $query;
$r = mysql_query($query);
```



```

if (mysql_num_rows($r)>0)
{
    $f = mysql_fetch_array($r);

    echo "<p>А теперь проверяем пароль...";

    if ($f['password']===$pass)
        echo "<p>авторизация прошла успешно";
    else
        die("<p>access denied");
}
else echo "<p>access denied";

?>

```

ПРИМЕЧАНИЕ

Приведенный здесь код вы найдете в файле *48-2.php* каталога *Glava_48* сопровождающего книгу электронного архива (см. приложение 4).

Вывод этого сценария представлен на рис. 48.2. Мы опять ввели вместо имени пользователя *admin"/**, а пароль не указали. Но в этот раз у нас ничего не вышло. Сценарий получил имя пользователя и всю запись этого пользователя из таблицы, но при проверке пароля оказалось, что переданный код не соответствует действительности, поэтому сценарий нас не "впустил".

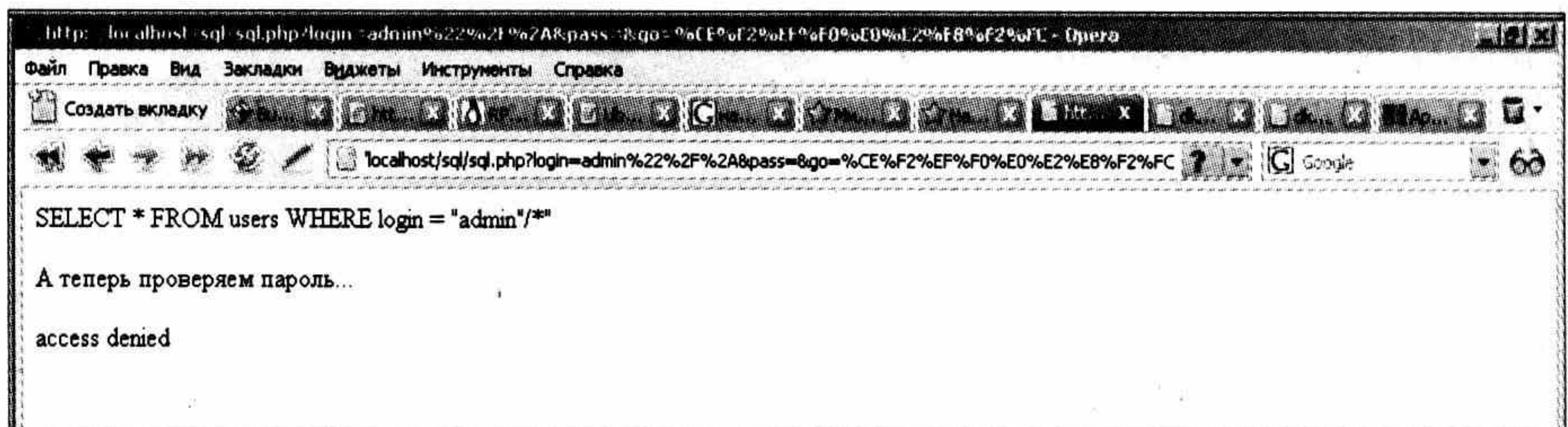


Рис. 48.2. Результат выполнения листинга 48.2

Желательно также обработать передаваемые переменные функциями *addslashes()* и *htmlspecialchars()*. Вторая функция нам уже встречалась, а первая, если вы не забыли, добавляет слэши перед кавычками:

```

$login = addslashes($login);
$login = htmlspecialchars($login);

```

В нашем случае вызывать указанные функции для переменной *\$pass* нет смысла, поскольку она не участвует в SQL-запросе.

Но даже все проделанное не всегда гарантирует стопроцентную защиту. А что, если вместо имени пользователя для авторизации используется ID пользователя, т. е. его числовой идентификатор в системе? Тогда в запросе не будет кавычек, и злоумышленник сможет вообще исполнить любой SQL-код с помощью оператора UNION. Все, что ему потребуется ввести, — это строку '1 UNION <SQL-код>' (без апострофов) в качестве идентификатора пользователя.

Чтобы не позволить злоумышленнику выполнить любой SQL-код, нужно перед помещением полученных данных (идентификатора пользователя) в SQL-запрос проверить их с помощью функции `is_numeric()`:

```
if (is_numeric($id)) {  
    $query = "select * from users where id = $id";  
    ...  
}
```

Так и только так вы защитите себя от этого вида SQL-инъекций.

48.5. Флуд

Флуд (от англ. *flood*) — это автоматическое размещение однотипных сообщений на форуме, в чате или в гостевой книге. Злоумышленник копирует форму для отправки сообщений на форум (в чат, гостевую книгу) и пишет несложную программку, имитирующую работу браузера, — она будет автоматически отправлять одно и то же сообщение на форум.

Решение этой проблемы очень простое — нужно всего лишь добавить в форму сессию. Файл `form.php` приведен в листинге 48.3.

Листинг 48.3. Файл `form.php`

```
<?php  
// запускаем сессию  
session_start();  
?>  
<form action=index.php method=post>  
  
<код формы>  
  
<input type=hidden name=ssid value=<?php echo session_id();?>>  
</form>
```

ПРИМЕЧАНИЕ

Приведенный здесь код вы найдете в файле `form.php` каталога `Glava_48` сопровождающего книгу электронного архива (см. приложение 4).

Как видите, мы добавили в форму скрытый параметр `ssid`, т. е. идентификатор сессии. Потом в сценарии `index.php` нужно будет сравнить идентификаторы сессии (листинг 48.4).

Листинг 48.4. Сценарий index.php

```
<?php
// запускаем
session_start();
// сравниваем настоящий идентификатор сессии с переданным из формы
// если они не совпадают, сценарий прекращает свою работу
if($_POST['ssid'] != session_id())
    die('Попытка флуда!!! Все в сад!!!!');
...
```

Надеюсь, я помог вам немного защитить свои сценарии. Но на всякий случай рекомендую пригласить стороннего специалиста для тестирования ваших сценариев на уязвимость. Самому можно даже и не пытаться тестировать, ведь так устроен человек — свои ошибки мы замечаем лишь тогда, когда уже поздно.

48.6. Защита форума PHPBB2 от спамеров

Форум PHPBB версии 2 сильно устарел, но он все еще установлен на многих сайтах. Бич второй версии — спамеры. Спам появляется на форуме, даже если разрешить писать сообщения только зарегистрированным пользователям, а при регистрации включить код подтверждения. Боты (программы-роботы) спамеров уже давно научились обходить код подтверждения phpbb2.

Но выход есть. Идея проста, как и все гениальное. Все боты написаны из расчета, что будет использоваться стандартная, никем не модифицированная, версия phpbb2. Мы можем добавить еще одно поле, предназначенное для ввода дополнительной информации при регистрации на форуме, а потом "смотреть", что ввел пользователь (или спам-бот). Пользователь ответит на вопрос, а спам-бот — нет, поскольку он ничего не знает об этом поле.

Реализуем задуманное. Откройте файл `profile_add_body.tpl` темы, активной в данный момент. Если мы имеем дело со скином `subSilver`, этот файл нужно искать в каталоге `templates/subSilver`. Добавьте в этот файл описание нового поля:

```
<tr>
<td class="row1" width="38%"><span class="gen">3 умножить на 3</span></td>
<td class="row2"><input type="text" style="width:200px" name="spam" value="0"
/></td>
</tr>
```

Наш дополнительный вопрос при регистрации нового пользователя будет звучать так: **3 умножить на 3**. В новом поле пользователь должен ввести ответ — **9**. Обратите внимание на имя переменной — `spam`. Желательно заменить это имя на любое другое, чтобы спамеры, прочитав эту книгу, не создали новую версию спам-бота. Вопрос тоже желательно изменить.

Далее нужно модифицировать файл `includes/usercp_register.php`. Найдите в нем следующие строки:

```
"U_AGREE_OVER13" =>
append_sid("profile.$phpEx?mode=register&agreed=true&"),
"U_AGREE_UNDER13" =>
append_sid("profile.$phpEx?mode=register&agreed=true&coppa=true")
```

Эти строки нужно заменить следующими:

```
"U_AGREE_OVER13" =>
append_sid("profile.$phpEx?mode=register&agreed=true&spam=9"),
"U_AGREE_UNDER13" =>
append_sid("profile.$phpEx?mode=register&agreed=true&coppa=true;spam=9")
```

Мы добавили в них правильный ответ (в данном случае — 9) на наш вопрос. Если вы изменили имя переменной (вместо имени `spam` используете любое другое), то именно его и нужно указать.

Теперь найдите следующую строку:

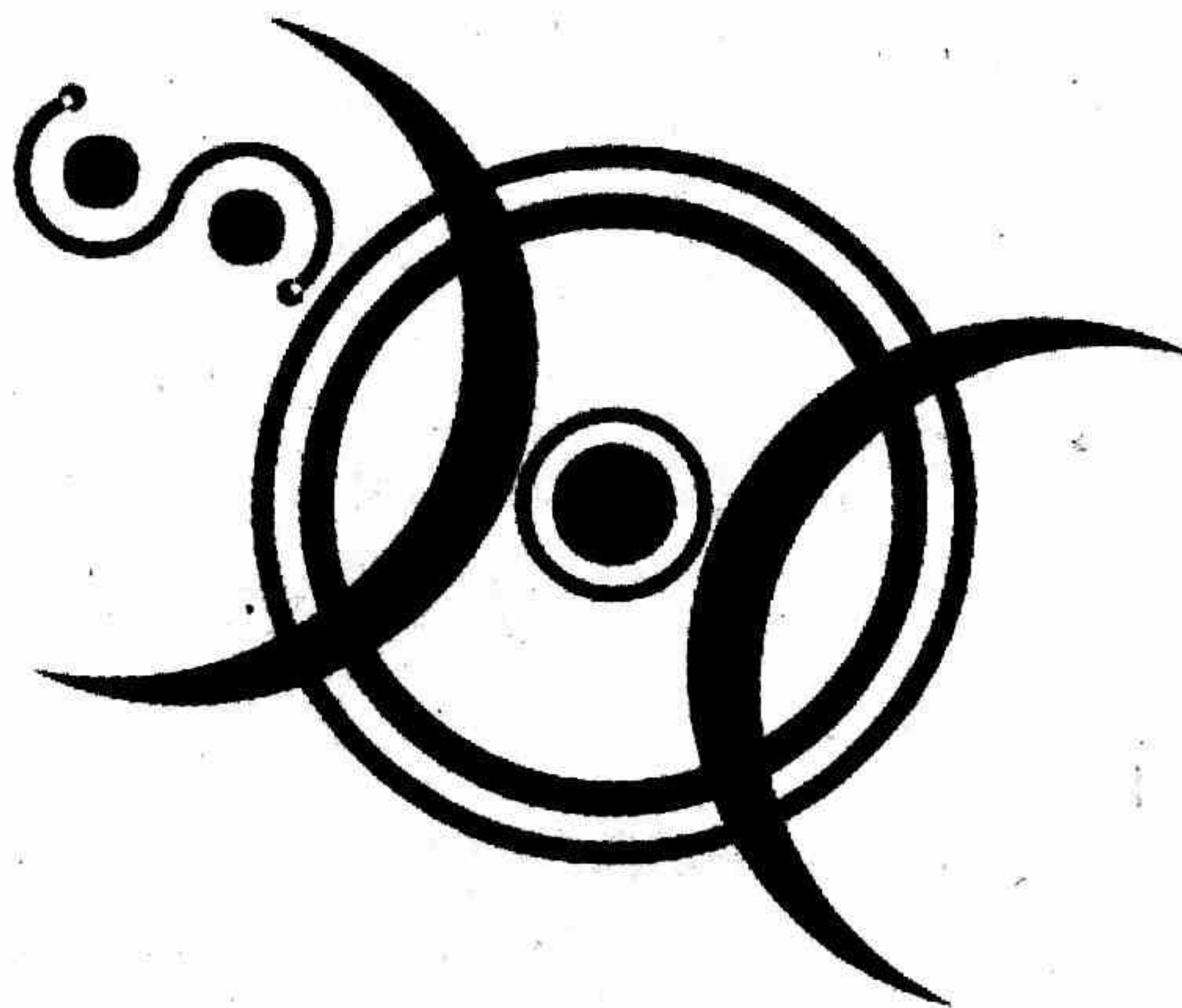
```
if ( $mode == 'register' )
```

После нее добавьте следующие строки:

```
// получаем значение переменной spam — то, что ввел пользователь
$answer = $HTTP_POST_VARS['spam'];
if ($answer=='') $answer = $HTTP_GET_VARS['spam'];
// сообщение, которое будет выведено пользователю (или боту),
// если ответ неправильный
$mes = <h1>Считать разучились? </h1>";

// если ответ неправильный, то завершаем работу сценария
// и выводим сообщение
if ($answer!='9') die ($mes);
```

Как видите, все предельно просто. Мы получаем ответ пользователя и, если он неправильный, прерываем сценарий регистрации нового пользователя. А поскольку незарегистрированные пользователи не могут ничего писать на форум, то спама у вас не будет. Спам возможен только в одном случае — если спамер зарегистрируется на форуме вручную и отправит сообщение со спамом. Но, как правило, делать ему это будет лень, поэтому на 99 % ваш форум теперь защищен от спама.

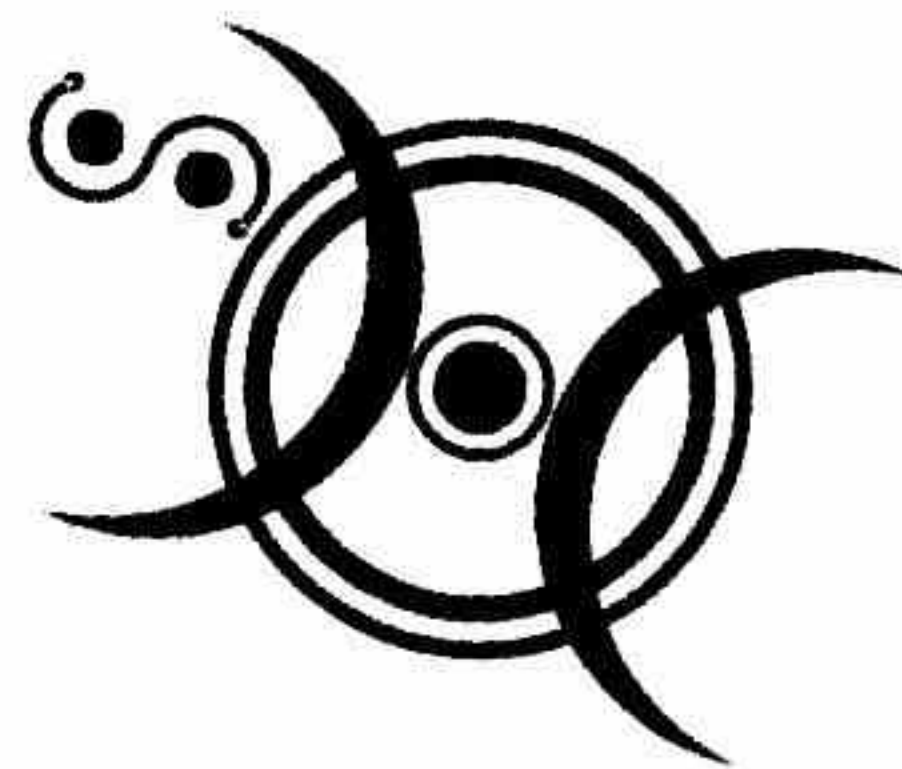


II. РАЗДЕЛ 11

Полезные сведения

Глава 49.	Визуальный редактор SPAW2 для вашего сайта
Глава 50.	WAP-сервер средствами PHP
Глава 51.	Пишем коммерческий сценарий

ГЛАВА 49



Визуальный редактор SPAW2 для вашего сайта

49.1. Знакомство с визуальным редактором и его базовая настройка

Представим, что вы разрабатываете блог или любой другой проект, подразумевающий ввод посетителями больших порций текста. Как организовать сам ввод текста? Тег `<TEXTAREA>` уже давно не соответствует требованиям современных сайтов, поскольку не позволяет форматировать текст. Можно, конечно, разрешить пользователям вводить HTML-теги, вырезая с помощью `strip_tags()` потенциально опасные теги, — точнее, разрешая только безопасные теги. Но, во-первых, далеко не все пользователи знают, что такое HTML. А, во-вторых, такой сайт будет выглядеть архаично на фоне ваших конкурентов, которые уже давно обзавелись визуальными редакторами.

На PHP написано много различных визуальных редакторов, которые без особых проблем можно найти в Интернете. А здесь мы рассмотрим SPAW2 — довольно удобный визуальный редактор, который легко подключить и адаптировать к любому сайту.

В первых двух изданиях этой книги описывалась разработка примитивного визуального редактора. Но наш редактор, действительно, был уж очень примитивным. И я решил, что тратить время на изобретение этого колеса заново весьма нерационально. Лучше употребить его с пользой на создание какой-либо уникальной функции для своего сайта. Тем более, что визуальные редакторы "есть в наличии", и вам остается только выбрать тот, который больше всего понравится.

Во втором издании книги информация о редакторе SPAW2 была приведена в качестве бонуса, а сейчас мы рассмотрим его подробнее. Этот редактор позволяет редактировать как визуальное представление, так и HTML-код текста, вставлять ссылки, таблицы, картинки, создавать списки и выполнять другие действия по форматированию текста. На рис. 49.1 показан редактор в действии, а на рис. 49.2 представлен процесс выбора картинки.

Использовать этот редактор очень просто. Скачайте с сайта www.spaweditor.com архив с редактором, распакуйте его и загрузите в любой каталог на вашем

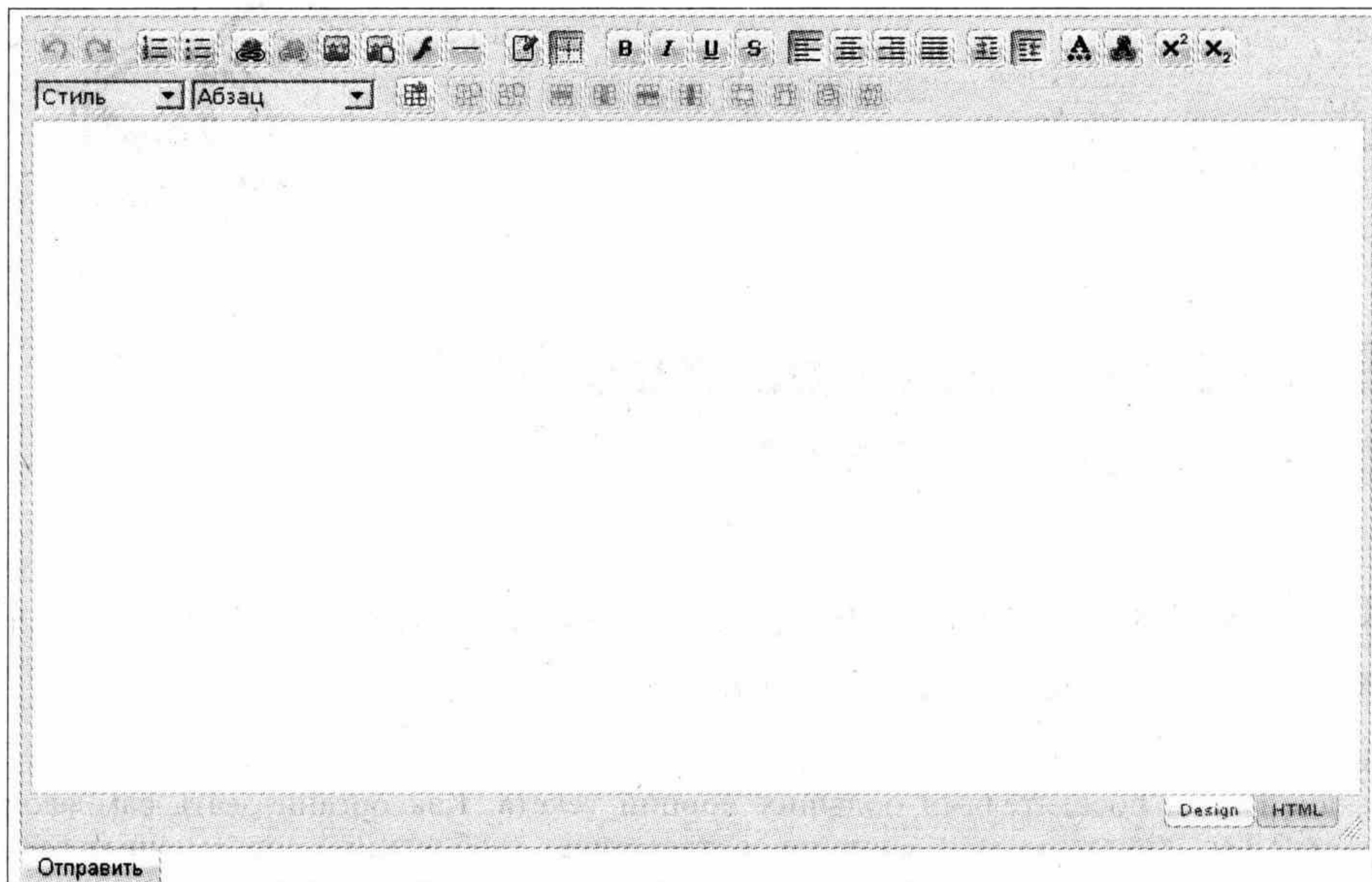


Рис. 49.1. Редактор SPAW2 в действии

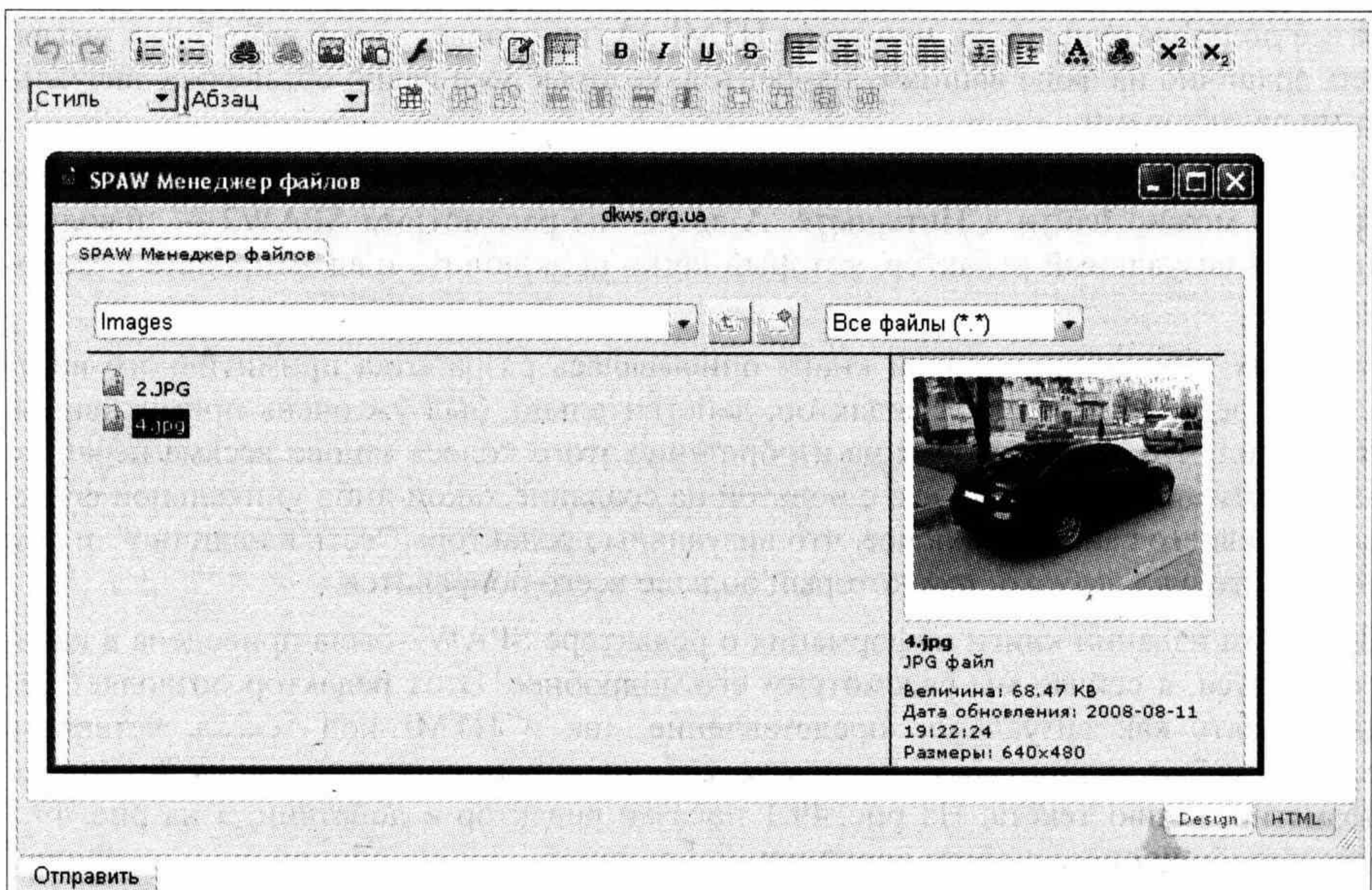


Рис. 49.2. Выбор картинки

веб-сервере — например, в каталог `spaw2`. Перейдите в каталог `spaw2/config` и откройте файл `config.php`. В этом конфигурационном файле необходимо отредактировать некоторые параметры:

