

Денис Колисниченко

PHP 5/6 и MySQL 6

Разработка Web-приложений 2-е издание

Санкт-Петербург

«БХВ-Петербург»

2010

УДК 681.3.06
ББК 32.973.26-018.2
К60

Колисниченко Д. Н.

К60 PHP 5/6 и MySQL 6. Разработка Web-приложений. —
2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2010. — 560 с.: ил.
+ CD-ROM — (Профессиональное программирование)
ISBN 978-5-9775-0581-9

На практических примерах описана разработка Web-приложений на языке PHP версий 5 и 6. Большая часть кода примеров совместима с обеими версиями PHP, но особое внимание уделено новым функциям PHP 6. Даны начала программирования на PHP: установка и настройка PHP и MySQL, выбор редактора PHP-кода, основы синтаксиса и самые полезные функции PHP. Рассмотрено создание собственного движка сайта и ряда дополнительных модулей — фотогалереи, RSS-граббера, модуля для работы с MP3, модуля продажи недвижимости, гостевой книги, а также применение мощного шаблонизатора Smarty и создание простейшего собственного шаблонизатора. В качестве хранилища данных использованы два сервера — самая современная версия MySQL 6 и "суперлегкий" сервер баз данных SQLite. Показано, как с помощью технологии Ajax добиться обновления данных на странице без ее перезагрузки. Во втором издании описаны новые инструменты для создания сложных проектов PEAR, CVS и PHPUnit. Прилагаемый компакт-диск содержит дополнительные главы, все листинги из книги, а также необходимое программное обеспечение.

Для Web-программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Владимир Красовский</i>
Компьютерная верстка	<i>Натали Смирновой</i>
Корректор	<i>Наталья Першакова</i>
Дизайн серии	<i>Инны Тачиной</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 01.04.10.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 45,15.

Тираж 2000 экз. Заказ №

"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0581-9

© Колисниченко Д. Н., 2010
© Оформление, издательство "БХВ-Петербург", 2010

Оглавление

ВВЕДЕНИЕ.....	1
Что нового во 2-м издании.....	1
Немного истории	2
PHP 6.....	3
MySQL	4
Поддержка читателей.....	5
 ЧАСТЬ I. ТЕОРИЯ	7
 РАЗДЕЛ 1. БЫСТРЫЙ СТАРТ	9
 ГЛАВА 1. УСТАНОВКА НЕОБХОДИМОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ	11
1.1. Нужно ли устанавливать программное обеспечение	11
1.2. Выбор PHP-редактора и FTP-клиента	13
1.3. Установка Apache + PHP + MySQL в Windows	16
1.4. Установка Apache + PHP + MySQL в Linux	21
1.5. Несколько советов	22
 ГЛАВА 2. ПЕРВАЯ PHP-ПРОГРАММА	23
2.1. Ваша первая программа	23
2.2. Запуск PHP-программы.....	24
2.3. Вывод текста без <i>echo</i>	25
 ГЛАВА 3. ОСНОВЫ СИНТАКСИСА PHP	28
3.1. Переменные.....	28
3.1.1. Правила объявления переменных. Имена переменных	28
3.1.2. Типы данных переменных	29
3.1.3. Булевы переменные	31
3.1.4. Операции над переменными.....	32
3.1.5. Ссылки	33
3.2. Константы.....	34

3.3. Выражения и операции	35
3.3.1. Что такое выражение	35
3.3.2. Арифметические операции	36
3.3.3. Логические выражения.....	36
3.3.4. Приоритеты операций	37
3.3.5. Операторы эквивалентности == и ===	38
3.3.6. Операции со строками.....	39
3.4. Условный оператор	40
3.5. Циклы.....	41
3.5.1. Цикл со счетчиком.....	41
3.5.2. Цикл <i>while</i>	42
3.5.3. Цикл <i>do-while</i>	42
3.5.4. Принудительное завершение цикла и пропуск итерации	43
3.6. Оператор выбора <i>switch-case</i>	43

РАЗДЕЛ 2. ПЕРЕДАЧА ПАРАМЕТРОВ PHP-ПРОГРАММАМ..... 45

ГЛАВА 4. МЕТОДЫ GET И POST.....47

4.1. Интерфейс CGI.....	47
4.2. Метод GET	49
4.3. Метод POST	50

ГЛАВА 5. ПРОТОКОЛ HTTP И ИНТЕРФЕЙС CGI.....51

5.1. Специальные переменные окружения CGI	51
5.2. Заголовки протокола HTTP	52
5.3. Коды ответов протокола HTTP	54