```
// по умолчанию en, вам нужно установить ru для русификации редактора
SpawConfig::setStaticConfigItem('default_lang','ru');

// ширина
SpawConfig::setStaticConfigItem('default_width','100%');
// высота
SpawConfig::setStaticConfigItem('default_height','400px');

// кодировка вывода по умолчанию
SpawConfig::setStaticConfigItem('default_output_charset','windows-1251');

// каталоги для загрузки флешек, картинок и файлов:
// directories
SpawConfig::setStaticConfigItem(
    'PG_SPAWFM_DIRECTORIES',
    array(
        array(
            'dir' =>
                SpawConfig::getStaticConfigValue('SPAW_DIR').'uploads/flash/',
            'caption' => 'Flash movies',
            'params' => array(
                'allowed_filetypes' => array('flash')
            )
        ),
        array(
            'dir' =>
                SpawConfig::getStaticConfigValue('SPAW_DIR').'uploads/images/',
            'caption' => 'Images',
            'params' => array(
                'default_dir' => true, // set directory as default (optional setting)
                'allowed_filetypes' => array('images')
            )
        ),
        array(
            'dir' =>
                SpawConfig::getStaticConfigValue('SPAW_DIR').'uploads/files/',
            'fsdir' =>
                SpawConfig::getStaticConfigValue('SPAW_ROOT').'uploads/files/', // optional
                absolute physical filesystem path
            'caption' => 'Files',
            'params' => array(
                'allowed_filetypes' => array('any')
            )
        )
    )
);
```



```

    ),
    ),
    SPAW_CFG_TRANSFER_SECURE
);

```

Не забудьте установить права доступа 777 к каталогам uploads, uploads/flash, uploads/images и uploads/files:

```

chmod 777 uploads
chmod 777 uploads/flash
chmod 777 uploads/images
chmod 777 uploads/files

```

Теперь перейдите в каталог demo и откройте файл demo.php. Сейчас мы его немного изменим. Суть заключается в том, чтобы создать форму, передающую текст, введенный пользователем, другому сценарию. Ранее мы использовали тег <TEXTAREA> для создания области ввода текста. Теперь вместо нее мы вызовем редактор SPAW2, который заменит эту область. В нашем случае мы будем передавать текст (точнее, HTML-код), введенный пользователем, с помощью SPAW2 файлу index.php. В листинге 49.1 представлен код измененного файла demo.php.

Листинг 49.1. Файл demo.php

```

<?php
// подключение основного include-файла
include("../spaw.inc.php");
$spaw1 = new SpawEditor("spaw1");
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8">

  </head>
  <body>
</a><p>
  <form method="post" action=index.php>
    <?php
    $spaw1->show();
    ?>
  <input type=submit>
  </form>
  </body>
</html>

```

Весь имеющийся здесь HTML-код — это форма для передачи текста. К SPAW2 относятся три следующих оператора:


```
// подключаем главный файл
include("../spaw.inc.php");
// создаем экземпляр редактора
$spaw1 = new SpawEditor("spaw1");
// вызываем редактор
$spaw1->show();
```

После вызова редактора пользователь может редактировать текст, а когда он нажмет кнопку **Отправить**, введенный текст будет передан сценарию `index.php`, состоящему всего из одной строки:

```
<? echo $_POST['spaw1']; ?>
```

В этом файле нет никакой премудрости — он просто выведет текст, введенный пользователем в браузер (текст передается как POST-переменная `spaw1`).

49.2. Адаптация редактора к вашему сайту

Как мы уже убедились, подключить и использовать редактор довольно просто. Сейчас мы рассмотрим несколько примеров практического использования этого редактора.

49.2.1. Загрузка в редактор текста статьи

Не всегда нужно выводить пустое окно редактора. Да, когда вы с нуля создаете статью или запись в блоге, надо отобразить его пустое окно. Но когда редактируется запись, следует отобразить редактор с загруженным в него текстом записи. В этом случае вызывать редактор нужно так:

```
// подключаем include-файл
include("spaw2/spaw.inc.php");
// создаем редактор (POST-переменная article_text) и загружаем
// в него текст из $row[post_html]
$spaw = new SpawEditor("article_text", $row[post_html]);
// отображаем редактор
$spaw->show();
```

49.2.2. Вывод редактора в TPL-шаблоне

Некоторая проблема возникает из-за того, что метод `show()` отображает сразу весь код редактора (HTML/JavaScript), что не совсем хорошо для проектов, в которых HTML-код не выводится PHP-сценарием прямо в браузер, а используются TPL-шаблоны.

Не беда. Мы можем методом `getHtml()` получить весь необходимый для отображения редактора HTML-код, а затем назначить его переменной TPL-шаблона, чтобы отобразить редактор в нужном месте сайта.

Код будет примерно таким:

```
$spaw = new SpawEditor("article_html", $row[txt]);  
// получаем код редактора в переменную $editor_code  
$editor_code = $spaw->getHtml();  
...  
// устанавливаем smarty-переменную editor  
$smarty->assign('editor', $editor_code);
```

Полный синтаксис создания редактора следующий:

```
$spaw = new SpawEditor(имя_переменной, значение,  
язык, режим панели инструментов, тема,  
ширина, высота, CSS-файл);
```

49.2.3. Запрет изменения размера редактора

Редактор позволяет пользователю изменять размер своего окна — для этого нужно потянуть за нижний правый угол окна редактора. Но такой функционал нередко нарушает дизайн сайта, что не есть хорошо. Поэтому нужно заранее установить допустимые размеры редактора (что было показано ранее) и запретить изменение размера.

Опции, отвечающей за запрет изменения размера окна редактора в процессе его выполнения, не существует, поэтому мне пришлось отредактировать файл `js/common/editor.js`. Из него я удалил следующие строки:

```
// resizing control  
SpawEditor.prototype.isResizing = false;  
SpawEditor.prototype.isVerticalResizingAllowed = function()  
{  
    var res = this.getConfigValue("resizing_directions");  
    res = res?res.toLowerCase():res;  
    if (res == 'vertical' || res == 'both')  
        return true;  
    else  
        return false;  
}  
SpawEditor.prototype.isHorizontalResizingAllowed = function()  
{  
    var res = this.getConfigValue("resizing_directions");  
    res = res?res.toLowerCase():res;  
    if (res == 'horizontal' || res == 'both')  
        return true;  
    else  
        return false;  
}  
SpawEditor.prototype.resizingGripMouseDown = function(event)  
{  
    this.lastMousePosX = event.clientX;  
    this.lastMousePosY = event.clientY;
```



```

this.isResizing = true;
SpawEngine.resizingEditor = this;

//SpawEngine.resizingEditor.hidePage(SpawEngine.resizingEditor.
                                                                    getActivePage());

// prevent gecko from dragging image
if (event.preventDefault)
    event.preventDefault();
}
SpawEditor.prototype.finalizeResizing = function()
{
    var resobj = this.isInDesignMode()?this.getPageIframeObject(
        this.getActivePage().name):this.getPageInput(this.getActivePage().name);
    for (var i=0; i<this.pages.length; i++)
    {
        var pif = this.getPageIframeObject(this.pages[i].name);
        var pta = this.getPageInput(this.pages[i].name);
        pif.style.height = resobj.offsetHeight + 'px';
        pta.style.height = resobj.offsetHeight + 'px';
        pta.style.width = resobj.offsetWidth + 'px';
    }
}

```

49.2.4. Изменение размеров редактора "на лету" — в процессе выполнения сценария

Вы можете изменять размеры редактора в процессе выполнения сценария — например, при добавлении комментария будет показана маленькая версия редактора, а при редактировании/добавлении записи — полноразмерная.

Изменить размер редактора можно так:

```

$spaw = new SpawEditor("text");
SpawConfig::setStaticConfigValue('default_height','200px');
SpawConfig::setStaticConfigValue('default_width','240px');
$spaw->show();

```

В этом случае высота редактора равна 200 пикселям, ширина — 240 пикселям. Также можно задать размер редактора при его создании:

```

$spaw = new SpawEditor('text','value','ru','mini','', '240px', '200px', '');

```

49.2.5. Панель инструментов редактора

Далеко не все кнопки на панели инструментов редактора нужны всем пользователям. Вы можете вызвать редактор с сокращенной версией панели инструментов — например, когда требуется отобразить форму быстрого ответа, и полный функционал не нужен.

Определено три предустановки панели инструментов:

- ☐ `standard` — стандартная (такую вы видите по умолчанию);
- ☐ `mini` — минимальная (компактная);
- ☐ `all` — панель инструментов со всеми кнопками.

Предустановка панели управления определяется при создании редактора.

Например, сейчас мы выберем компактный вид редактора:

```
$spawl = new SpawEditor('text','', 'ru', 'mini', '', '350px', '200px', '');
```

Но мы на этом не остановимся и продолжим исследование редактора далее. Допустим, мы хотим создать собственную панель инструментов с определенным набором кнопок. Откройте файл `config/config.php` и найдите в нем код:

```
SpawConfig::setStaticConfigItem('toolbarset_mini',  
    array(  
        "format" => "format_mini",  
        "edit" => "edit",  
        "tools" => "tools"  
    )  
);
```

Этот код определяет, какие кнопки мы увидим на компактной панели инструментов. В данном случае на ней будут размещены следующие кнопки: `format_mini` — кнопки группы форматирования (обычный набор называется `format`), `edit` — кнопки редактирования текста и `tools` — различные утилиты.

Теперь попробуем создать собственную панель инструментов с кнопками следующих групп:

- ☐ кнопки выравнивания — но не все, а только по левому краю и по центру;
- ☐ кнопки форматирования — но не стандартный набор, а такой, в котором нет кнопки зачеркивания текста;
- ☐ кнопки вставки ссылки и картинки, а также кнопка, открывающая окно параметров картинки;
- ☐ кнопки отмены и повтора действий — но без остальных кнопок из набора `edit`;
- ☐ кнопки списков (нумерованного и маркированного).

Может, кому-то именно такая панель инструментов и покажется странной, но она создается в демонстрационных (учебных) целях, чтобы показать вам, как размещать на панели не весь набор кнопок, а только определенные кнопки.

Итак, сформируем пользовательскую панель инструментов (назовем ее `custom`). Для этого создадим в каталоге `plugins/core/lib/toolbars/` файл `custom.toolbar.php`. В этом файле будут описаны кнопки, которые расположены на панели инструментов `custom` (листинг 49.2).

Listing 49.2. Файл custom_toolbar.php

```
<?php
$items = array
(
/* Жирный */
new SpawTbButton("core", "bold", "isStandardFunctionEnabled",
"isStandardFunctionPushed", "standardFunctionClick"),
/* Курсив */
new SpawTbButton("core", "italic", "isStandardFunctionEnabled",
"isStandardFunctionPushed", "standardFunctionClick"),
/* Подчеркнутый */
new SpawTbButton("core", "underline", "isStandardFunctionEnabled",
"isStandardFunctionPushed", "standardFunctionClick"),
/* Разделитель */
new SpawTbImage("core", "separator"),
/* По левому краю */
new SpawTbButton("core", "justifyleft", "isStandardFunctionEnabled",
"isStandardFunctionPushed", "standardFunctionClick", SPAW_AGENT_ALL, true),
/* По центру */
new SpawTbButton("core", "justifycenter", "isStandardFunctionEnabled",
"isStandardFunctionPushed", "standardFunctionClick", SPAW_AGENT_ALL, true),
/* Разделитель */
new SpawTbImage("core", "separator"),
/* Отменить */
new SpawTbButton("core", "undo", "isStandardFunctionEnabled", "",
"standardFunctionClick"),
/* Повторить */
new SpawTbButton("core", "redo", "isStandardFunctionEnabled", "",
"standardFunctionClick"),
/* Разделитель */
new SpawTbImage("core", "separator"),
/* Нумерованный список */
new SpawTbButton("core", "insertorderedlist", "isStandardFunctionEnabled",
"isStandardFunctionPushed", "standardFunctionClick", SPAW_AGENT_ALL, true),
/* Маркированный список */
new SpawTbButton("core", "insertunorderedlist", "isStandardFunctionEnabled",
"isStandardFunctionPushed", "standardFunctionClick", SPAW_AGENT_ALL, true),
/* Вставка ссылки */
new SpawTbButton("core", "hyperlink", "isInDesignMode", "", "hyperlinkClick",
SPAW_AGENT_ALL, true),
/* Удаление ссылки */
new SpawTbButton("core", "unlink", "isStandardFunctionEnabled", "",
"standardFunctionClick", SPAW_AGENT_ALL, true),
/* Вставка картинки */
new SpawTbButton("core", "image", "isInDesignMode", "", "imageClick",
SPAW_AGENT_ALL, true),
```



```
/* Свойства картинки */
```

```
new SpawTbButton("core", "image_prop", "isInDesignMode", "", "imagePropClick",
SPAW_AGENT_ALL, true)
);
?>
```

Описание остальных кнопок можно найти в файлах папки `plugins/core/lib/toolbars/`. Так, в файле `format.toolbar.php` описаны кнопки панели управления форматированием.

Созданием файла `custom.toolbar.php` нам не обойтись. Нужно также прописать нашу панель в файле `config.php`. При желании можно создать код, отвечающий за создание абсолютно нового набора, но проще отредактировать набор `mini`, чтобы он выводил только нашу панель:

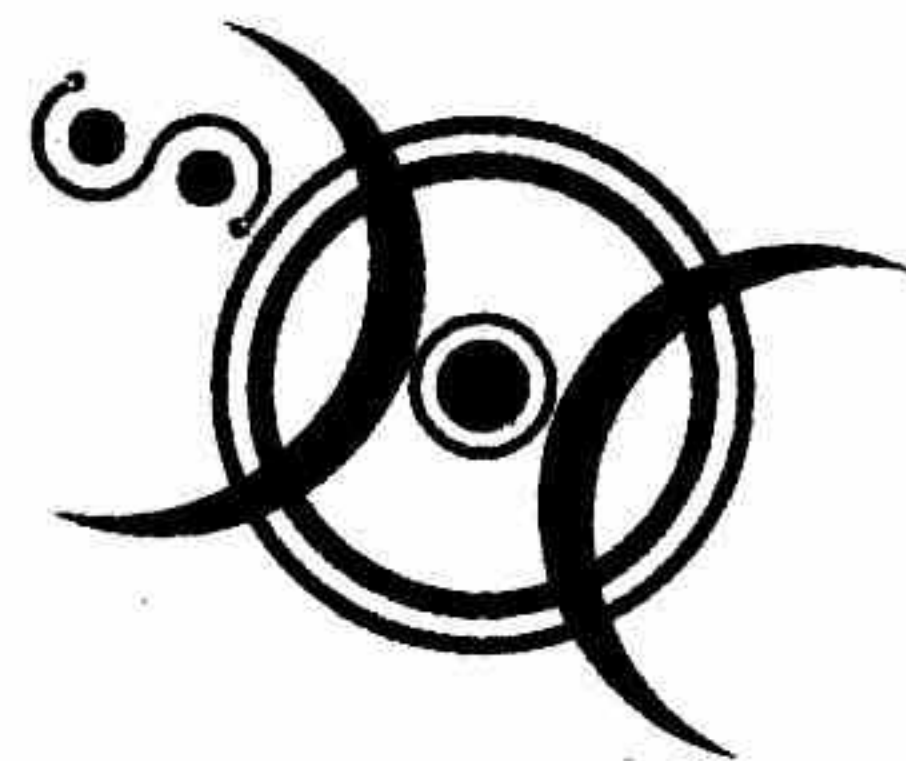
```
SpawConfig::setStaticConfigItem('toolbarset_mini',
array(
"format" => "custom"
)
);
```

Собственно, потом можно будет выбрать новую панель инструментов, как было показано ранее.

49.2.6. Ошибка Error 105 (net::ERR_NAME_NOT_RESOLVED)

В некоторых случаях вы можете столкнуться с подобной ошибкой. Для ее устранения нужно открыть конфигурационный файл `config/config.php` и раскомментировать следующие строки:

```
SpawConfig::setStaticConfigItem('SPAW_DIR', '/spaw2/');
// and the following settings will be calculated automaticly
SpawConfig::setStaticConfigItem('SPAW_ROOT',
str_replace("\\", "/", realpath(dirname(__FILE__)."/..").'/');
SpawConfig::setStaticConfigItem('DOCUMENT_ROOT',
substr(SpawConfig::getStaticConfigValue('SPAW_ROOT'), 0, strlen(SpawConfig::
getStaticConfigValue('SPAW_ROOT')) -
strlen(SpawConfig::getStaticConfigValue('SPAW_DIR'))));
```

ГЛАВА 50

WAP-сервер средствами PHP

50.1. Есть ли необходимость в WAP?

Мобильные телефоны уже давно перестали быть просто "звонилками", они превратились в компьютеры, мало чем уступающие обычным компьютерам при повседневном применении. С мобильного телефона (смартфона) можно бороздить просторы Интернета, отправлять почту, делать с его помощью фотографии, снимать видео, играть в игры, графика которых скоро сравняется с графикой игр для настольных компьютеров. Не верите? Тогда взгляните на игру Shadowgun для планшетов на базе Android.

С любого современного телефона можно зайти на самый обычный веб-сайт. Но, учитывая размер экрана телефона и то, что не все телефоны поддерживают альбомное отображение страницы (при повороте телефона), работать с таким сайтом не очень удобно. Просмотр страницы становится затруднен, не говоря уже о полноценной работе.

Если вы хотите, чтобы вашим посетителям было удобно работать с сайтом, следует позаботиться о создании его мобильной версии. Лучше всего для этого подходит протокол WAP (Wireless Application Protocol).

Нужно отметить, что протокол WAP довольно древний — он существовал еще в те времена, когда мобилки не умели заходить на обычные сайты. Для них и создавались мобильные версии страничек. Но ничто не мешает нам создать такую страничку и в наше время.

Не нужно думать, что WAP — мертвый протокол. Наоборот, сейчас есть его усовершенствованная версия WAP 2.0, отличительной особенностью которой является поддержка сокращенных версий XHTML и CSS, что позволяет открывать WAP-сайт на обычном компьютере обычным браузером без всяких плагинов. Первая версия протокола WAP использовала для разметки информации язык WML (Wireless Markup Language). При желании вы можете изучить этот язык разметки — он существенно проще, чем HTML. Проще тем, что в нем нет многочисленных стандартов, и ваша страничка будет одинаково выглядеть на различных устройствах, чего нельзя сказать о страницах на HTML, когда одна и та же страница по-разному отображается в разных браузерах.

Одним словом, протокол WAP будет востребован до тех пор, пока пользователи полностью не перейдут на планшеты или подобные устройства, т. е. пока нельзя комфортно просмотреть всю страничку на экране небольшого мобильного устройства.

В этой главе мы рассмотрим настройку сервера Apache и разработку PHP-сценария, выводящего WML-код, а сам язык WML рассматривать не будем.

50.2. Настройка веб-сервера Apache

Чтобы добавить поддержку WAP в сервер Apache, нужно добавить в конфигурационный файл Apache `httpd.conf` следующие строки:

```
AddType text/vnd.wap.wml .wml
AddType image/vnd.wap.wbmp .wbmp
AddType application/vnd.wap.wmlc .wmlc
AddType text/vnd.wap.wmlscript .wmls
AddType application/vnd.wap.wmlscriptc .wmlsc
CharsetDefault utf-8
CharsetSourceEnc utf-8
```

Осталось только перезагрузить Apache. Желательно для мобильной версии сайта создать отдельный поддомен — например: **wap.firma.ru**.

Если редактировать общий файл конфигурации Apache не желательно (или у вас нет на это прав), то можно отредактировать файл `.htaccess`. Сначала в панели управления хостингом (например, в DirectAdmin — как туда зайти, спрашивайте в службе поддержки хостинг-провайдера) создайте поддомен для мобильной версии сайта. Потом перейдите в каталог, соответствующий поддомену, — он называется так же, как и поддомен, — и создайте в нем файл `.htaccess` следующего содержания (листинг 50.1).

Листинг 50.1. Файл `.htaccess` (поддержка WAP)

```
DirectoryIndex index.wml
AddType text/vnd.wap.wml wml
AddType application/vnd.wap.wmlc wmlc
AddType text/vnd.wap.wmlscript wmls
AddType application/vnd.wap.wmlscriptc wmlsc
AddType image/vnd.wap.wbmp wbmp
CharsetDefault utf-8
CharsetSourceEnc utf-8
```

Настройка сервера завершена, теперь нужно протестировать поддержку WAP. Создайте файл `index.wml` (листинг 50.2) и попробуйте обратиться к вашему WAP-серверу с телефона. Обратите внимание — WML-файлы нужно создавать в кодировке UTF-8, если вы не хотите проблем с отображением кириллицы.

Листинг 50.2. Файл index.wml

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="main" title="Мобильная версия сайта">
    <p>Добро пожаловать на WAP-сайт</p>
    <p><a href="#page2">Вторая страница</a></p>
</card>
<card id="page2" title="Вторая страница">
    <p>Просто вторая страница</p>
    <p><a href="#main">На главную</a></p>
</card>
</WML>
```

Итак, у нас есть тестовая страница. Если она отобразилась в вашем телефоне, то вы все сделали правильно. Осталось только выучить язык WML, но, к сожалению, его рассмотрение выходит за рамки этой книги.