ГЛАВА 6. ПЕРЕДАЧА ПАРАМЕТРОВ ПОСРЕДСТВОМ HTML-ФОРМЫ.....55

6.1. Создание простейшей формы и ее обработка в сценарии	55
6.2. Создание пользовательского интерфейса с помощью формы	58
6.2.1. Ввод текста. Теги <i>INPUT</i> и <i>TEXTAREA</i>	60
6.2.2. Зависимые и независимые переключатели	61
6.2.3. Списки выбора	63
6.2.4. Форма для передачи файлов	65
6.2.5. Кнопки	65
6.3. Проверка параметров формы.....	66
6.3.1. Проверка корректности e-mail	67
6.3.2. Проверка правильности номера кредитной карты	68
6.3.3. Удаление лишних пробелов.....	70

ГЛАВА 7. ЗАПОМИНАНИЕ ПАРАМЕТРОВ С ПОМОЩЬЮ COOKIES И СЕССИЙ	71
7.1. Что такое Cookies и как с ними работать	71
7.2. Механизм сессий	74
7.2.1. Сессии и Cookies: преимущества и недостатки	74
7.2.2. Для чего нужны сессии.....	74
7.2.3. Как работает механизм сессий	76
7.2.4. Обход Cookies	78
7.3. Массивы и Cookies.....	78
ГЛАВА 8. ОТДЕЛЬНОЕ СЛОВО О ДИРЕКТИВЕ REGISTER_GLOBS	79
8.1. Почему опасно использовать <i>register_globals</i>	79
8.2. Если <i>register_globals</i> отключена.....	84
8.3. PHP 6 и <i>register_globals</i>	86
РАЗДЕЛ 3. МАССИВЫ И СПИСКИ	87
ГЛАВА 9. ОСНОВНЫЕ ОПЕРАЦИИ НАД МАССИВАМИ И СПИСКАМИ.....	89
9.1. Массив и список. Цикл <i>foreach</i>	89
9.2. Функции <i>list()</i> и <i>array()</i>	91
9.3. Удаление массива	92
9.4. Слияние массивов.....	93
9.5. Функция <i>print_r()</i>	95
ГЛАВА 10. ФУНКЦИИ СОРТИРОВКИ МАССИВОВ.....	97
10.1. Функции для сортировки массивов	97
10.2. Функция <i>sort()</i> — сортировка списка	97
10.3. Функция <i>asort()</i> — сортировка массива по значениям	98
10.4. Функция <i>ksort()</i> — сортировка по ключам.....	99
10.5. Функции <i>array_reverse()</i> и <i>shuffle()</i>	100
10.6. Собственная функция сортировки	101
ГЛАВА 11. ОСОБЫЕ ОПЕРАЦИИ НАД МАССИВАМИ	103
11.1. Добавление и удаление элементов массива	103
11.2. Упаковка переменных в массив и их извлечение.....	105
11.3. Получение части массива.....	106

РАЗДЕЛ 4. ФУНКЦИИ В PHP	107
ГЛАВА 12. ПОЛЕЗНЫЕ СТАНДАРТНЫЕ ФУНКЦИИ	109
12.1. Генератор случайных чисел	109
12.2. Дата и время	110
12.2.1. Кратко о timestamp	110
12.2.2. Функции <i>strtotime()</i> и <i>checkdate()</i>	111
12.2.3. Вывод даты	112
12.3. Математические функции	114
ГЛАВА 13. ФУНКЦИИ ДЛЯ РАБОТЫ СО СТРОКАМИ	115
13.1. Основные строковые функции	115
13.2. Специальные функции замены	117
13.3. Преобразование строки	117
13.4. Функции преобразования кодировок	119
13.5. Функции для работы с отдельными символами строки	120
13.6. Функция <i>md5()</i>	120
13.7. Функция <i>explode()</i> : выделение подстрок	121
ГЛАВА 14. РАБОТАЕМ С ФАЙЛАМИ И КАТАЛОГАМИ	122
14.1. Права доступа в UNIX	122
14.2. Чтение файла	125
14.2.1. Использование функций <i>fopen()</i> и <i>fread()</i>	125
14.2.2. Использование функции <i>file()</i> : построчное чтение файла	128
14.2.3. Чтение всего файла: функция <i>file_get_contents()</i>	130
14.3. Запись файла	130
14.4. Создание временных файлов	131
14.5. Работа с CSV-файлами	131
14.6. Специальные функции для работы с файлами	134
14.6.1. Функции для работы с именами файлов	134
14.6.2. Работа с правами доступа	135
14.6.3. Копирование, переименование и удаление файлов	136
14.6.4. Время доступа к файлу	137
14.6.5. Другие полезные функции	138
14.7. Совместный доступ к файлу	139
14.8. Функции для работы с каталогами	140