50.3. PHP-сценарий

Теперь дело дошло до PHP-сценария. Переименуйте файл index.wml в main.wml и создайте PHP-сценарий, приведенный в листинге 50.3.

Листинг 50.3. PHP-сценарий с поддержкой WAP

```
<?php
Header("Content-Type: text/vnd.wap.wml");

echo file_get_contents('main.wml')
?>
```

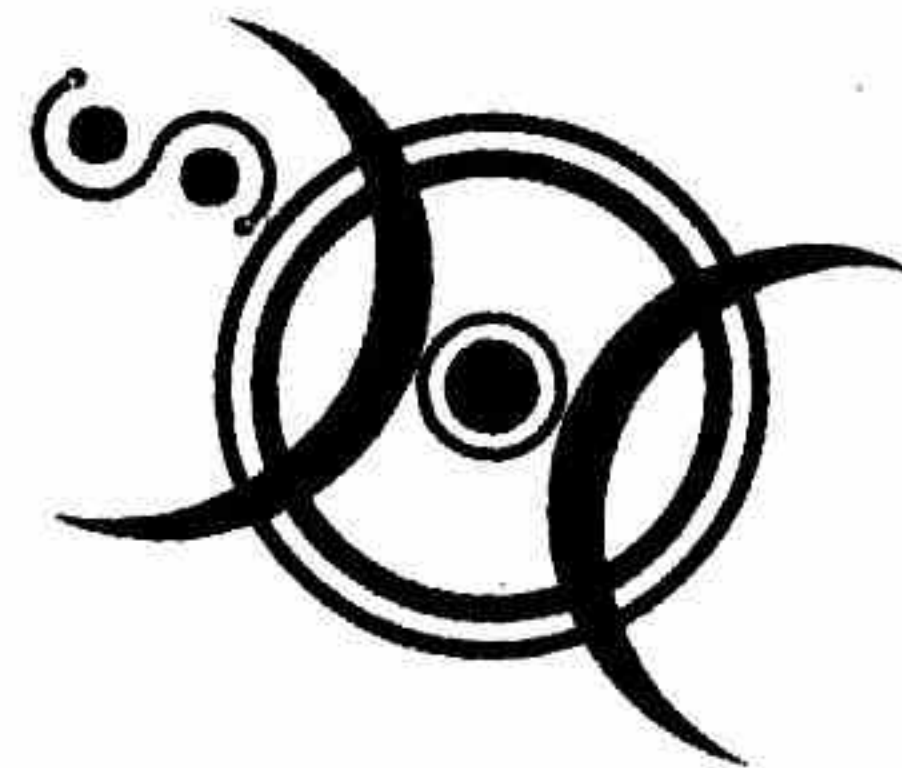
Самое главное — вывести следующий заголовок:

```
Header("Content-Type: text/vnd.wap.wml");
```

После этого можно выводить WML-код, который в нашем случае хранится в файле main.wml.

Как видите, ничего сложного здесь нет. При желании вообще можно написать универсальный сценарий, передающий или мобильную, или полноценную версию кода сайта клиенту. Для этого следует проанализировать переменные окружения HTTP_ACCEPT (список MIME-типов поддерживаемых браузером данных) и HTTP_USER_AGENT (браузер пользователя) и на основании значений этих переменных отобразить или WML-код, или полноценный HTML-код.

ГЛАВА 51



Пишем коммерческий сценарий

51.1. Zend Guard: за и против

Вы — профессиональный разработчик PHP. Предположим, что вы написали довольно удобный движок для сайта, т. е. систему управления контентом сайта (CMS). Понятно, что вам хочется получать за свой труд "дивиденды", поэтому вы выложили в Интернет демо-версию этого продукта, позволяющую ознакомиться с ним всем заинтересовавшимся. Демо-версия была скопирована несколькими пользователями. Она им понравилась. Кого-то возможности демо-версии вполне устроили, а кто-то захотел большего. И этот кто-то делает первый заказ и перечисляет в ваш кошелек WebMoney первые 50 долларов (100, 200, 1000 — сейчас не важно). Вы отправляете ему по электронной почте полную версию своей CMS. Потом последовал второй заказ, затем — третий. Вроде бы все развивается хорошо.

Но спустя некоторое время вы обнаружили, что заказы поступать перестали. А в Сети, как грибы после дождя, начали появляться сайты, использующие вашу полную версию. В чем дело, вы уже догадались. Кто-то "поделиться" с Интернетом своей полной версией или кто-то наглým образом просто ее украл (давайте называть вещи своими именами) у законного владельца.

Понятно, что о вознаграждении за ваш труд можно будет забыть — ведь если где-то в Интернете есть полная версия CMS, которую можно скачать бесплатно (ну и что, что она пиратская — у нас пока к этому относятся не так строго, как на Западе), то никто и не подумает ее у вас покупать. Не нужно строго судить людей — таков наш менталитет, никуда от этого не денешься. Должно пройти время, чтобы пользователи перешли на лицензионные версии.

Что же делать? Забыть о своем проекте? Нет, нужно собраться с силами и выпустить новую версию, которая будет обладать лучшим функционалом. Но ведь история может повториться снова. Вы опять потратите время, получите деньги за пару заказов, и на этом ваш бизнес прекратится — кто-то опять вашу новинку украдет и начнет распространять бесплатно.

Чтобы этого не случилось, нужно свой код защитить. Можно защитить его финансово, а можно технически. Финансовая защита заключается в назначении за ваш

движок максимальной цены. Да, вы получите в десять раз больше, но у вас окажется всего один клиент. Что ж, это, во-первых, позволит лучше отслеживать утечку кода — у вас будет всего один клиент, а не десять, поэтому отследить, кто именно выложил пиратскую версию, будет проще. А во-вторых, если кто-то отдаст за ваш движок пару тысяч долларов, то не думаю, что этот кто-то станет делиться своим приобретением с другими пользователями. Опять же сработает менталитет. Лично я никому бы не подарил программное обеспечение на пару тысяч долларов, за которые отдал свои кровные.

А тот пират, что отдал 50 долларов за вашу CMS, а потом выложил ее на общедоступном сервере, будет уже просто не в состоянии заплатить 2000 долларов за новую версию. Может денег у него и хватит, но не думаю, что он потратит их на такую благотворительность. Обычно финансовая версия работает. Но все же есть вероятность того, что архив с вашей CMS опять окажется на общедоступном интернет-сервере. Как? Например, вашу разработку купит организация, а ее недобросовестный администратор возьмет, да и выложит на своем личном сервере архив с вашей CMS, — ему ведь все равно, деньги-то платил не он... Это так называемый человеческий фактор.

Полностью избежать несанкционированного копирования и изменения вашего кода позволяют продукты типа Zend Guard. Программа Zend Guard кодирует ваш PHP-код в байт-код, в результате чего никто, кроме вас, не сможет изменить PHP-код, хотя он будет нормально выполняться интерпретатором PHP. Должен отметить, что это идеальная защита от несанкционированного распространения вашего PHP-кода. Дело в том, что в самом коде вы лицензируете CMS для определенного домена, и если домен, на котором запущена система, отличается от лицензированного, код не будет работать.

Да, кто-то по-прежнему сможет украсть закодированную версию движка. Но изменить код, а значит, и домен, для которого лицензирована украденная копия движка, пират не сможет.

Работает Zend Guard просто превосходно. Но, как всегда, есть одно "но". Существенным, хоть и единственным недостатком Zend Guard является его цена. Вам придется отдать за Zend Guard 600 долларов... в год. Тут надо считать — если вы на своем продукте зарабатываете в год больше 600 долларов (хотя бы в два раза), то имеет смысл приобрести Zend Guard. Если же ваш доход меньше, о Zend Guard вам придется забыть.

Дорого?

Это еще дешево, ведь четвертая версия Zend Guard стоила 995 долларов! Почему цену снизили? Видимо, для повышения привлекательности продукта.

Далее мы рассмотрим, как использовать Zend Guard. А вы уже сами решите, нужен он вам или нет. Последние версии Zend Guard для Linux и Windows всегда можно скачать с www.zend.com.

51.2. Установка Zend Guard

В Windows вам нужно просто запустить файл `ZendGuard-5_5_0.exe` (рис. 51.1), а в Linux — сначала распаковать архив `ZendGuard-5_5_0.tar.gz`, а потом — запустить файл `ZendGuard-4_0_1.bin`.

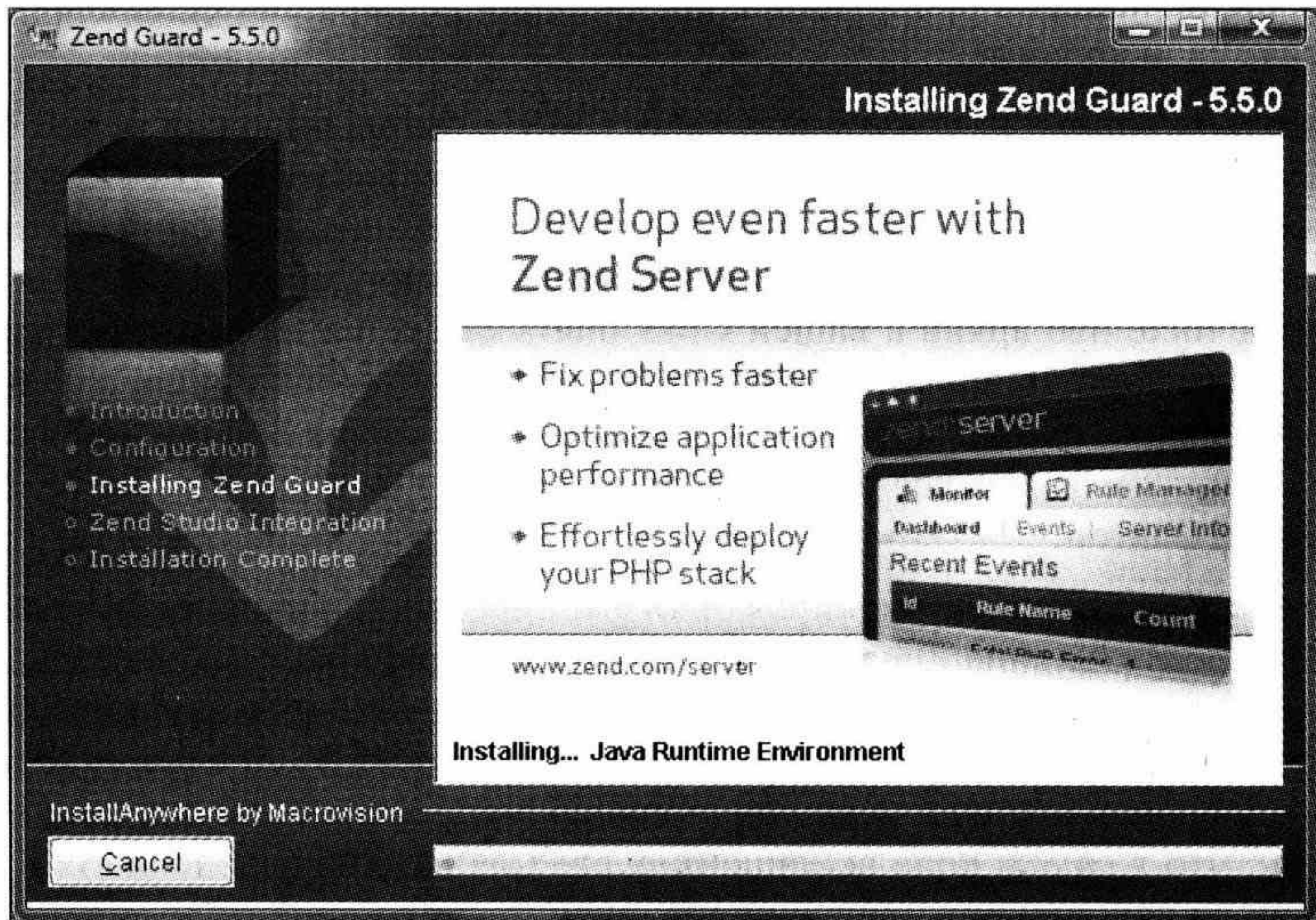


Рис. 51.1. Установка Zend Guard

При установке проблем обычно не возникает. Можно сделать лишь пару замечаний:

- ☐ перед установкой лучше закрыть все приложения;
- ☐ в процессе установки инсталлятор спросит вас, хотите ли вы интегрировать Zend Guard с Zend Studio (если она у вас установлена). Если вы установили Zend Studio (кстати, профессиональная версия стоит "всего" 299 долларов), то не стоит отказываться от этой возможности.

51.3. Быстрая защита сценария

Пробная (trial) версия Zend Guard работает 14 дней, а генерируемые ею лицензии — всего три дня, впрочем, этого вполне достаточно для ознакомления с продуктом.

Попробуем зашифровать PHP-проект. Выберите в основном окне Zend Guard (рис. 51.2) команду меню **File | New | Zend Guard Project**. Введите название проекта, версию и каталог (**Output Location**), в котором будут храниться результирующие (зашифрованные) файлы (рис. 51.3).

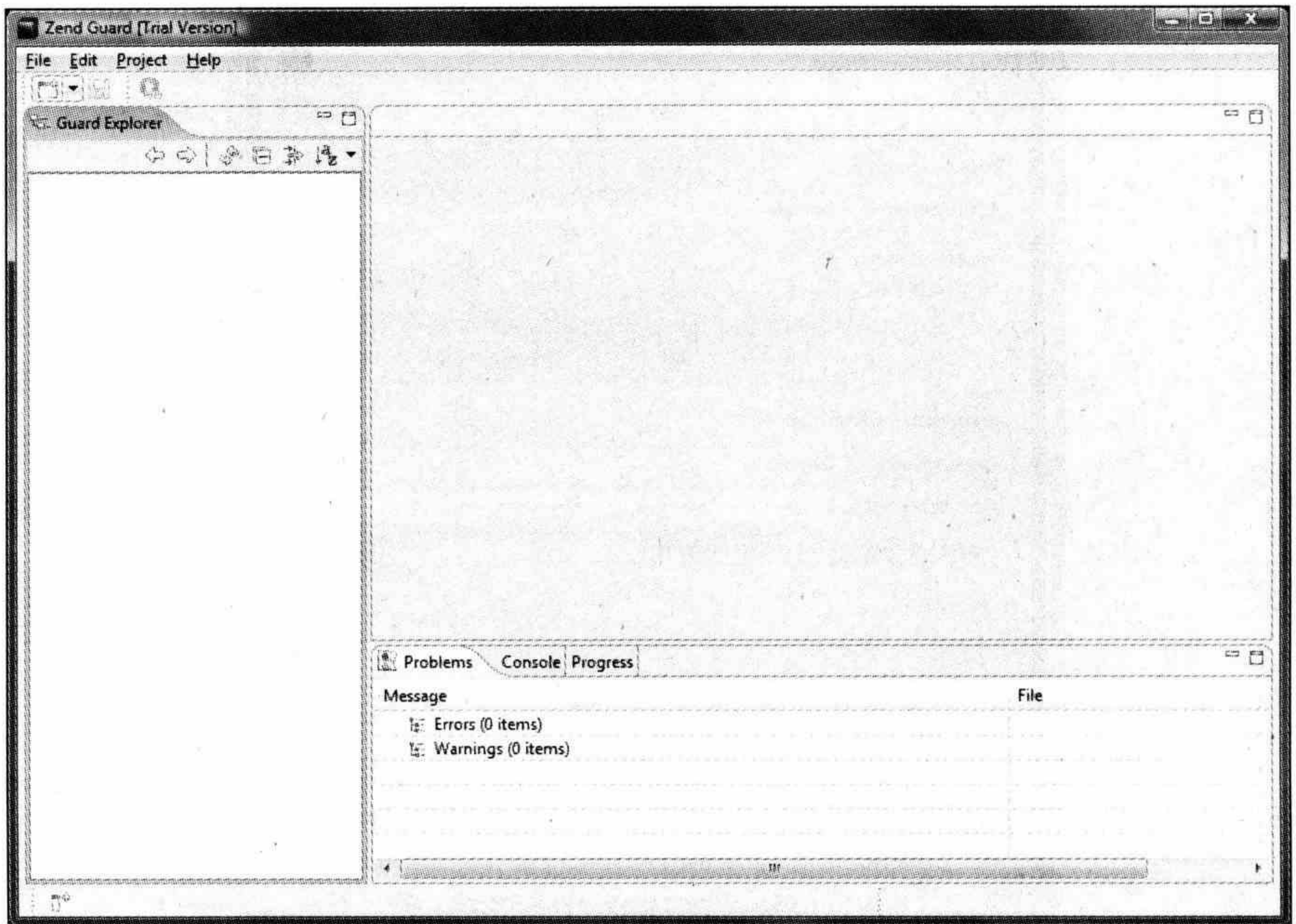


Рис. 51.2. Основное окно Zend Guard

Далее нужно добавить файлы в наш проект (рис. 51.4). Кнопка **Add Folder** добавляет папку, кнопка **Add File** — файл. Наш пробный проект будет состоять из одного файла `example.php`:

```
<?php
$name = 'Denis';
echo "Hello $name";
?>
```

В общем-то можно уже нажать кнопку **Finish**, а можно нажать кнопку **Next** и установить дополнительные параметры проекта (рис. 51.5), в частности:

- ☐ **PHP Version** — версия PHP;
- ☐ **Short Tag Support** — поддержка короткого тега PHP;
- ☐ **Files to encode** — расширения файлов, которые нужно зашифровать (шифровать следует не все файлы, а только файлы, содержащие PHP-код).

Вид основного окна Zend Guard после установки всех параметров проекта показан на рис. 51.6.

Осталось выбрать команду **Project | Encode**, и проект будет закодирован. Откройте закодированный файл (он помещен в каталог, заданный параметром **Output Location**). Как видите (рис. 51.7), код зашифрованного файла осмысленному прочтению на поддается.

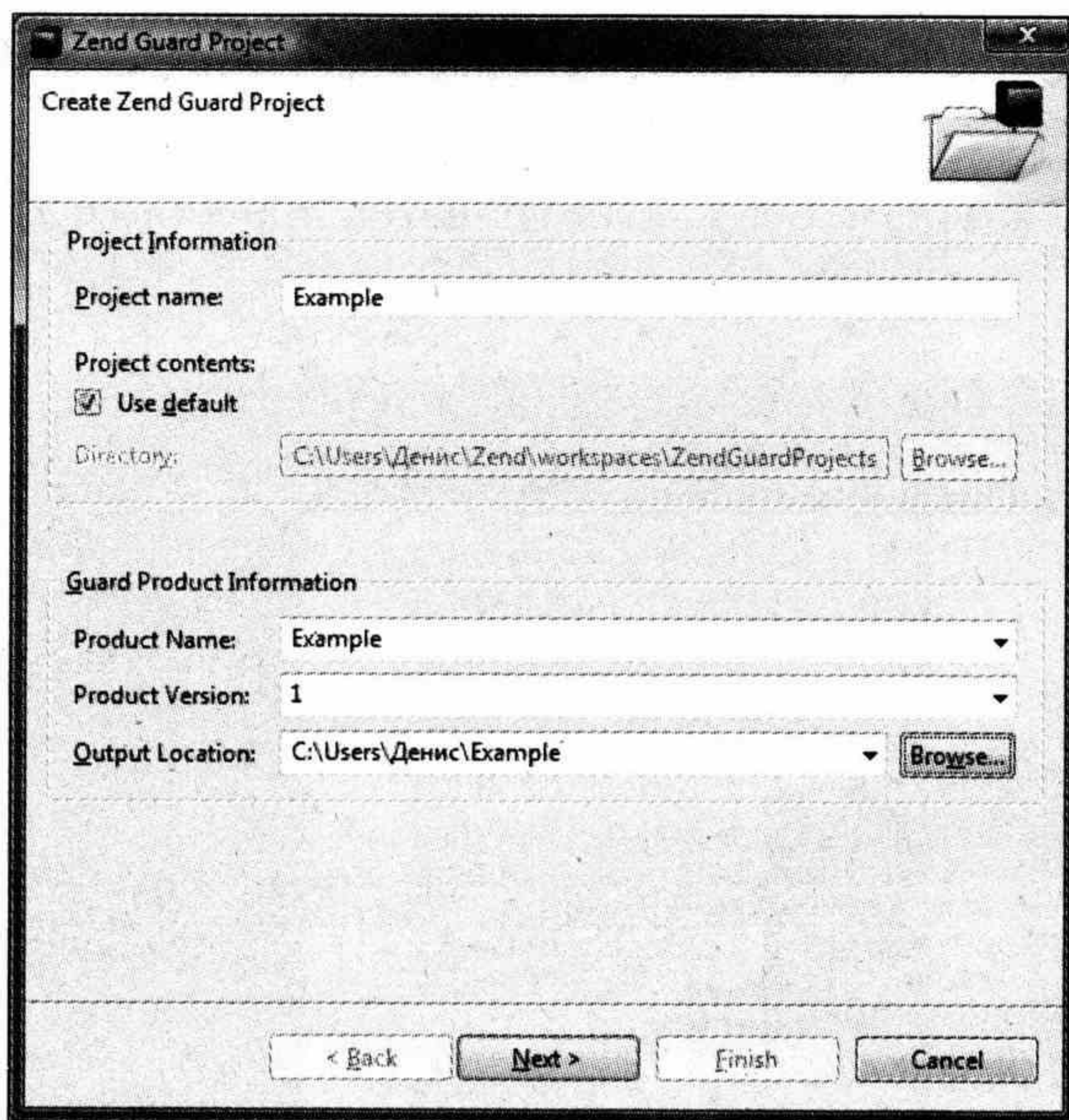


Рис. 51.3. Создание нового проекта: шаг 1

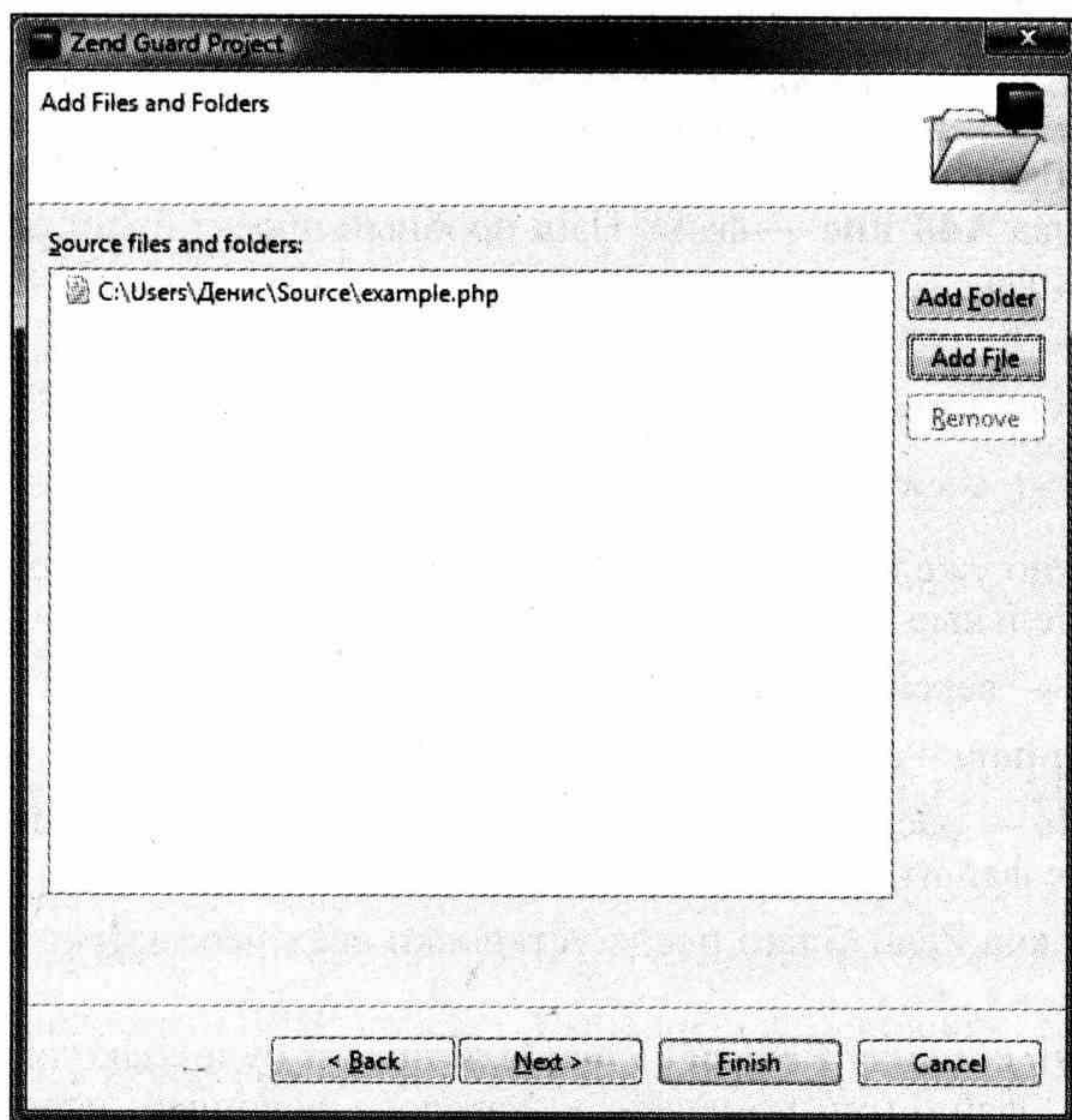


Рис. 51.4. Создание нового проекта: шаг 2

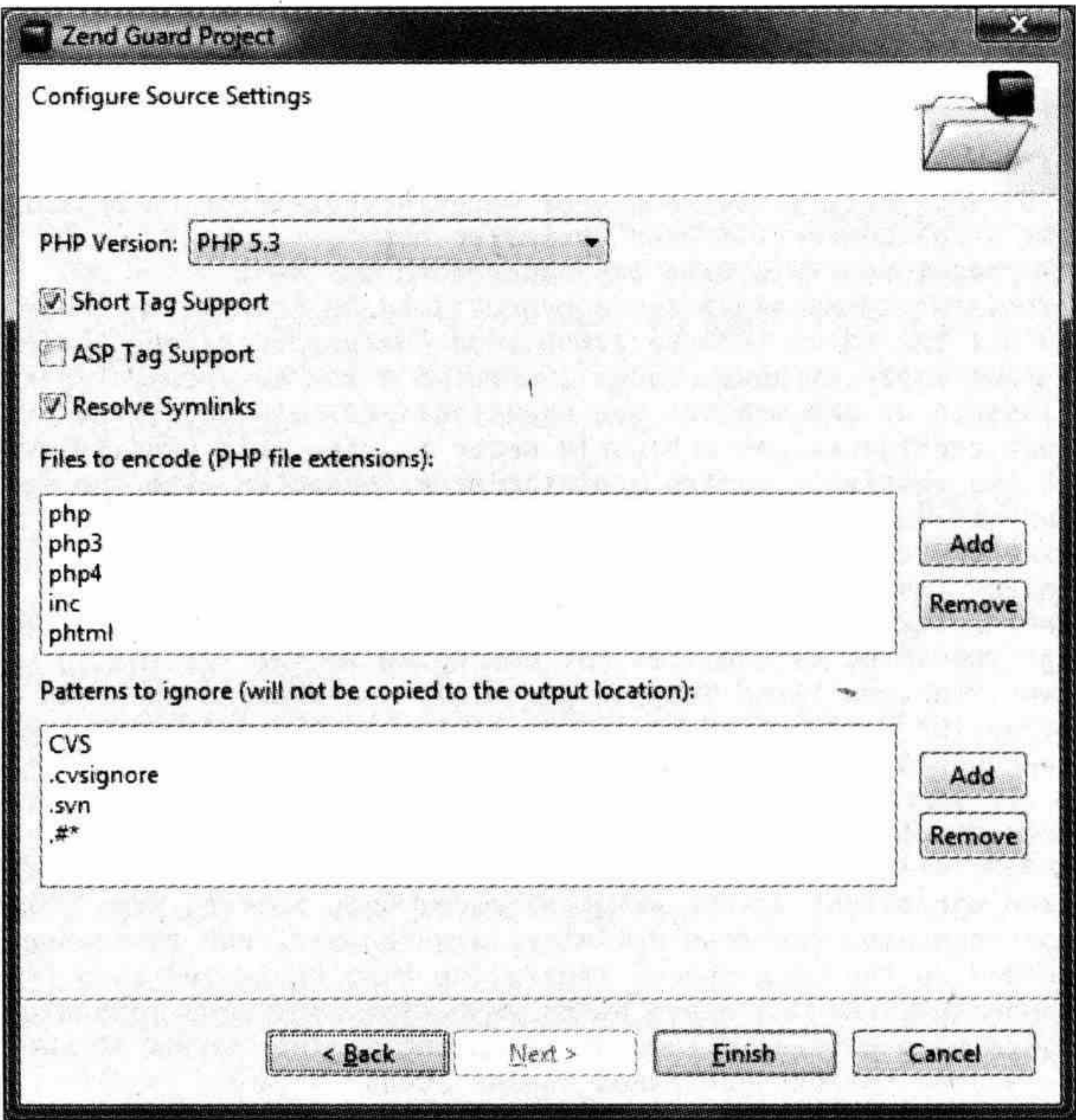


Рис. 51.5. Создание нового проекта: шаг 3

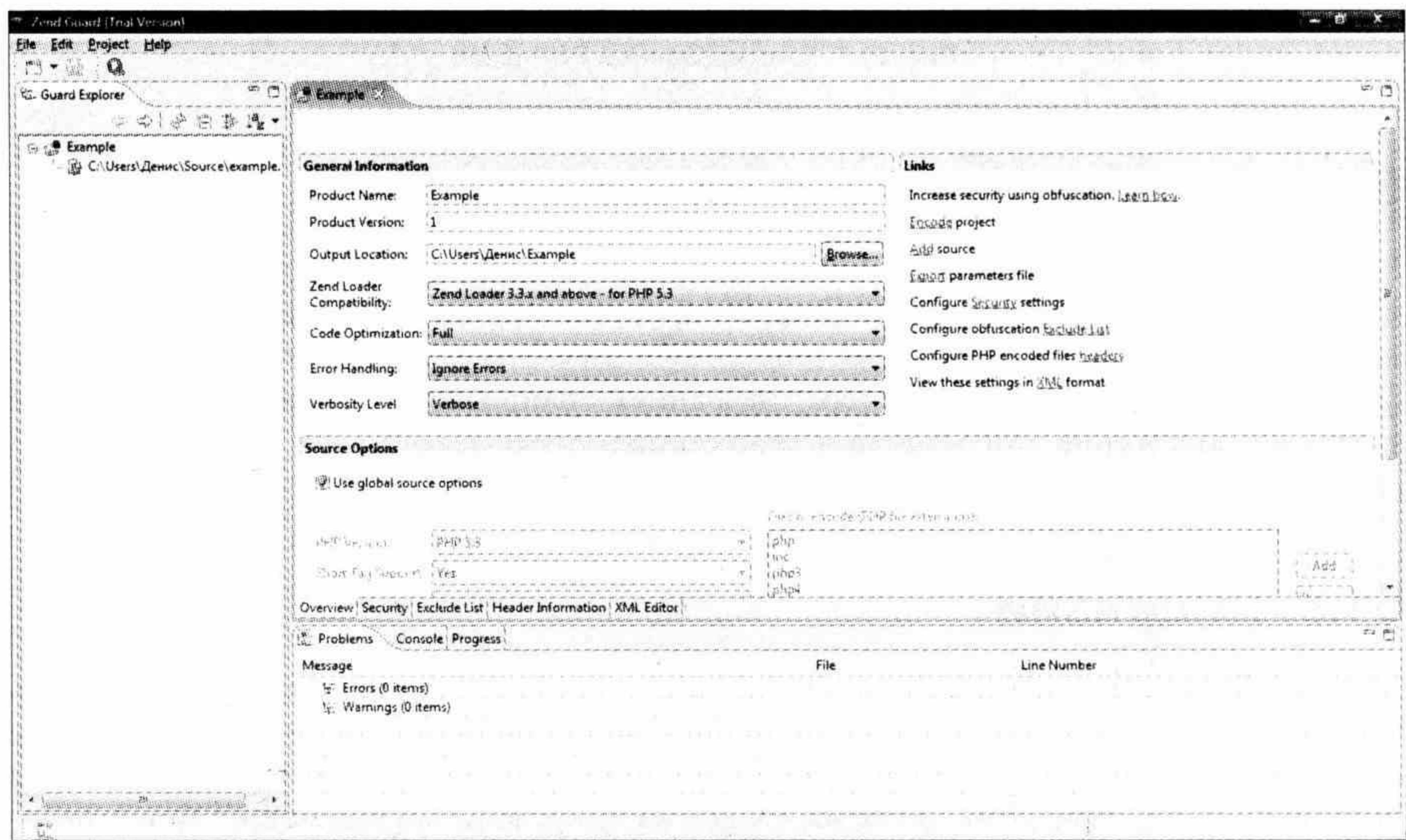


Рис. 51.6. Проект создан


```

<?php @Zend;
4147;
/* ЪЪc!This is not a text file!ЮнЪ */
print <<<EOM
<html><body><a href="http://www.zend.com/store/products/zend-safeguard-suite.php"
></a><center><h1>Zend Optimizer not
installed</h1></center><p>This file was encoded by the <a
href="http://www.zend.com/products/zend_guard">Zend Guard</a>. In order to run
it, please install the <a href="http://www.zend.com/products/zend_optimizer">Zend
Optimizer</a> (available without charge), version 3.0.0 or later. </p><h2>Seeing
this message instead of the website you expected?</h2>This means that this
webserver is not configured correctly. In order to view this website properly,
please contact the website's system administrator/webmaster with the following
message:<br><br><tt>The component "Zend Optimizer" is not installed on the Web
Server and therefore cannot service encoded files. Please download and install
the Zend Optimizer (available without charge) on the Web
Server.</tt><br><br><b>Note</b>: Zend Technologies cannot resolve issues related
to this message appearing on websites not belonging to <a
href="http://www.zend.com">Zend Technologies</a>. <h2>What is the Zend
Optimizer?</h2><p>The Zend Optimizer is one of the most popular PHP plugins for
performance-improvement, and has been available without charge, since the early
days of PHP 4. It improves performance by scanning PHP's intermediate code and
passing it through multiple Optimization Passes to replace inefficient code
patterns with more efficient code blocks. The replaced code blocks perform
exactly the same operations as the original code, only faster. </p><p>In addition
to improving performance, the Zend Optimizer also enables PHP to transparently
load files encoded by the Zend Guard. </p><p>The Zend Optimizer is a free product
available for download from <a href="http://www.zend.com">Zend Technologies</a>.
Zend Technologies also developed the PHP scripting engine, known as the <a
href="http://www.zend.com/products/zend_engine">Zend
Engine</a>.</p></body></html>
EOM;
exit();
__halt_compiler();
?>

```

Рис. 51.7. Закодированный PHP-файл

51.4. Ошибка Unable to locate XML file

Вы выбираете команду **Encode**, а вместо закодированного файла видите сообщение о невозможности обнаружить XML-файл проекта. Ошибка связана со сложным путем к XML-файлу (пробелы, нестандартные символы). Нужно поместить XML-файл проекта в каталог с "простым" названием (например, C:\projects). После этого задать расположение этого файла можно опцией `--xml-file`:

```
GuardEngine --xml-file <xml-file-path>
```

51.5. Лицензии

Да, никто не может отредактировать и даже прочитать зашифрованные сценарии. Однако они по-прежнему выполняются. И если вы не предусмотрели в своем коде специальных защитных мероприятий, кто-то может опять установить вашу CMS на несколько узлов, при этом заплатив только за один. Или же вообще не заплатив — код ведь выполняется нормально. А редактировать его и не нужно — все и так работает.

Ограничить использование кода можно путем лицензирования. Вы можете указать или время работы кода, или установить ограничения лицензии. Но можно обойтись и без лицензирования.

В свое приложение вы помещаете код, который скачивает с вашего сервера список ваших клиентов (их доменные имена). Если доменного имени сервера, на котором запущено приложение, нет в списке клиентов, то ваше приложение просто прекращает работу. Это самый простой алгоритм, позволяющий защитить код от несанкционированного копирования.

Создать лицензию можно командой **File | New | Product License**. Далее следуйте инструкциям (рис. 51.8):

- ☐ выберите имя продукта (**Product Name**) и введите имя того, для кого предназначена лицензия (**Register To**);
- ☐ введите полный путь к файлу лицензии (**Save License As**);
- ☐ сгенерируйте GPK-ключ. Для этого нажмите ссылку **Configure License Keys** и в открывшемся окне нажмите кнопку **Generate**. Вернитесь в предшествующее окно и выберите только что созданный GPK-ключ;

Product License

Create Product License

Gather information from existing license file

Product Information

Product Name: Example

Register To: Home Lts

Target File

Save License As: C:\den\license Browse...

Guard Private Key

Generated License Key 1 Configure License Keys

☐ Use License Key saved in external file Browse...

Product Expiration

☒ License never expires

☐ License expires on: 01/04/2012

< Back Next > Finish Cancel

Рис. 51.8. Создание лицензии

- ☐ введите срок окончания лицензии (**Product Expiration**). В trial-версии эта возможность заблокирована, т. к. генерируемые лицензии и так ограничены сроком в три дня;
- ☐ нажмите кнопку **Next**.

Вы также можете ограничить действие лицензии IP-адресами, Zend-хостами и ограничить количество одновременно работающих пользователей (рис. 51.9) — одним словом, выбор ограничений огромен. Проще всего ввести IP-адреса клиента — на других серверах ваш сценарий работать не будет.

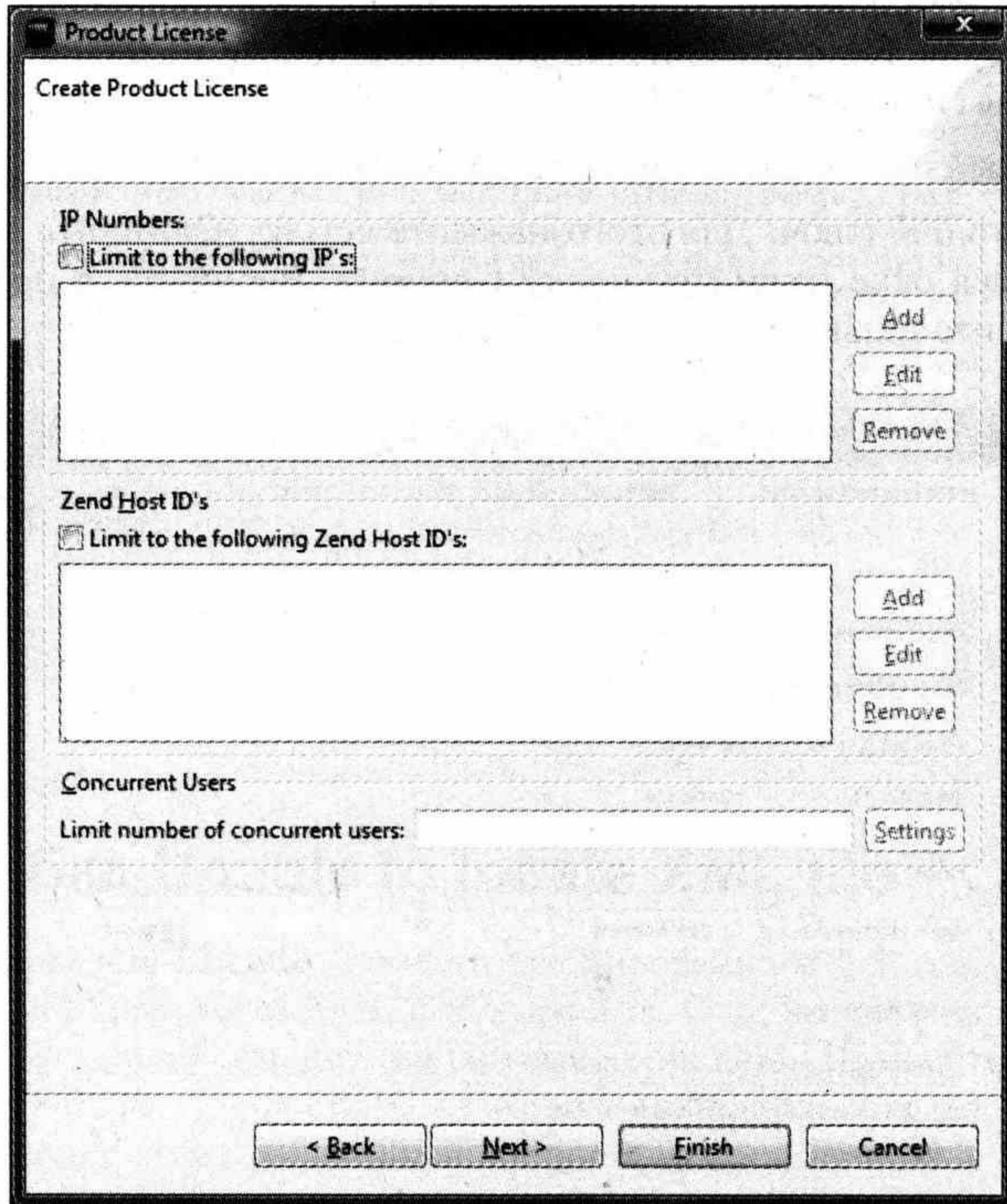


Рис. 51.9. Ограничение действия лицензии

В лицензию можно добавить дополнительную информацию (с помощью кн **Add**) — например, об авторских правах (рис. 51.10). Завершается создание ли зии нажатием кнопки **Finish**.

Созданную лицензию нужно подключить к проекту. Для этого на вкладке **Sec** следует выбрать только что созданную лицензию (рис. 51.11).

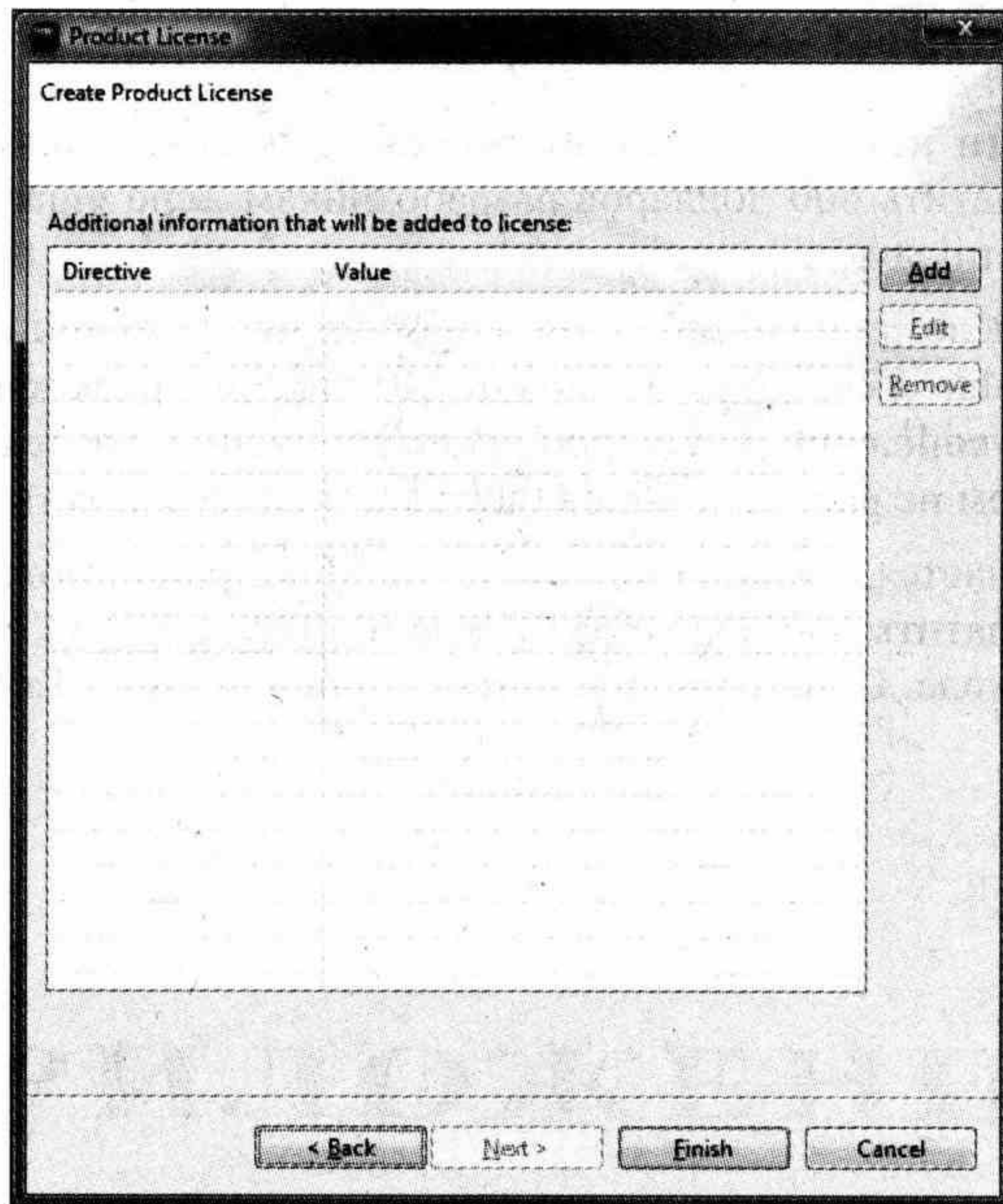


Рис. 51.10. Дополнительная информация

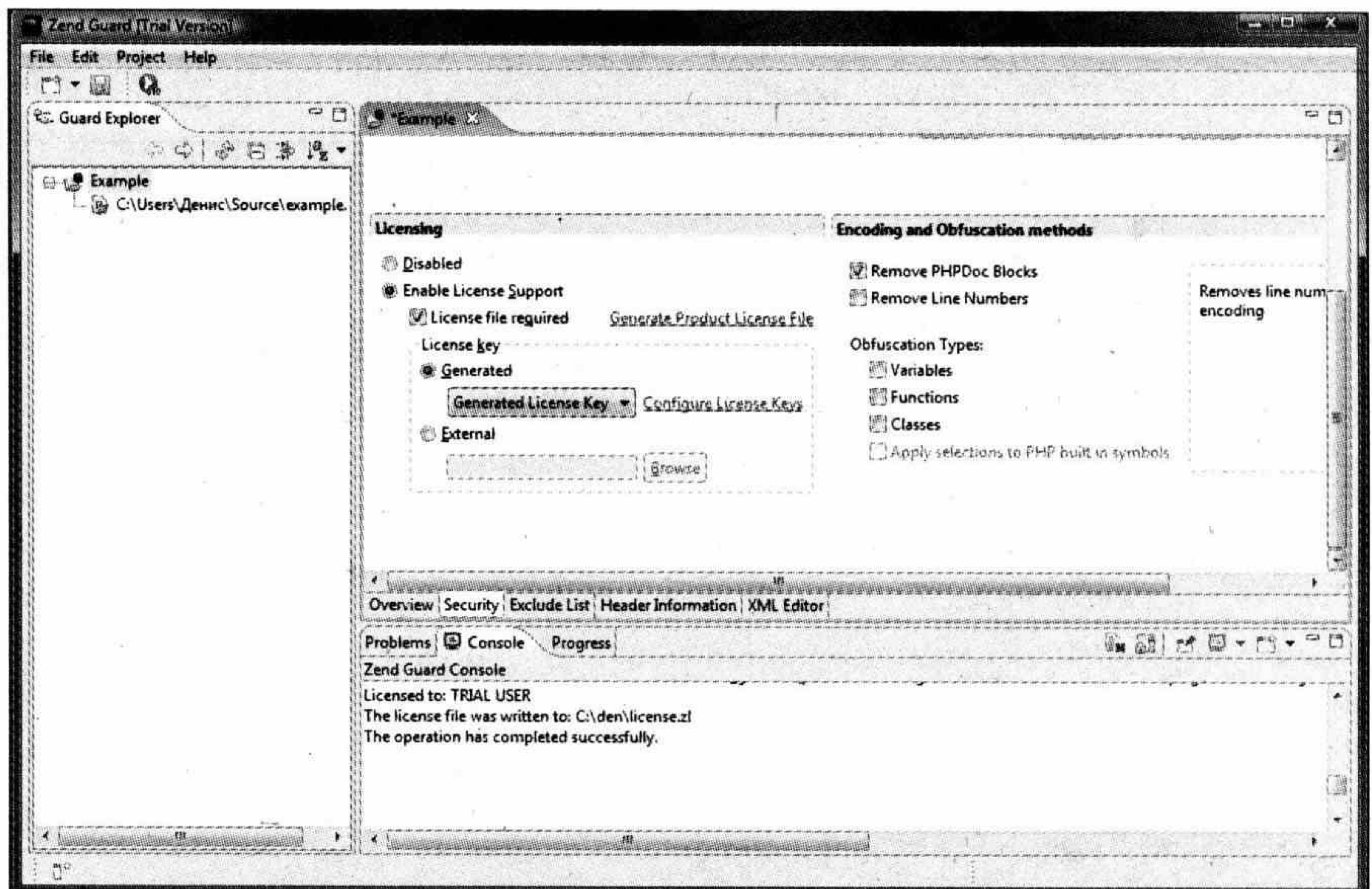


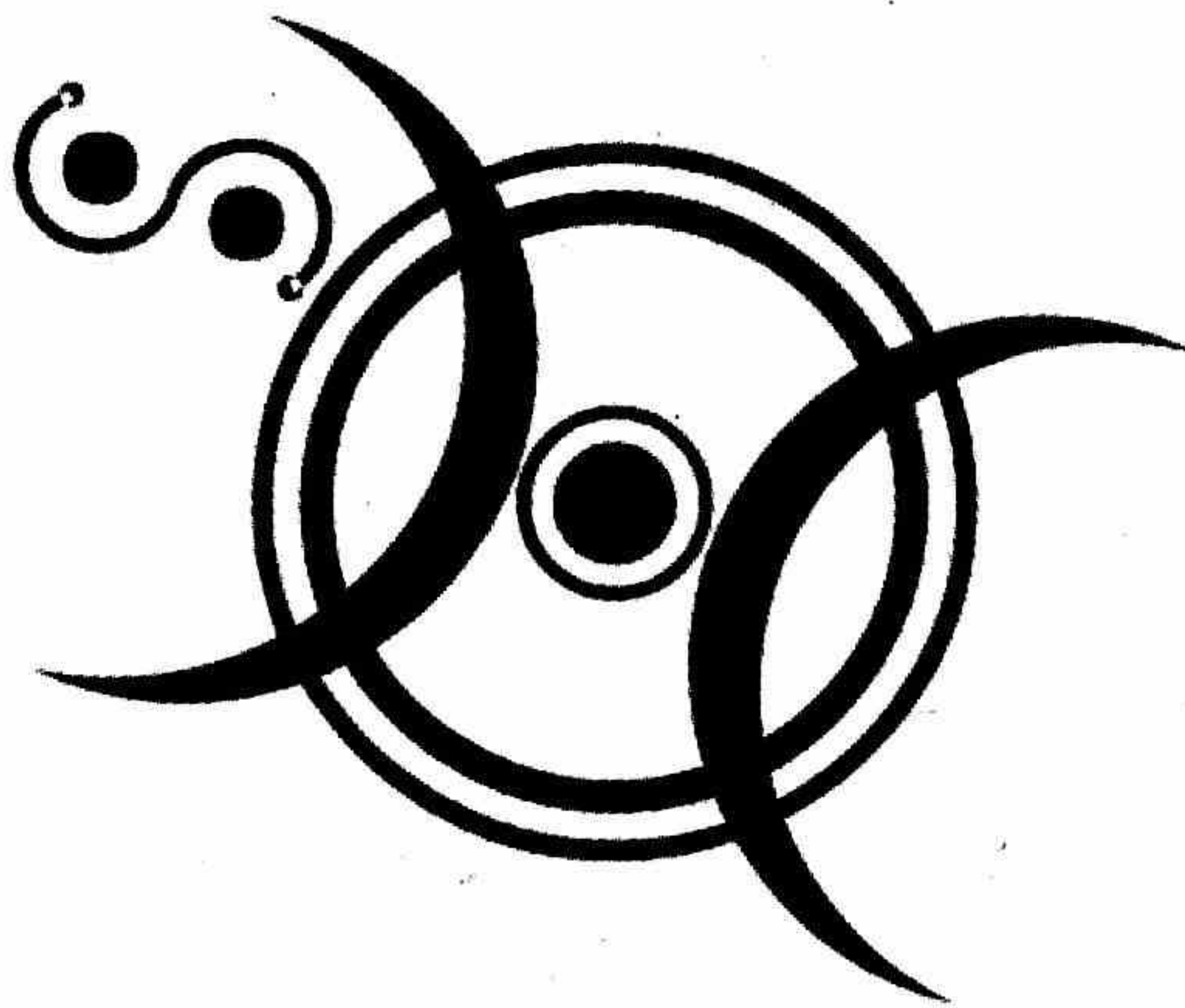
Рис. 51.11. Лицензирование продукта

51.6. Zend Guard — панацея?

Поражаюсь алчности некоторых разработчиков. Они хотят защитить свои сценарии, но не хотят платить 600 долларов разработчикам Zend Guard, пользуясь взломанными версиями продукта.

Чтобы у вас не было такого соблазна, сообщу, что сценарии, закодированные предыдущей версией Zend Guard, можно бесплатно раскодировать на сайте <http://www.showmycode.com/>. Сценарии, зашифрованные последней версией Zend Guard, этим ресурсом не раскодируются (во всяком случае, пока).

Как ни крути, получается, что байт-код Zend Guard все равно можно раскодировать. Стоит ли за него платить 600 долларов? С одной стороны, есть вероятность раскодирования вашего кода. С другой, дилетанты это не сделают. Так что решать только вам.



ПРИЛОЖЕНИЯ

Приложение 1.	Программа phpMyAdmin
Приложение 2.	Flash и PHP
Приложение 3.	Оптимизация PHP-кода
Приложение 4.	Описание электронного архива

ПРИЛОЖЕНИЕ 1

Программа phpMyAdmin

phpMyAdmin — веб-приложение, предназначенное для реализации функций MySQL-клиента. Использовать эту программу намного удобнее, чем стандартный MySQL-клиент — программу `mysql`, входящую в состав сервера MySQL. phpMyAdmin обладает интуитивно-понятным интерфейсом, разобраться с которым может даже самый начинающий пользователь.

Особенность phpMyAdmin заключается в том, что для работы с базой данных пользователю совсем не обязательно знать язык SQL. Программа позволяет создавать, модифицировать и удалять таблицы и записи таблиц без знания SQL. После каждой операции программа выводит SQL-запрос, который был выполнен, так что с помощью этой программы можно легко и наглядно выучить SQL.

Дистрибутив phpMyAdmin можно скачать со страницы по ссылке: http://www.phpmyadmin.net/home_page/index.php.

Распакуйте полученный архив в какой-нибудь каталог. Скопируйте файл `config.sample.inc.php` в файл `config.inc.php` и откройте файл `config.inc.php` — а это основной конфигурационный файл phpMyAdmin — в любом текстовом редакторе. Найдите в нем следующую строку:

```
$cfg['Servers'][$i]['host'] = 'localhost';
```

Как вы уже догадались, эта переменная устанавливает имя MySQL-сервера. В большинстве случаев значение `localhost` изменять не требуется.

Затем вам нужно добавить следующую строку:

```
$cfg['Servers'][$i]['auth_type'] = 'http';
```

Эта переменная устанавливает тип аутентификации. Тип `http` подразумевает ввод имени пользователя и пароля в окне, показанном на рис. П1.1.

Кроме типа аутентификации `http` можно использовать тип аутентификации `config`, когда имя пользователя и пароль хранятся в файле конфигурации (тогда программа не будет их запрашивать). Однако такой режим назначать не рекомендуется из соображений безопасности.

После входа в phpMyAdmin вы увидите список баз данных, доступных пользователю, под именем которого вы вошли в программу. Вам нужно выбрать одну из баз

данных, с которой вы хотите работать (рис. П1.2). Если вашему пользователю доступна только одна база данных, то вы сразу увидите список таблиц этой базы данных (рис. П1.3).

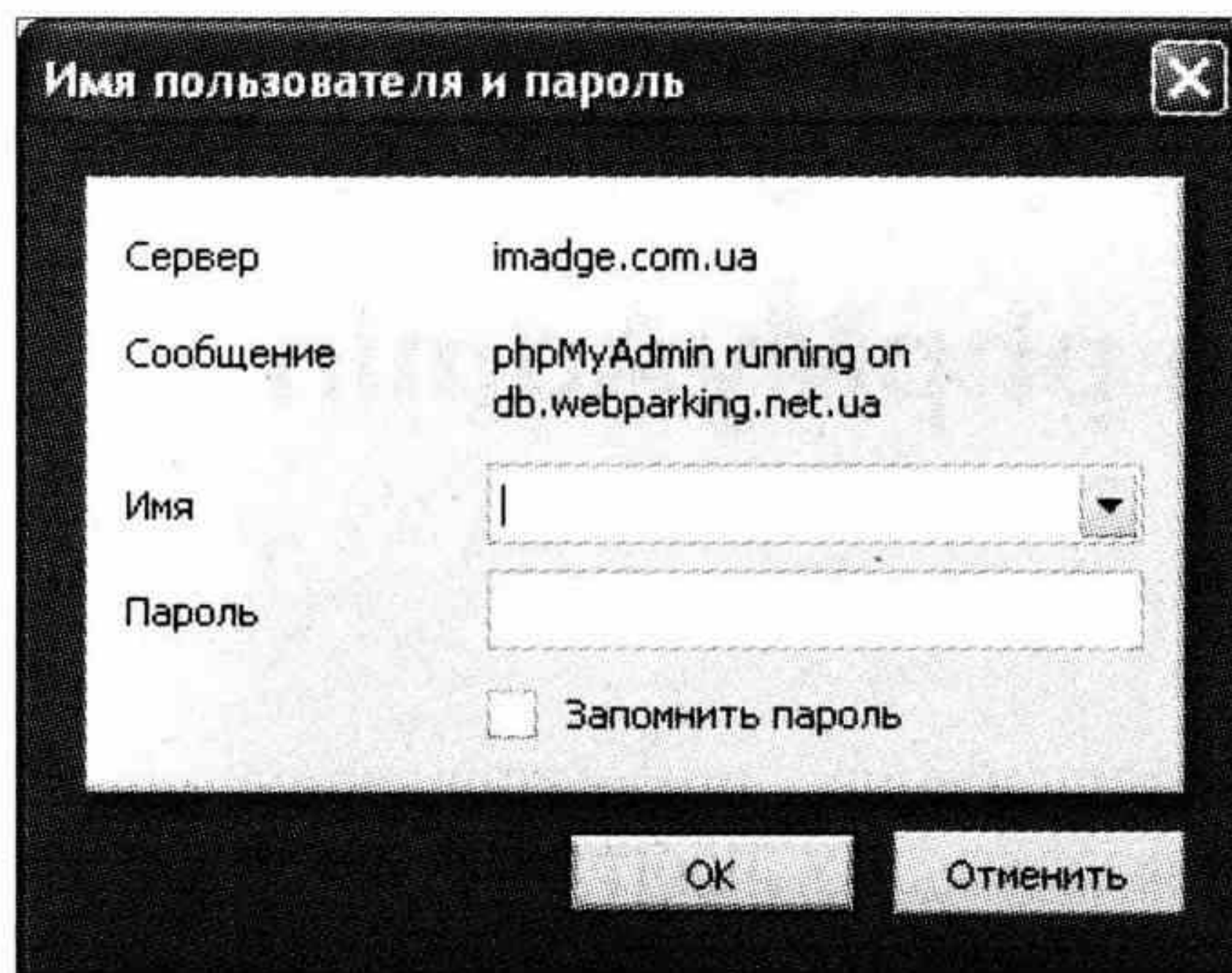


Рис. П1.1. Окно для ввода имени пользователя и пароля

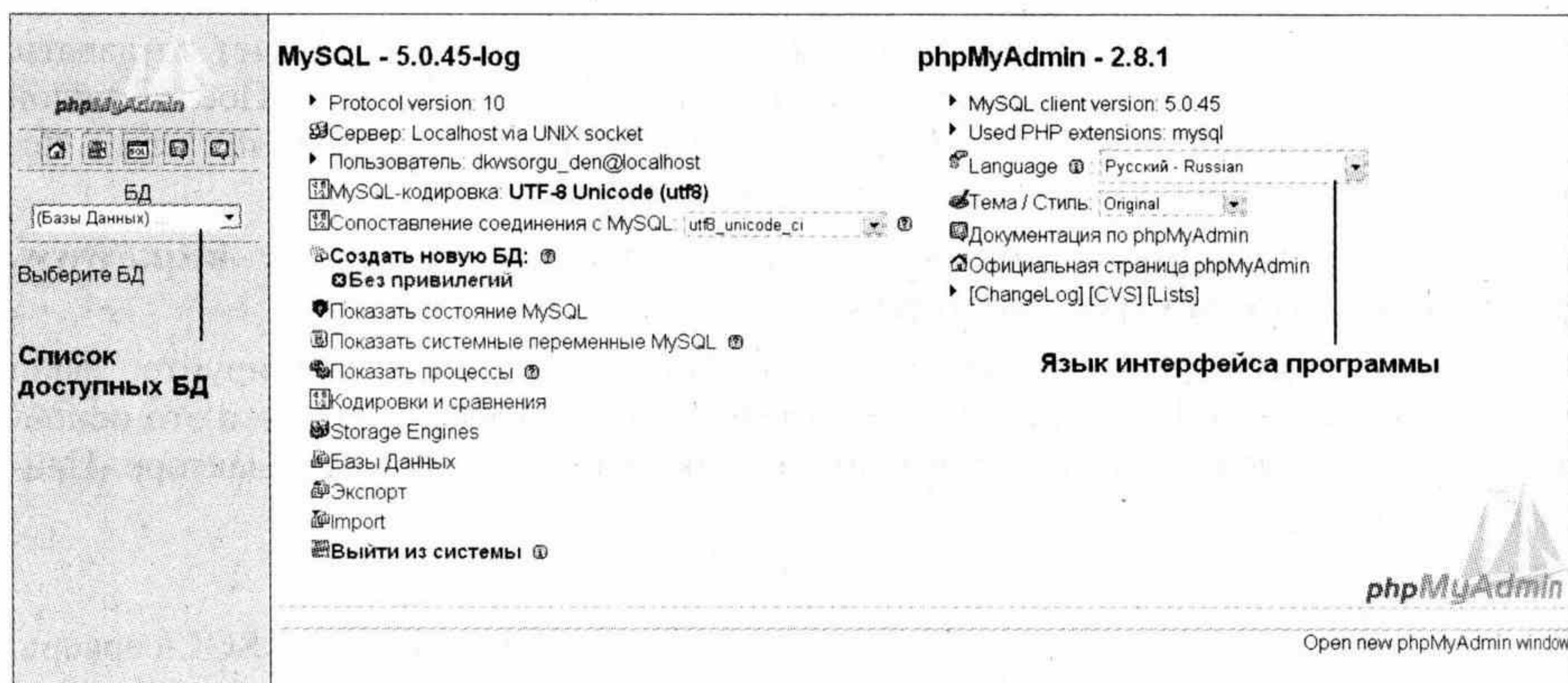


Рис. П1.2. Выбор базы данных

П1.1. Действия над таблицами

В колонке **Действие** (см. рис. П1.3) отображается список действий, доступных с конкретной таблицей (рис. П1.4). Вы можете просмотреть содержимое таблицы, просмотреть структуру таблицы, произвести поиск по таблице, вставить новую запись, удалить все записи и удалить таблицу.

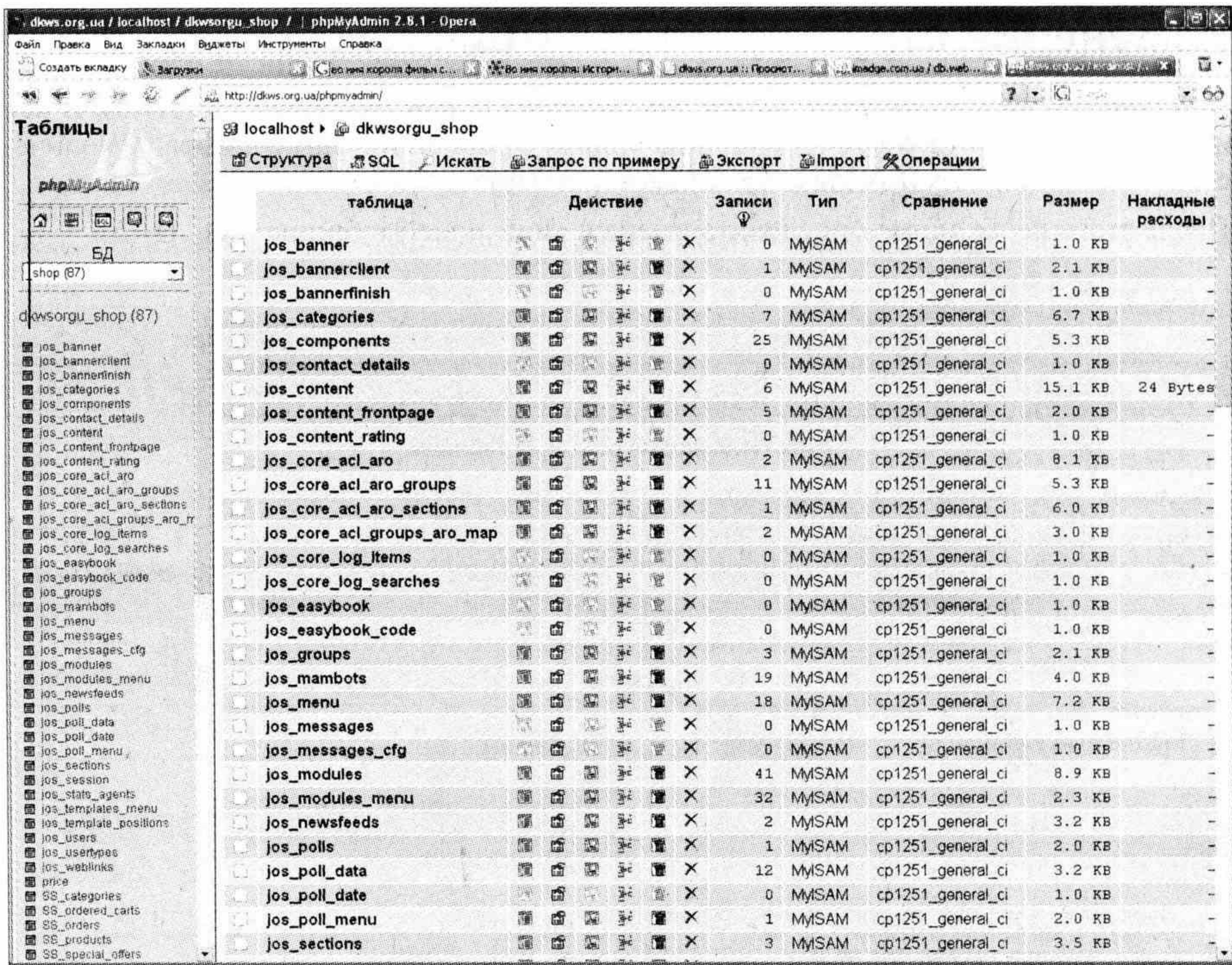


Рис. П1.3. Список таблиц



Рис. П1.4. Список доступных действий

П1.2. Создание новой таблицы. Изменение структуры таблицы

Под списком таблиц имеется форма для создания новой таблицы (рис. П1.5). В нее нужно ввести имя таблицы (допустимы только буквы латиницы и цифры!), количество полей (**Number of fields**) и нажать кнопку **Пошел**.

После этого необходимо определить имена и типы полей таблицы (рис. П1.6).

Рис. П1.5. Форма создания таблицы

1 Для типов поля "enum" и "set", введите значения по этому формату: 'a','b','c'...
Если вам понадобится ввести обратную косую черту ("\"") или одиночную кавычку (") среди этих значений, поставьте перед ними обратную косую черту (например, \"xyz\" или 'a\\'b').

2 Для значений по умолчанию просто введите единственное значение без экранирования и кавычек, используя этот формат: a

Рис. П1.6. Имена и типы полей таблицы

Определив имена полей и их типы, нажмите кнопку **Сохранить** для создания таблицы.

В открывшемся окне (рис. П1.7) вы увидите:

- ☐ доступные операции над таблицей (операция **Обзор** недоступна, поскольку таблица пока не содержит данных);
- ☐ запрос, который использовался для создания таблицы;
- ☐ структуру таблицы;
- ☐ информацию об индексах.

Обратите внимание:

- ☐ вы можете добавлять поля таблицы — для этого в поле **Add** введите количество полей, которые вы хотите добавить, и нажмите кнопку **Пошел**. Затем вам нужно будет определить имена добавленных полей и их типы;

❑ вы можете изменить любое поле (колонок **Действие** для поля): изменить его тип, сделать поле первичным индексом (ключом), присвоить атрибут уникальности или же просто удалить поле.

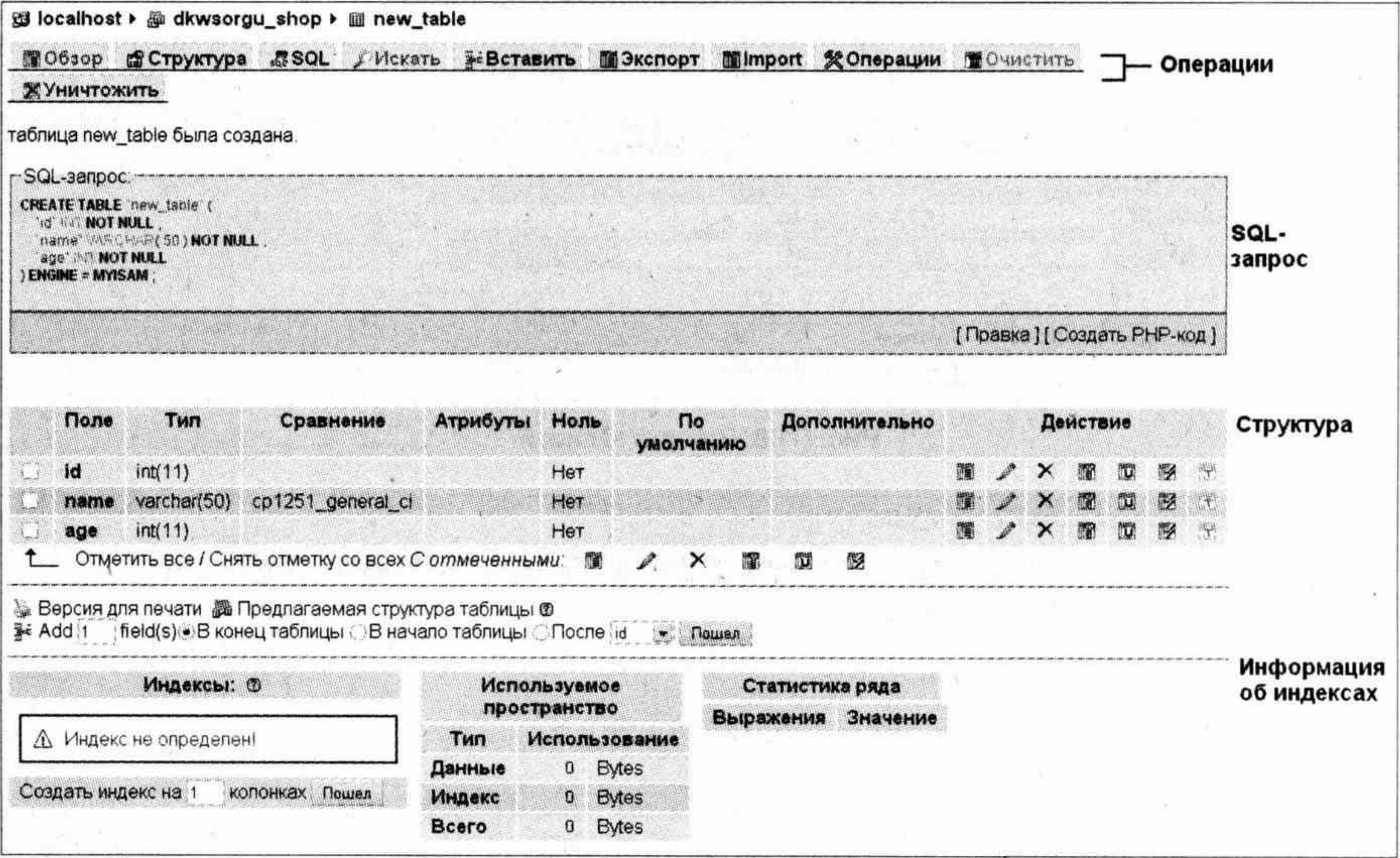


Рис. П1.7. Окно новой таблицы

П1.3. Вставка записей

Для добавления новой записи нажмите кнопку **Вставить** в верхней части окна (см. рис. П1.7). Программа позволяет добавлять сразу две записи (рис. П1.8), но если вы хотите добавить только одну запись, отметьте флажок **Игнорировать** для второй записи.

П1.4. Обзор таблицы

Как только в таблице появится хотя бы одна запись, кнопка **Обзор** станет доступной (рис. П1.9). В режиме обзора вы можете производить операции над записями таблицы: добавлять, редактировать, удалять.

П1.5. Выполнение произвольного SQL-кода

Нажав кнопку **SQL**, вы можете ввести произвольный SQL-код, который будет передан MySQL-серверу для выполнения (рис. П1.10).

localhost > dkwsorgu_shop > new_table

Обзор Структура SQL Искать Вставить Экспорт Import ~~Операции~~ Очистить

~~Уничтожить~~

Поле	Тип	Функция	Ноль	Значение
id	int(11)		1	
name	varchar(50)			Denis
age	int(11)		24	

☐ Игнорировать

Поле	Тип	Функция	Ноль	Значение
id	int(11)		2	
name	varchar(50)			Max
age	int(11)		20	

Вставить новый ряд and then Возврат

Пошел Переустановить

Рис. П1.8. Вставка записей

localhost > dkwsorgu_shop > new_table

Обзор Структура SQL Искать Вставить Экспорт Import ~~Операции~~ Очистить

~~Уничтожить~~

Показывает записи 0 - 1 (2 всего, Запрос занял 0.0001 сек)

SQL-запрос

SELECT *
FROM `new\_table`
LIMIT 0, 30

[Правка][Описать SQL][Создать PHP-код][Обновить]

Показать: 30 рядов от 0

В горизонтальном режиме, заголовки после каждых 100 ячеек

	Id	name	age
☐	1	Denis	24
☐	2	Max	20

↑ Отметить все / Снять отметку со всех С отмеченными: ☐ ☒

Показать: 30 рядов от 0

В горизонтальном режиме, заголовки после каждых 100 ячеек

Рис. П1.9. Обзор таблицы

localhost > dkwsorgu_shop > new_table

Обзор Структура SQL Искать Вставить Экспорт Import ~~Операции~~ Очистить

~~Уничтожить~~

Выполнить SQL запрос(ы) на БД

SELECT * FROM `new\_table` WHERE 1

Поля
id
name
age

☒ Показать данный запрос снова

Пошел

Рис. П1.10. Ввод произвольного SQL-кода

П1.6. Резервное копирование БД

Программа phpMyAdmin позволяет выполнить резервное копирование базы данных. Нажмите кнопку **Экспорт**. Не изменяйте никакие параметры экспорта (рис. П1.11), а просто нажмите кнопку **Пошел**. Если БД уж очень большая, установите сжатие в ZIP-файл.

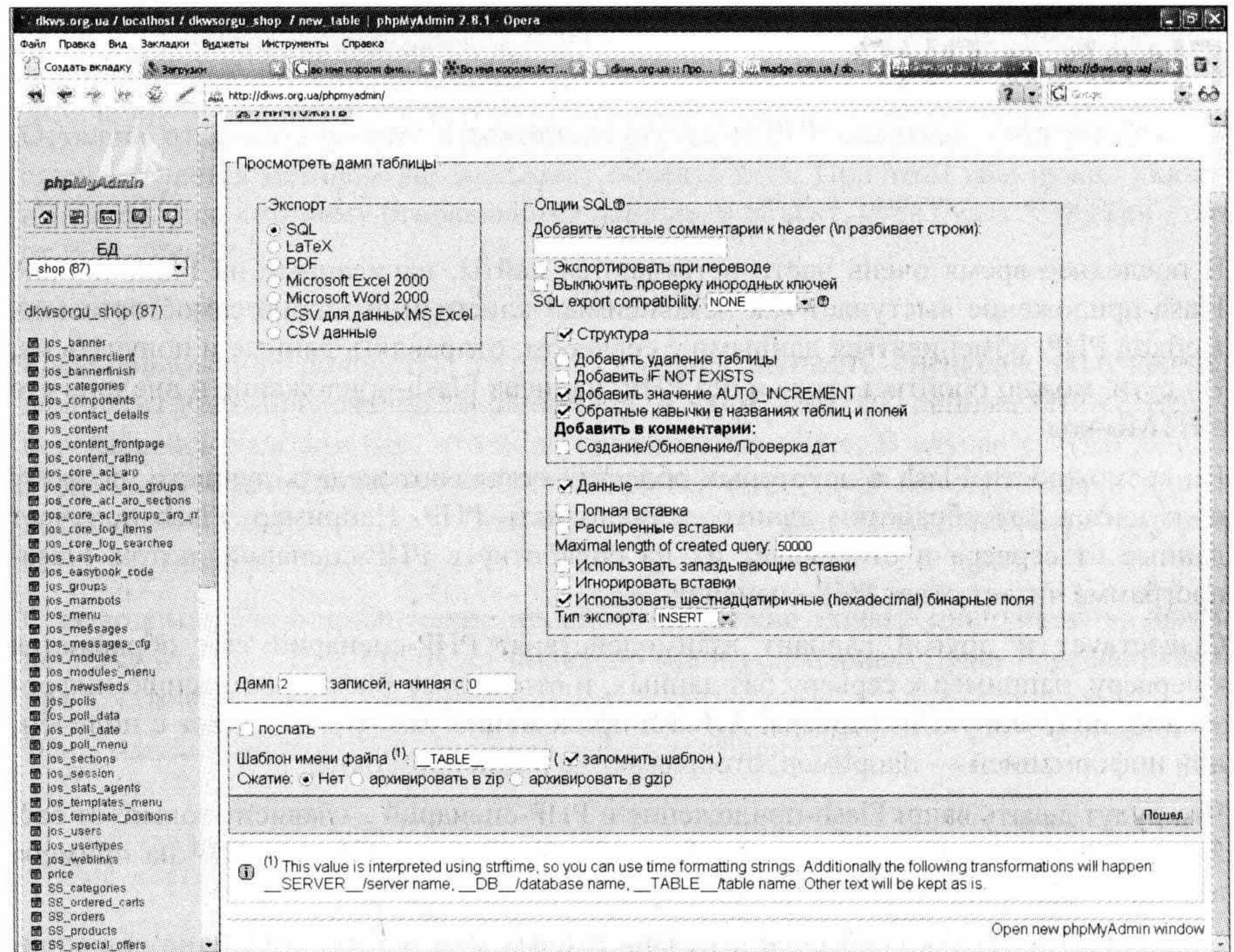


Рис. П1.11. Параметры экспорта

На выходе вы получите SQL-код, который нужно сохранить как файл с расширением `sql`. Если вы применили ZIP-сжатие, то будет создан ZIP-архив с SQL-файлом.

Если SQL-файл небольшой, то для восстановления его из резервной копии достаточно скопировать его содержимое в буфер обмена, затем нажать кнопку **SQL** и вставить SQL-код в соответствующую форму (см. рис. П1.10). Если же база данных большая, и вы использовали ZIP-сжатие, то для восстановления из резервной копии лучше использовать функцию **Импорт**. При импорте надо указать текстовый файл (а не архив), содержащий SQL-код.

ПРИЛОЖЕНИЕ 2

Flash и PHP

В последнее время очень часто встречаются сайты, написанные на Flash и PHP. Flash-приложение выступает как независимый клиент и способно самостоятельно, в обход PHP, обмениваться данными с сервером: отправлять данные и получать их. По сути, можно обойтись даже и без PHP, написав Flash-приложение и внедрив его в HTML-код.

Но возможности Flash в некоторых областях оставляют желать лучшего, поэтому приходится для обработки данных использовать PHP. Например, Flash получает данные от сервера и отправляет их на обработку в PHP-сценарий, потом Flash-программа читает ответ PHP-сценария.

Существует и другой вариант взаимодействия: PHP-сценарий сам обращается к серверу, например к серверу баз данных, и отправляет Flash-приложению информацию, полученную из сервера. А Flash-приложение уже что-то делает с полученной информацией — например, отображает ее в красивом окне.

Что будут делать ваши Flash-приложение и PHP-сценарий — зависит только от вас. Я лишь покажу, как организовать обмен данными между Flash и PHP на примере передачи одной строки.

Существует несколько способов передачи данных между Flash и PHP, но общепринятой практикой является использование XML.

Прежде всего нужно в Flash-приложении создать объект XML:

```
var str1 = "some string"  
var xml1:XML = new XML(str1);
```

Мы объявили переменную `str1` (это наша строка) и объект `xml1`. После этого следует задать тип выводимого контента (из Flash-приложения):

```
xml1.contentType = "text/xml";
```

Затем надо подготовить обработчик, который станет заниматься получением результата. Ведь нас интересует не столько отправка данных в PHP-сценарий, сколько получение ответа от PHP-сценария — результата обработки отправленных данных.

Наш обработчик будет выглядеть так:

```
var response_xml:XML = new XML();
response_xml.onLoad = function(success) {
    if(success) {
        trace(this.toString());
    } else {
        trace("error");
    }
}
```

Осталось отправить данные и дождаться ответа от PHP-сценария. Отправка данных осуществляется методом `sendAndLoad()` объекта `xml1`. При этом нам нужно указать имя сценария, которому отправляются данные, и объект, куда будет загружен ответ от PHP-сценария:

```
xml1.sendAndLoad("process.php", response_xml);
```

Flash-приложение полностью готово. Осталось написать сценарий, получающий данные от Flash-приложения. В случае с AJAX (см. главу 35) данные PHP-сценарию передавались методом GET, что облегчало их получение. В случае с Flash данные отправляются непосредственно на стандартный ввод (такой метод передачи данных называется RAW POST DATA). Получить RAW POST DATA можно вызовом функции `file_get_contents("php://input")`.

С выводом в Flash-приложение проще — не нужно ничего придумывать, просто используйте `echo`. Все, что будет выведено вашим сценарием, будет передано Flash-приложению.

Итак, наш небольшой PHP-сценарий теперь выглядит так:

```
<?php
// получаем данные из Flash-приложения
$input = file_get_contents("php://input");

// обрабатываем данные, результат записываем в переменную
// $result:
$result = "Hello, world!";

// выводим результат
echo $result;
?>
```

Вот, на этом и все. Надо только отметить, что все данные, выводимые из PHP, должны быть выведены в кодировке UTF-8. Когда вы выводите только англоязычный текст и цифры, кодировка UTF-8 не обязательна, а вот для корректного отображения кириллицы требуется UTF-8. Для перекодирования в UTF-8 можно использовать функцию `iconv()`:

```
iconv("CP1251", "UTF-8", "текст");
```


Однако у этой функции есть два недостатка: она не везде работает (ее наличие зависит от настроек PHP) и она создает большую нагрузку на сервер, что во многих серьезных проектах является критичным. Многие программисты предпочитают разрабатывать свои функции для преобразования текста в кодировке cp1251 в UTF-8. Вот одна из таких функций:

```
function cp1251_to_utf8 ($txt) {
    $in_arr = array (
        chr(208), chr(192), chr(193), chr(194),
        chr(195), chr(196), chr(197), chr(168),
        chr(198), chr(199), chr(200), chr(201),
        chr(202), chr(203), chr(204), chr(205),
        chr(206), chr(207), chr(209), chr(210),
        chr(211), chr(212), chr(213), chr(214),
        chr(215), chr(216), chr(217), chr(218),
        chr(219), chr(220), chr(221), chr(222),
        chr(223), chr(224), chr(225), chr(226),
        chr(227), chr(228), chr(229), chr(184),
        chr(230), chr(231), chr(232), chr(233),
        chr(234), chr(235), chr(236), chr(237),
        chr(238), chr(239), chr(240), chr(241),
        chr(242), chr(243), chr(244), chr(245),
        chr(246), chr(247), chr(248), chr(249),
        chr(250), chr(251), chr(252), chr(253),
        chr(254), chr(255)
    );

    $out_arr = array (
        chr(208).chr(160), chr(208).chr(144), chr(208).chr(145),
        chr(208).chr(146), chr(208).chr(147), chr(208).chr(148),
        chr(208).chr(149), chr(208).chr(129), chr(208).chr(150),
        chr(208).chr(151), chr(208).chr(152), chr(208).chr(153),
        chr(208).chr(154), chr(208).chr(155), chr(208).chr(156),
        chr(208).chr(157), chr(208).chr(158), chr(208).chr(159),
        chr(208).chr(161), chr(208).chr(162), chr(208).chr(163),
        chr(208).chr(164), chr(208).chr(165), chr(208).chr(166),
        chr(208).chr(167), chr(208).chr(168), chr(208).chr(169),
        chr(208).chr(170), chr(208).chr(171), chr(208).chr(172),
        chr(208).chr(173), chr(208).chr(174), chr(208).chr(175),
        chr(208).chr(176), chr(208).chr(177), chr(208).chr(178),
        chr(208).chr(179), chr(208).chr(180), chr(208).chr(181),
        chr(209).chr(145), chr(208).chr(182), chr(208).chr(183),
        chr(208).chr(184), chr(208).chr(185), chr(208).chr(186),
        chr(208).chr(187), chr(208).chr(188), chr(208).chr(189),
        chr(208).chr(190), chr(208).chr(191), chr(209).chr(128),
        chr(209).chr(129), chr(209).chr(130), chr(209).chr(131),
        chr(209).chr(132), chr(209).chr(133), chr(209).chr(134),
```



```
chr(209).chr(135), chr(209).chr(136), chr(209).chr(137),  
chr(209).chr(138), chr(209).chr(139), chr(209).chr(140),  
chr(209).chr(141), chr(209).chr(142), chr(209).chr(143)
```

```
);
```

```
$txt = str_replace($in_arr,$out_arr,$txt);
```

```
return $txt;
```

```
}
```

Эта функция создает гораздо меньшую нагрузку на сервер, чем `iconv()`.

ПРИЛОЖЕНИЕ 3

Оптимизация PHP-кода

П3.1. Правильное использование *echo*

Оператор `echo` может существенно снизить производительность программы, поэтому старайтесь по возможности его избегать. Если нужно вывести несколько строк, постарайтесь обойтись одним оператором `echo`, а не несколькими.

Рассмотрим небольшой пример:

```
echo "<html>";  
echo "<head>";  
echo "<title>My Site</title>";  
echo "</head>";  
echo "<body>";
```

Это пример нерационального использования ресурсов. Пять операторов `echo` следует заменить всего одним — так будет намного быстрее:

```
echo "<html>  
<head>  
<title>My Site</title>  
</head>  
<body>";
```

Но на производительность сценария влияет не столько количество операторов `echo`, сколько их содержимое. Предположим, что у нас есть переменная `$txt`:

```
$txt = "просто текст";
```

Нам ее нужно вывести еще с некоторым текстом. Мы знаем, что это можно сделать двумя способами:

- ☐ `$txt = "Значение переменной txt: $txt";`
- ☐ `$txt = "Значение переменной txt: " . $txt;`

Обычно реализуют первый способ, поскольку он удобнее. Но второй способ на 40 % быстрее первого. Да, когда вам необходимо вывести всего одну строку, суще-

ственного ускорения вы не почувствуете, но если у вас сложный сценарий, который выводит значения нескольких десятков переменных, результат будет заметен даже без специальных замеров времени выполнения.

П3.2. Длина переменных

Средняя длина переменных в PHP составляет 7–10 символов. Если вы научитесь укладываться в 7 символов, то ускорите работу сценария минимум на 15 %, поскольку, чем короче имя переменной, тем быстрее с ней работает PHP. Конечно, нужно помнить и о том, что имя переменной должно нести в себе какой-то смысл. Особенно это касается главных (результатирующих) переменных сценария. Счетчики и промежуточные переменные (переменные, содержащие промежуточные результаты) можно называть кратко: `$i`, `$j`, `$k`, `$A`.

Предположим, что у нас есть переменная с очень длинным именем. Очень длинное имя — это всего лишь 32 символа. Если принять время обработки этой переменной интерпретатором PHP за единицу (100 %), то переменная с именем в один символ будет обрабатываться быстрее на 32 %. Если имя переменной не превысит 7 символов, PHP будет с ней работать быстрее на 30 %, а вот если имя переменной превысит эти 7 символов, то разница будет менее заметна — ускорение составит всего 22 %.

Если же увеличить длину переменной в два раза (14–16 символов), то PHP будет работать с ней почти так же, как и с длинной переменной (в 32 символа), — ускорение окажется небольшим — порядка 10–15 %.

Понятно, что особо себя ущемлять не следует, ведь разница между очень коротким именем переменной (1 символ) и нормальным (до 7 символов) составляет всего 2 % — ни вы, ни пользователи это не почувствуете. А когда граница в 7 символов превышена, тогда производительность начинает снижаться.

П3.3. Правильное обращение к элементам массива

Существенно ускорить PHP-программу можно за счет правильного обращения к ассоциативным и многомерным массивам (конечно, при условии, что ваша программа вообще использует такие массивы).

Обратиться к элементу ассоциативного массива можно двумя способами: `$ARR[name]` и `$ARR['name']`. Первый способ удобнее, поскольку не нужно заключать имя элемента массива в кавычки. Зато второй способ намного эффективнее. В зависимости от специфики вашей программы можно добиться ускорения до 40 %. Ради этих 40 % многие готовы пожертвовать своим комфортом, во всяком случае, я — точно.

Если в вашей программе есть многомерный массив, и вам требуется вывести значения его элементов, тогда переменную массива при выводе текста нужно вынести за пределы строки, как это мы делали с переменными.

Вот пример неоптимизированного кода:

```
echo "Телефон: $ARR[name][surname][phone]";
```

А вот как будет выглядеть код после оптимизации:

```
echo "Телефон: " . $ARR['name']['surname']['phone'];
```

Мы вынесли переменную массива за пределы текстовой строки, а также заключили имена элементов массива в апострофы. Благодаря всему этому мы ускорили выполнение данного оператора `echo` еще на 20 %.

П3.4. Циклы

Очень часто в PHP-программах встречаются циклы `foreach`, `for` и `while`. Так вот, везде, где это возможно, старайтесь избегать лишнего цикла `foreach`, поскольку он работает на 30 % медленнее, чем циклы `for` и `while`. Да, при работе с ассоциативными массивами от `foreach` никуда не денешься, однако некоторые программисты применяют `foreach` даже для перебора обычных массивов, а этого делать не следует.

А вот если сравнивать циклы `for` и `while`, то они работают приблизительно с одинаковой скоростью.

При работе с массивами (когда в цикле нужно вывести все их элементы) часто используются функции `count()` и `sizeof()`.

Вот пример функции `count()` для определения числа элементов массива:

```
for ($i=0; $i<count($test); $i++)  
    printf($test[$i]);
```

То же самое можно сделать с помощью функции `sizeof()`:

```
for ($i=0; $i<sizeof($test); $i++)  
    printf($test[$i]);
```

Как вы думаете, какой вариант работает быстрее? Вы будете удивлены, но быстрее работает `sizeof()`. Для тех, кто не читал официальное руководство PHP: функция `sizeof()` — это псевдоним функции `count()`, значит, если подумать логически, `count()` должен работать быстрее. Но на практике получаем совершенно другие результаты. Выводы можно сделать следующие:

- ❑ если число элементов в массиве не превышает 65 000, то разницы между `count()` и `sizeof()` нет;
- ❑ при числе элементов более 65 000, `sizeof()` будет работать немного быстрее. К слову, если число элементов превысит 100 000, то скорость работы `sizeof()` увеличится примерно на 15 %.

П3.5. Чтение файлов

Прочитать текстовый файл можно двумя способами:

1. Открыть файл с помощью `fopen()` и в цикле прочитать его содержимое (с помощью `gfets()`).
2. Прочитать весь файл сразу с помощью функции `file`.

Вот пример кода для первого способа:

```
$f = fopen( "some_big_text_file.txt", "r" ) or die(1);  
while( $x[] = fgets($f,1000) );  
fclose($f);
```

Обратите внимание — мы только читаем файл в массив, а не выводим на экран (в браузер). Вывод нас сейчас не интересует.

Теперь сделаем то же самое (т. е. прочитаем файл в массив `$x`), но с помощью функции `file()`:

```
$x = file( "some_big_text_file.txt " );
```

Здесь предпочтительнее будет функция `file()`, поскольку она, во-первых, удобнее, а во-вторых, в 2,5 раза быстрее.

Давайте проведем небольшой эксперимент. Сначала вычислим время выполнения первого фрагмента кода, а затем — второго. Для этого воспользуемся сценарием, представленным в листинге П3.1.

Листинг П3.1. Чтение файла в массив

```
<?php
```

```
function timer_start()  
{  
    $mtime = microtime();  
    $mtime = explode(" ", $mtime);  
    $mtime = $mtime[1] + $mtime[0];  
    $start_time = $mtime;  
  
    return $start_time;  
}  
  
function timer_end()  
{  
    $mtime2 = microtime();  
    $mtime2 = explode(" ", $mtime2);  
    $mtime2 = $mtime2[1] + $mtime2[0];  
    $end_time = $mtime2;
```



```
    return $send_time;
}

$t1 = timer_start();

$f=fopen("some_big_text_file.txt","r") or die(1);
while($x[]=fgets($f,1000));
fclose($f);

//$x = file("some_big_text_file.txt");

$t2 = timer_end();

$time = substr($t2 - $t1,0,4);

echo "Time ".$time." sec"

?>
```

Что делают функции `timer_start()` и `timer_end()`, думаю, понятно без дополнительных пояснений. Функцию `timer_start()` следует вызвать до выполнения первого оператора нужного нам фрагмента кода, а `timer_end()` — после выполнения последнего оператора этого фрагмента. В результате у нас будет две переменных: `$t1` и `$t2`. Предпоследний оператор вычисляет время выполнения фрагмента (разницу между `$t2` и `$t1`), а последний — выводит это время.

Для эксперимента был выбран текстовый файл размером 1 Мбайт. Первый фрагмент кода справился с чтением заданного файла в массив за 0,05 с. Затем мы закомментировали первый фрагмент кода и раскомментировали второй (выделен жирным). Время выполнения — всего 0,02 с. Конечно, вы могли подумать, что файл оказался прокэширован, поэтому для большей чистоты эксперимента исходный файл был переименован, и немного изменен сценарий (ведь потребовалось указать другое имя файла). Время выполнения — опять 0,02 с.

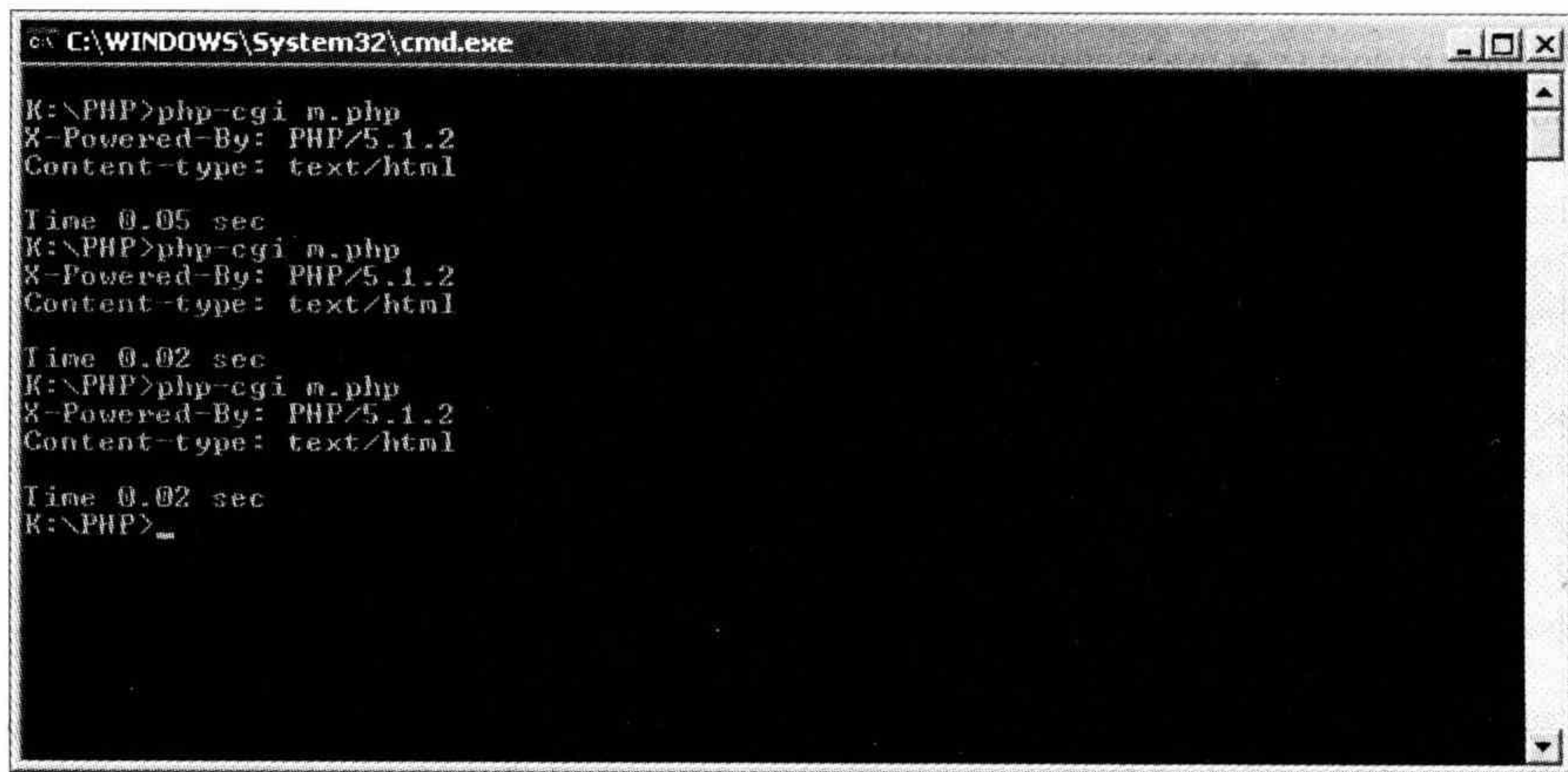
Результат эксперимента показан на рис. ПЗ.1: первый запуск — 0,05 с, два последующих — по 0,02 с.

Но ведь когда мы работаем с файлами, нам нужно не только их прочитать, но и обработать. Поэтому некоторые программисты предпочитают выполнять эти операции одновременно, т. е. в цикле читают строку и сразу же ее обрабатывают. Такое решение целесообразно, когда *размер файла превышает 8 Мбайт*. Почему именно 8 Мбайт? Очень часто хостинг-провайдеры устанавливают ограничители ресурсов для PHP-сценариев. Это понятно — ведь вы далеко не единственный пользователь, поэтому если вы будете использовать ресурсы сервера по максимуму, что тогда останется другим? Правильно, ничего. Поэтому в файле `php.ini` устанавливаются квоты системных ресурсов. По умолчанию квоты выглядят так:

```
; максимальное время выполнения сценария в секундах
max_execution_time = 30
```



```
; максимальное время (в секундах), которое сценарий может потратить  
; на обработку загруженных данных  
max_input_time = 60  
  
; максимальный объем памяти, который может быть выделен сценарию  
memory_limit = 8M
```



```
C:\WINDOWS\System32\cmd.exe  
K:\PHP>php-cgi m.php  
X-Powered-By: PHP/5.1.2  
Content-type: text/html  
Time 0.05 sec  
K:\PHP>php-cgi m.php  
X-Powered-By: PHP/5.1.2  
Content-type: text/html  
Time 0.02 sec  
K:\PHP>php-cgi m.php  
X-Powered-By: PHP/5.1.2  
Content-type: text/html  
Time 0.02 sec  
K:\PHP>_
```

Рис. ПЗ.1. Время выполнения тестового сценария

Если быть предельно точным, то максимальный объем файла, который может быть прочитан с помощью функции `file()`, даже меньше 8 Мбайт. Потому что 8 Мбайт — это вся память, которая выделяется сценарию, но ведь кроме загружаемых данных есть еще и другие переменные, которые тоже занимают место в памяти. Кстати, хостер может без уведомления пользователей изменять указанные лимиты ресурсов, поэтому перед написанием сценария желательно уточнить конкретные значения.

Выводы можно сделать следующие:

- ☐ если размер файла не превышает 7–7,5 Мбайт, тогда функция `file()` будет работать намного быстрее;
- ☐ если же файл больше 7,5 Мбайт, то для чтения файла лучше организовать цикл. Это в 2,5 раза медленнее, зато даст экономию системных ресурсов.

ПЗ.6. Вывод текста: *echo* против *printf*

Для чего нужна оптимизация кода PHP? Хотя бы для того, чтобы уложиться в те самые 30 с (см. *ранее*), которые отводятся на выполнение сценария. В самом начале этого приложения мы разобрались, как правильно использовать оператор `echo`. Но кроме `echo` вывести текст можно и с помощью функции `printf()`. Что лучше: `echo` или `printf()`? Чтобы получить ответ на этот вопрос, немного модифицируем лис-

тинг ПЗ.1. Читать файл мы будем с помощью `file`, а вот выводить — с помощью `echo` в первый раз и `printf()` — во второй.

Рассмотрим фрагмент кода:

```
$x = file("some_big_text_file.txt");
for ($i=0; $i<sizeof($x); $i++) echo($x[$i]);
//for ($i=0; $i<sizeof($x); $i++) printf($x[$i]);
```

В первом случае выполнение сценария заняло 8,48 с. Во втором — 8,40 с. Последующие запуски сценария показали, что особой разницы между `echo` и `printf()` нет — работают они приблизительно с одинаковой скоростью. Причем однозначно нельзя сказать, что `printf()` всегда работает быстрее (немного, но быстрее), чем `echo`. При выводе на консоль (т. е. при локальном запуске сценария) `printf()` работала немного быстрее. А вот когда нужен вывод в браузер (т. е. когда сценарий запускается на сервере, и вывод сценария передается в браузер), тогда `printf()` работает существенно медленнее: 14,3 с против 9,10 с у `echo` (рис. ПЗ.2). Конечно, тут можно учесть погрешность сервера (т. е. в момент, когда запускался `echo`-сценарий, сервер, возможно, был менее загружен, чем в момент запуска `printf`-сценария), но все равно разница впечатляет.

Таким образом, для вывода текста предпочтительнее использовать оператор `echo`.



Рис. ПЗ.2. Время выполнения `printf()`

В разд. ПЗ.1 было отмечено, что строки, по возможности, нужно стараться вывести одним оператором `echo`. Возможно, у вас появилась мысль вместо нашего цикла прочитать все содержимое файла в одну переменную, а потом вывести ее? Другими словами, заменить код:

```
$x = file("some_big_text_file.txt");  
for ($i=0; $i<sizeof($x); $i++) echo($x[$i]);
```

более компактным:

```
$txt = join('', file("some_big_text_file.txt"));  
echo $txt;
```

Парадоксально, но предложенный код работает медленнее, чем исходный вариант. Получается, что функция `join()`, объединяющая все элементы массива в одну строку, работает медленнее, чем 11 915 (именно столько строк было в файле) вызовов `echo` в цикле. Если вы запустите модифицированный сценарий, то заметите, что вывод начнется не сразу, а с небольшой задержкой — это элементы массива объединяются в строку.

Как же выводить текст? Я рекомендую создавать более компактный код. Он проще для восприятия человеком и занимает меньше места при написании. Разница во времени несущественна: первый фрагмент выполняется в среднем 8,5 с, а второй — за 8,9. И это при довольно большом размере файла (1 Мбайт). Впрочем, обычно редко когда приходится выводить такие большие файлы, поэтому разница не будет заметна. Конечно, если вам дорога каждая миллисекунда (у вас очень сложный сценарий, который еле-еле вписывается в отведенные 30 с), то экономия даже 0,4 с окажется значительной.

ПЗ.7. "Лишние" вызовы функций

Иногда так увлекаешься самим процессом программирования, что и не замечаешь, как одна и та же функция вызывается несколько раз, причем ее вызовы совершенно бесполезны. Посмотрите еще раз на приведенный далее цикл, и вы поймете, о чем я говорю:

```
for ($i=0; $i<sizeof($x); $i++) echo($x[$i]);
```

Как уже было отмечено, в нашем файле 11 915 строк, следовательно, будет осуществлено 11 915 итераций цикла. И каждый раз при этом будет вызываться функция `sizeof()` для определения размера массива. А ведь ее можно вызвать всего один раз:

```
$N = sizeof($x);  
for ($i=0; $i<$N; $i++) echo($x[$i]);
```

С помощью такого простого изменения удалось сэкономить еще 0,08 с.

Вот другой характерный пример вызова "лишней" функции:

```
$f=fopen("some_file.txt","r") or die(1);  
while(!feof($f)) $x[]=fgets($f,1000);  
fclose($f);
```


Алгоритм прост — пока не достигнут конец файла, следует читать строку из файла и передавать ее в массив. В этом случае лишней будет функция `feof()`, поэтому ее можно исключить. Код, позволяющий обойтись без `feof()`, уже был приведен ранее:

```
$f=fopen("some_big_text_file.txt","r") or die(1);  
while($x[]=fgets($f,1000));  
fclose($f);
```

Убрав "лишние" вызовы `feof()`, удалось ускорить программу на 10–15 % (при работе с большими файлами). Но в данном случае (чтение файла) лучше использовать функцию `file()`, т. к. при больших размерах файла (по 5–6 Мбайт) разница в 50 %, которой можно добиться с помощью `file()`, будет вполне ощутимой.

П3.8. Контрольные замеры

Напоследок произведем контрольные замеры — ведь нужно узнать результаты нашей оптимизации. Тест будет состоять из двух частей: чтение из файла и вывод на консоль.

Конфигурация компьютера следующая:

- ❑ процессор — AMD Athlon X2 1,9 ГГц;
- ❑ оперативная память — 2,0 Гбайт;
- ❑ жесткий диск — Fujitsu 250 Гбайт (буфер — 8 Мбайт);
- ❑ операционные системы — Windows Vista Home Basic, Linux Ubuntu 10.10;
- ❑ программная среда — PHP 5.3.1.

Почему именно Windows Vista и Linux Ubuntu? Скажу честно: просто эти операционные системы были на момент теста под рукой.

Запуск сценариев производился локально (из консоли), размер текстового файла — 4 Мбайт. Каждый сценарий запускался три раза, потом вычислялось среднее арифметическое полученных результатов.

П3.8.1. Чтение файла

Чтение файла выполняется всеми тремя рассмотренными способами. Для большего удобства созданы три сценария: `s1.php`, `s2.php`, `s3.php`. В сценарии помещены функции вычисления времени выполнения (см. листинг П3.1). Фрагменты кода, помещенные в каждый сценарий, приведены в листингах П3.2–П3.4.

Листинг П3.2. Фрагмент сценария `s1.php`

```
$f=fopen("some_big_text_file.txt","r") or die(1);  
while($x[]=fgets($f,1000));  
fclose($f);
```


Листинг ПЗ.3. Фрагмент сценария s2.php

```
$f=fopen("some_big_text_file","r") or die(1);
while(!feof($f)) $x[]=fgets($f,1000);
fclose($f);
```

Листинг ПЗ.4. Фрагмент сценария s3.php

```
$x = file("some_big_text_file.txt");
```

Результаты выполнения этих сценариев сведены в табл. ПЗ.1.

Таблица ПЗ.1. Результаты первой части тестирования

Сценарий	Время выполнения в Windows, с	Время выполнения в Linux, с
s1	0,18	0,14
s2	0,21	0,17
s3	0,10	0,08

Обратите внимание на то, что чтение файла в Linux происходит быстрее, чем в Windows.

Теперь переходим ко второй части теста — тестированию вывода на экран.

ПЗ.8.2. Вывод содержимого файла

Поскольку нас интересует только вывод текста, то для чтения самого файла использовалась функция `file()`, как самая быстрая. Также мы исключили лишние вызовы функции `sizeof()`.

Вывод также осуществлялся тремя способами:

- ☐ Оператором `echo` без оптимизации.
- ☐ Оператором `echo` с оптимизацией.
- ☐ Функцией `printf()`.

Ранее мы говорили о том, что нужно выносить переменные за пределы текстовых строк, так вот выводом без оптимизации считается:

```
echo "$x[$1]";
```

А вот вывод с оптимизацией:

```
echo $x[$i];
```

Аналогично первой части тестирования были созданы три сценария, но, чтобы избежать повторения имен файлов, их называли `s4.php`, `s5.php` и `s6.php` (листинги ПЗ.5–ПЗ.7).

Листинг ПЗ.5. Фрагмент файла s4.php

```
$x = file("some_big_text_file.txt");
$N = sizeof($x);

for ($i=0; $i<$N; $i++) echo "$x[$i]";
```

Листинг ПЗ.6. Фрагмент файла s5.php

```
$x = file("some_big_text_file.txt");
$N = sizeof($x);

for ($i=0; $i<$N; $i++) echo $x[$i];
```

Листинг ПЗ.7. Фрагмент файла s6.php

```
$x = file("some_big_text_file.txt");
$N = sizeof($x);

for ($i=0; $i<$N; $i++) printf($x[$i]);
```

Результаты выполнения этих сценариев сведены в табл. ПЗ.2.

Таблица ПЗ.2. Результаты второй части тестирования

Сценарий	Время выполнения в Windows, с	Время выполнения в Linux, с
s4	34,3	16,1
s5	34,1	15,8
s6	34,2	17,9

В Windows все три сценария выполняются приблизительно одинаково... медленно. Мне даже пришлось изменить файл `php.ini` (увеличить максимальное время выполнения сценария), чтобы дождаться результата. Понадобилось 34 секунды на вывод текстового файла размером 4 Мбайт.

Вывод занимает львиную долю времени (сравните с табл. ПЗ.1), и нужно отметить, что вывод в Windows изрядно "подтормаживает", особенно на фоне Linux.

Сравните значения второй и третьей колонок табл. ПЗ.2 и вы поймете, почему практически все веб-серверы работают под управлением Linux (UNIX), но не Windows.

На примере Linux можно увидеть, что наши старания не прошли даром. "Оптимизированный" вариант `echo` сэкономил нам 0,3 с по сравнению с обычным (неоптимизированным) и 2,1 с по сравнению с функцией `printf()`.

П3.9. Выводы

В заключение этого приложения перечислим основные выводы.

- ☐ Вынесение переменных за пределы текстовых строк действительно позволяет экономить время выполнения сценария, в чем мы убедились на практике.
- ☐ Помните о "лишних" вызовах функций — иногда можно немного изменить сценарий, исключив "лишние" вызовы функций, как было здесь показано.
- ☐ Если есть выбор операционной системы, которая будет установлена на сервере, предпочтительнее использовать Linux (думаю, не нужно говорить почему).
- ☐ Не всегда так называемые алиасы (aliases), т. е. функции-псевдонимы, работают медленнее исходных функций, что и было показано на примерах.
- ☐ Иногда для оптимизации работы сценария недостаточно просто оптимизировать код, следует поменять весь алгоритм сценария. Например, вместо построчного чтения файла в цикле можно использовать всего одну функцию, которая проделает всю "грязную" работу.

ПРИЛОЖЕНИЕ 4

Описание электронного архива

Электронный архив к книге выложен на FTP-сервер издательства по адресу: <ftp://ftp.bhv.ru/9785977535144.zip>. Ссылка доступна и со страницы книги на сайте www.bhv.ru.

Структура архива представлена в табл. П4.1.

Таблица П4.1. Структура электронного архива

Файл	Описание
Edition_3	Главы 43–45 из третьего издания книги (система продажи недвижимости)
software	Все необходимое программное обеспечение (PHP, MySQL, XAMPP и пр.)
Glava_2	Листинги из главы 2
Glava_5	Листинги из главы 5
Glava_6	Листинги из главы 6
Glava_7	Листинги из главы 7
Glava_12	Листинги из главы 12
Glava_15	Листинги из главы 15, класс watermark
Glava_16	Листинги из главы 16
Glava_18	SQL-код из главы 18
Glava_20	Листинги из главы 20
Glava_21	Листинги из главы 21
Glava_22	Листинги и шаблоны из главы 22
Glava_23	Листинги, шаблонизатор Smarty, документация для Smarty
Glava_24	Листинги из главы 24
Glava_25	Листинги из главы 25
Glava_26	Листинги из главы 26
Glava_28	Листинги из главы 28

Таблица П4.1 (окончание)

Файл	Описание
Glava_29	Сценарий загрузки файлов
Glava_30	Листинги из главы 30
Glava_31	Листинги из главы 31, классы отправки и приема почты
Glava_32	Листинги из главы 32
Glava_33	Листинги из главы 33
Glava_34	Сценарий работы с MP3-файлами
Glava_35	Примеры использования AJAX
Glava_36	Простейший движок сайта, разработанный в главе 36, SQL-код, вспомогательные файлы
Glava_37	Простейшая галерея
Glava_38	Простейшая гостевая книга
Glava_39	Сценарий импорта новостей, парсер SimpleHTML
Glava_40	Счетчик для сайта
Glava_42	Простейшая система голосования
Glava_43	Таблицы базы данных сложного проекта
Glava_44	Листинги файлов сложного проекта
Glava_48	Листинги из главы 48

Предметный указатель

\$

`$_FILES` 330

A

`addAttachment()` 355
`addslashes()` 148
`addHtmlImage()` 354
`addslashes()` 148
Advanced Poll 465
Ajax 392
Apache 31
APOP 363
`array()` 112
`array_count_values()` 131
`array_diff()` 128
`array_diff_assoc()` 128
`array_fill()` 127
`array_fill_keys()` 127
`array_flip()` 131
`array_key_exists()` 130
`array_keys()` 131
`array_map()` 133
`array_merge()` 114
`array_merge_recursive()` 114
`array_pop()` 124
`array_product()` 129
`array_push()` 124
`array_rand()` 130
`array_replace()` 132
`array_reverse()` 121
`array_search()` 132
`array_shift()` 125
`array_slice()` 127
`array_sum()` 129
`array_unique()` 130

`array_unshift()` 125
`array_values()` 131
`array_walk()` 133
`array_walk_recursive` 133
`asort()` 119
AUTO_INCREMENT 438
AWFFull 461
AWStats 462

B

`basename()` 168
Bigmir.net 462
`bin2hex()` 157
Blowfish 151
Breakpoint 316

C

CAPTCHA 90
CGI 69, 73
CGI-приложение 70
`checkdate()` 139
`chop()` 148
`chr()` 150
`compact()` 126
Content-Transfer-Encoding 351
Content-type 351
`convert_cyr_string()` 149, 240
`convert_uencode()` 149
Cookie 298
Cookies 94, 98, 298, 467
`copy()` 169
`count()` 109, 572
`count_chars()` 153
`covert_uudecode()` 149
cracker 522

CRC3 151
crc32() 151
create_watermark() 189
crond 451
crypt() 151
CSV 152, 165, 360
cURL 446
current() 110
CVS 310

D

date() 139
define() 56
DeleteMessage() 363
DES 151
die() 46
DirectAdmin 514
dirname() 172
disable_functions 520
display_errors 520

E

Eclipse 317
end() 110
ereg() 380
ereg_replace() 380
eregi() 382
eregi_replace() 382
error_reporting 519
exif_imagetype() 174
exif_read_data() 175
explode() 153
extract() 126

F

fclose 578
fclose() 163
feof() 162, 163, 578
fflush() 165
fgetcsv() 166
fgets() 164, 192
fgetss() 163, 164
file() 79, 160, 163, 573
file_exists() 168
file_get_contents() 79, 160, 164
file_put_contents() 164
file_uploads() 328
filetime() 170

filemtime() 170
filesize() 170
filetype() 170
FileZilla 31
Flash 566
flock() 171
fopen() 160, 161, 573
fprintf() 156
fputcsv() 167
fputs() 192
fread 160
fread() 162, 192
fsockopen() 191
ftp_chdir() 345
ftp_connect() 345
ftp_exec() 347
ftp_fget() 346
ftp_fput() 346
ftp_login() 345
ftp_mkdir() 346
ftp_nlist() 346
ftp_pasv() 348
ftp_pwd() 346
ftp_rmdir() 346
func_get_args() 201
fwrite() 164, 192

G

get_defined_constants() 57
getenv() 73
getFile() 354
gethostbyaddr() 196
gethostbyname() 196
gethostbyname() 196
getTag() 388
gfets() 573
Google Analytics 462

H

Header() 74, 178
hex2bin() 157
html_entity_decode() 146
htmlentities() 146
HtmlMimeMail() 352
HtmlSpecialChars() 145, 523
htmlspecialchars_decode() 145
HTML-код 49, 403, 404, 414
HTML-файл 424
HTML-форма 77
◇ элементы формы 80

I

iconv() 567
 ID3-теги 386
 imagearc() 184
 imagechar() 180
 imagecharup() 180
 imagecolorallocate() 180
 imagecolorat() 181
 imagecolortransparent() 182
 imagecopyresampled() 186
 ImageCreateFromPng() 178
 imageellipse() 184
 imagefilledarc() 184
 imagefilledellipse() 184
 imagefilledpolygon() 185
 imagefilledrectangle() 184
 ImageGif() 178
 imagepolygon() 185
 imagerectangle() 184
 imageresize() 186
 imagerotate() 185
 ImageString 455
 imagestring() 179
 imagestringup() 181
 imagesx() 186
 imagesy() 186
 imagefttext() 181
 implode() 153
 in_array() 129
 ini_set() 518
 is_array() 87
 is_numeric() 87, 528
 is_string() 87
 isset() 54

J

JavaServer Pages 39
 join() 79, 153
 jpeg2wbmp() 177

K

key() 110
 ksort() 120

L

lastRSS 369
 lcfirst() 147
 LIMIT 242

list() 111
 ListMessages() 363
 log_errors 520
 LogMiner 461
 ltrim() 148

M

mail() 351
 max_execution_time 574
 max_input_time 575
 MD5 151
 md5() 151
 md5_file() 151
 MD5-хэш 442
 memory_limit 575
 Mercurial 311
 Mercury 39
 Microsoft Internet Information Server 69
 MIME 357
 mkdir() 172
 money_format() 157
 move_uploaded_file() 170
 MP3_Id 387
 mSQL 233
 mt_rand() 137
 Multipart-формат 328
 MySQL 402, 403, 407
 mysql_close() 225
 mysql_connect() 224
 mysql_create_db() 230
 mysql_db_query() 230
 mysql_drop_db() 230
 mysql_errno() 226
 mysql_error() 225
 mysql_fetch_array 227
 mysql_fetch_assoc 227
 mysql_fetch_field 227
 mysql_fetch_object 227
 mysql_fetch_row 227
 mysql_num_rows 227
 mysql_pconnect() 225
 mysql_query 227
 mysql_select_db() 224

N

natcasesort() 123
 natsort() 123
 Netscape Communications 513
 next() 110
 nl2br() 145
 number_format() 157

O

Open() 362
opendir() 173
Oracle 233, 305
ord() 150
ORDER BY 240

P

page tagging 459
PCRE 377
PEAR 305, 387
PECL 306
Photoshop 174
PHP 401
PHP Accelerator 258
PHP Expert Editor 29
PHP Mailer 359
php.ini 518
PHP_OS 57
PHP_VERSION 57
phpinfo() 28
PHPMailer 39
phpMyAdmin 410, 426, 452, 458, 471, 472
PHP-код 403
png2wbmp() 177
pos() 111
PostgreSQL 233
preg_match() 197
prev() 110
primary index 210
primary key 210
print() 154
print_r() 115
printf() 155, 575

Q

Quoted-Printable 351

R

range() 128
RAW POST DATA 567
readdir() 173
realpath() 168
reCAPTCHA 90
RegExp 377
register_globals 101
rename() 169, 170
Reply-To 351

Request For Comments, RFC 193
reset() 110
RetrieveMessage() 363
rmdir() 172
rsort() 118
RSS 366
rtrim() 147

S

scandir() 172
secondary index 210
serialize() 100
session identifier 98, 298
session_id() 99, 299
session_is_register() 99
session_name() 99, 299
session_register() 98, 299
session_start() 98, 299
session_unregister() 99, 299
session_unset() 99
SET NAMES 240
setcookie() 95
setCookie() 98, 298
setFrom() 355
setHtml() 354
setlocale() 156
setReturnPath() 355
setSubject() 355
setText() 354
SHA1 151
sha1() 151
sha1_file() 151
shuffle() 121
SID 98, 298
SimpleHTML 448
sizeof() 109, 572
Smarty 257, 258
◇ setup.php 261
◇ конфигурация шаблонов 264
◇ модификаторы переменных 266
◇ переменные 263
◇ подключение к проекту 260
◇ специальные переменные 278
◇ стандартные функции 269
◇ тестирование 260
◇ шаблоны 262
smtp_port 352
socket_set_blocking() 195
sort() 118
sprintf() 155

SQL 207
 SQL injection 522
 sqlite_busy_timeout 237
 sqlite_changes 237
 sqlite_close() 235
 sqlite_column() 236
 sqlite_error_string 237
 sqlite_escape_string 238
 sqlite_exec 238
 sqlite_fetch_all 238
 sqlite_fetch_array() 236
 sqlite_fetch_object 238
 sqlite_fetch_single() 236
 sqlite_field_name 238
 sqlite_has_more 238
 sqlite_has_prev 238
 sqlite_key 238
 sqlite_last_error 238
 sqlite_last_insert_rowid 238
 sqlite_libversion 238
 sqlite_next() 236
 sqlite_open() 235
 sqlite_query() 235, 237
 sqlite_rewind() 236
 sqlite_seek() 236
 sqlite_unbuffered_query 238
 SQL-запрос 409–411
 SQL-модификатор
 ◇ AUTO_INCREMENT 213
 ◇ DEFAULT 213
 ◇ NOT NULL 213
 ◇ PRIMARY KEY 213
 SQL-оператор
 ◇ CREATE 211
 ◇ DELETE 216
 ◇ GROUP BY 219
 ◇ HAVING 221
 ◇ INSERT 214
 ◇ SELECT 215
 ◇ UPDATE 214
 SQL-функция
 ◇ AVG 217
 ◇ COUNT 217
 ◇ MAX 217
 ◇ MIN 217
 ◇ SUM 217
 SSL 513
 str_ireplace() 145
 str_repeat() 144
 str_replace() 144
 str_word_count() 154

strcasecmp() 144
 strcmp() 144
 strcspn() 142, 145
 strip_tags() 147, 523
 stripos() 143
 stripslashes() 148
 stristr() 143
 strlen() 142
 strnatcasecmp() 144
 strnatcmp() 144
 strncasecmp() 144
 strncmp() 144
 strpos() 143
 strrev() 148
 strripos() 143
 strrpos() 143
 strspn() 142, 145
 strstr() 143
 strtolower() 148
 strtotime() 138
 strtoupper() 148
 substr() 144
 substr_count() 154
 Subversion 310
 Sybase, 233

T

time() 138
 timestamp 138, 170
 TLS 513
 tmpfile() 165
 Tomcat 39
 TortoiseHG 311
 touch() 170
 TPL-шаблон 405, 426
 TPL-шаблоны 254
 trim() 148

U

uasort() 121
 ucfirst() 148
 ucwords() 148
 uksort() 121
 unlink() 170
 unserialize() 100, 127
 unset() 54, 113
 upload_max_filesize 328
 upload_tmp_dir 328
 urldecode() 149
 urlencode() 149

usort() 121
UTF-8 157
◇ Преобразование из CP1251 в UTF8 567

V

vfprintf() 156
vprintf() 156

W

Watch-список 316
watermark 187
Webalizer 460

WordPress 444
wordwrap() 149

X

XAMPP 38
XML 368, 392, 566
XSS 522
XSS-атака 523

Z

Zend Accelerator 258
Zend Studio 29

A

Агрегатор 366
Ассоциативный массив 110
Аутентификация 96, 297

Б

Базы данных 209, 403, 409
блок try-catch 289
Блокировка 171

В

Веб-сервер 424
Версия для печати 425
Владелец 158
Вторичный индекс 210
Выражение 57

Г

Голосование 465
Габбер 372, 404

Д

Дескриптор 191
Деструктор 285

Директива

◇ error_reporting 46, 321
◇ register_globals 101
◇ short_open_tag 48
Документ RFC 193

З

Заголовки HTTP 74
Запись 209

И

Имя переменной в PHP 316
Инкапсуляция 282
Инструкция
◇ echo 48
◇ include 251
◇ include_once 251, 253
◇ require 88, 251
◇ require_once 251, 253
Исключительные ситуации 289
Итераторы 293

К

Квантификатор 379
Класс 281
◇ абстрактные методы 287

Класс (прод.)

- ◇ защищенные переменные 286
- ◇ модификаторы 287
- ◇ частные переменные 286

Клонирование 288

Кодировка символов 146, 149, 239

Команда

- ◇ chmod 159
- ◇ mysqladmin 42
- ◇ service 42
- ◇ services.msc 37
- ◇ yum install 42
- ◇ получение вывода команды
в переменную 61

Комментарии 48

Константы 56

Конструктор 284

Л

Локаль 156

М

Макросы 378

Маркировка страниц 459

Массив 109

- ◇ \$_COOKIE 95, 101
- ◇ \$_FILES 101
- ◇ \$_GET 71
- ◇ \$_POST 72
- ◇ \$_REQUEST 72, 101
- ◇ \$_SESSION 99
- ◇ замена в массиве 132
- ◇ поиск в массиве 132
- ◇ сравнение 128

Метод

- ◇ GET 71
- ◇ POST 71

Механизм сессий 96

Н

Наследование 282, 285

Натуральная сортировка 122

О

Объект 282, 287

ООП 281

Оператор

- ◇ -> 284
 - ◇ break 64
 - ◇ continue 64
 - ◇ foreach 110
 - ◇ if 61
 - ◇ instanceof 293
 - ◇ switch-case 64
 - ◇ throw 289
 - ◇ равенства 59
- Операции
- ◇ арифметические 57
 - ◇ логические 58
 - ◇ получение вывода системной команды 61
 - ◇ строковые 60

П

Первичный индекс 210

Переменная 51

- ◇ инициализация 51
- ◇ логическая (булева) 53
- ◇ проверка существования 54
- ◇ уничтожение 54
- ◇ функции проверки типа 53

Переменные

- ◇ типы 52
- ◇ окружения 73

Поле 210

Полиморфизм 282

Права доступа 158

Пространства имен 294

Псевдосимволы 377, 380

Р

Разработка CMS 401

Регулярные выражения 377

Рекурсия

- ◇ непрямая 200
- ◇ прямая 200

С

Свойство 283

Сессии 96, 298

Система

- ◇ Eclipse 317
- ◇ автоматического тестирования
SimpleTest 317

Системы контроля версий 310
Сниффер 513
Создание меню 412
Сокет 191
Спам 434
Список 110
Ссылки 55
◇ символические 55
СУБД 209
суперглобальные массивы 105

Т

Таблица 209
Теги 48
Точка останова 316

У

Указатель \$this 284
Управляющая конструкция 377

Ф

Фильтр
◇ FILTER_SANITIZE_EMAIL 385
◇ FILTER_SANITIZE_URL 385
◇ FILTER_VALIDATE_EMAIL 385

◇ FILTER_VALIDATE_IP 385
◇ FILTER_VALIDATE_URL 385
Фильтры 384
Форма, поиск в Yandex 92
Функции для работы с файлами 192
Функция
◇ die() 48, 317
◇ is_cc_num() 88
◇ is_email() 87, 197
◇ vsprintf() 156
◇ вложенность 199
◇ область видимости 199
◇ проверка доступа к файлу или каталогу 168
◇ проверка типа переменной 53

Ц

Циклы 62

Ш

Шаблонизатор 254, 258, 404, 414, 426
Шаблоны 403

Э

Экземпляр класса 282

nethouse

Создайте сайт для своего бизнеса. Сами!
Просто. Удобно. Бесплатно.

www.nethouse.ru

Никогда еще создание сайта не было таким простым и удобным.
Предоставьте своему бизнесу возможность выйти на новый уровень
и найти новых клиентов.

Majordomo

Хостинг. Домены. Серверы.

(812) 335-35-45 (495) 727-22-78
www.majordomo.ru

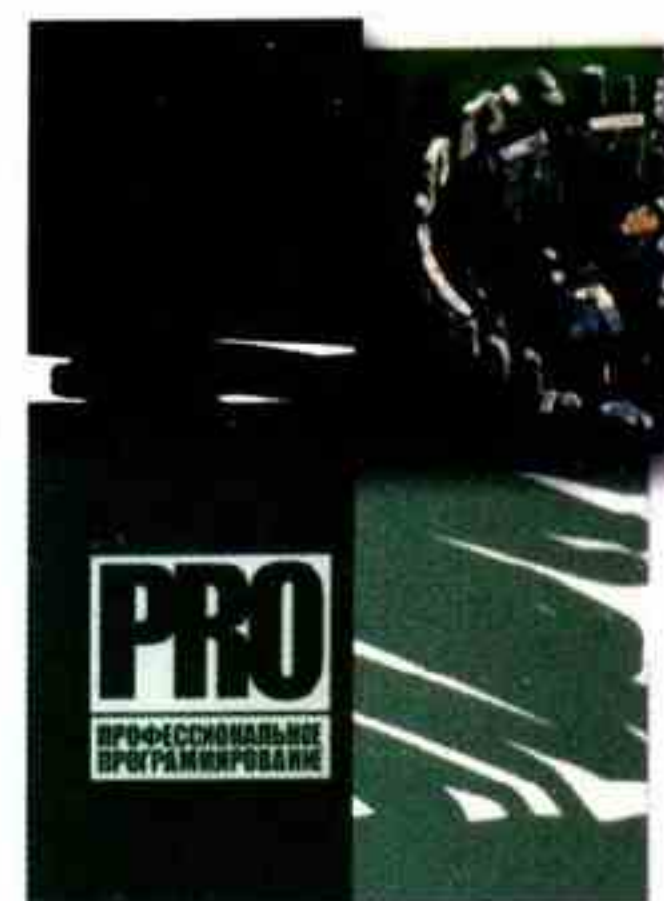
Входит в пятерку крупнейших хостинг-провайдеров России.
На рынке с 2000 года. Полный комплекс услуг,
связанных с размещением Вашего сайта в сети Интернет.

PHP и MySQL

Разработка веб-приложений

5-е издание

Создание профессиональных веб-приложений



Д. Колисниченко



Колисниченко Денис Николаевич — инженер-программист и системный администратор. Автор более 50 книг компьютерной тематики, в том числе «Командная строка Linux и автоматизация рутинных задач», «Профессиональное программирование на PHP», «Android для пользователя. Полезные программы и советы», «Самоучитель Microsoft Windows 8», «Microsoft Windows 8 для пользователей», «Интернет: от "чайника" к пользователю», «Linux. От новичка к профессионалу», «Серверное применение Linux» и др.

На практических примерах описана разработка веб-приложений на языке PHP версии 5.5. Даны начала программирования на PHP: установка и настройка Apache, PHP и MySQL, выбор редактора PHP-кода, синтаксис языка и самые полезные функции PHP. Рассмотрено создание собственного движка сайта и ряда дополнительных модулей — фотогалереи, RSS-граббера, модуля для работы с MP3, гостевой книги, а также применение мощного шаблонизатора Smarty и создание простейшего собственного шаблонизатора. В книге представлено два реальных проекта: универсальная система управления контентом, которая подойдет для построения типичного сайта, и одного из реальных проектов автора. В качестве хранилища данных использованы два сервера — самая современная версия MySQL и «суперлегкий» сервер баз данных SQLite, в том числе рассматривается механизм кэширования MySQL. Показано, как с помощью технологии Ajax добиться обновления данных на странице без ее перезагрузки. Приведено описание программного продукта Zend Guard, описание настройки WAP-сервера и использования редактора SPAW2.

КАТЕГОРИЯ:

Web-программирование



Дополнительные главы, все листинги из книги, а также необходимое программное обеспечение можно скачать по ссылке <ftp://ftp.bhv.ru/9785977535144.zip>, а также со страницы книги на сайте www.bhv.ru.

ISBN 978-5-9775-3514-4



БХВ-ПЕТЕРБУРГ

191036, Санкт-Петербург,
Гончарная ул., 20

Тел.: (812) 717-10-50,
339-54-17, 339-54-28

E-mail: mail@bhv.ru
Internet: www.bhv.ru



PHP и MySQL
Разработка веб-приложений