ГЛАВА 15. ВЫВОД ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ СРЕДСТВАМИ PHP	142
15.1. Библиотека GD.....	142
15.1.1. Получение информации об изображении.....	143
15.1.2. Конвертирование графических форматов	146
15.1.3. Вывод текста поверх картинки.....	149
15.1.4. Прозрачность.....	152
15.2. Изменение размера изображения	152
15.3. Создание водяных знаков	154
 ГЛАВА 16. РАБОТА С СЕТЕВЫМИ СОКЕТАМИ В PHP.	
СЕТЕВЫЕ ФУНКЦИИ	157
16.1. Еще раз о том, что такое сокет	157
16.2. Использование сокетов	158
16.3. Пример использования сокетов.....	159
16.4. Блокирующий и неблокирующий режимы сокета	163
16.5. DNS-функции	163
 ГЛАВА 17. СОБСТВЕННЫЕ ФУНКЦИИ.....	165
17.1. Зачем нужны собственные функции	165
17.2. Особенности функций в PHP.....	166
17.3. Объявление функции.....	166
17.4. Области видимости функций.....	167
17.5. Вложенность функций.....	168
17.6. Переменное число аргументов	170
17.7. Передача массивов в качестве параметров	171

РАЗДЕЛ 5. СЕРВЕРЫ БАЗ ДАННЫХ MYSQL 6 И SQLITE

ГЛАВА 18. ОСНОВЫ SQL	175
18.1. Немного истории	175
18.2. Преимущества SQL.....	176
18.3. Как выглядят запросы	177
18.4. Что такое база данных.....	177
18.5. Создание таблиц	179
18.6. Добавление записей в таблицу	183
18.7. Обновление записей	184
18.8. Выборка записей.....	185
18.9. Удаление записей.....	186

18.10. Встроенные функции.....	187
18.11. Группировка записей. Сложные запросы.....	188
ГЛАВА 19. ФУНКЦИИ ДЛЯ РАБОТЫ С MYSQL.....	192
19.1. Подключение к серверу MySQL.....	192
19.2. Несколько MySQL-соединений.....	194
19.3. Передача запросов серверу.....	195
19.4. Работа с базой данных. Создание базы данных.....	200
19.5. Функция <i>mysql_real_escape_string(\$content)</i>	200
ГЛАВА 20. АЛЬТЕРНАТИВНАЯ БАЗА ДАННЫХ SQLITE.....	202
20.1. MySQL vs SQLite: что лучше.....	202
20.2. Открытие базы данных.....	204
20.3. Передача запросов.....	205
20.4. Работа с результатом запроса.....	205
20.5. Список PHP-функций для работы с SQLite.....	207
ГЛАВА 21. ПОЛЕЗНЫЕ ПРИЕМЫ ПРИ РАБОТЕ С MYSQL 6	210
21.1. Выбор кодировки.....	210
21.2. Сортировка: вывод новинок. Вывод случайных записей	211
21.3. Постраничный вывод таблицы	212
РАЗДЕЛ 6. ИНСТРУМЕНТЫ ДЛЯ СОЗДАНИЯ СЛОЖНЫХ ПРОЕКТОВ	219
ГЛАВА 22. РАЗРАБОТКА СОБСТВЕННОГО ШАБЛОНИЗАТОРА	221
22.1. Организация файлов и каталогов проекта.....	221
22.2. Выносим параметры в отдельный файл	224
22.3. Подключение дополнительных файлов.....	225
22.3.1. Инструкции <i>include</i> и <i>require</i>	225
22.3.2. Альтернативный способ подключения сценариев	226
22.3.3. Инструкции <i>include_once</i> и <i>require_once</i>	228
22.4. Шаблоны.....	228
ГЛАВА 23. ШАБЛОНИЗАТОР SMARTY	233
23.1. Что такое Smarty	233
23.2. Установка Smarty	234

23.3. Создание <code>setup.php</code>	237
23.4. Разработка шаблонов Smarty	238
23.4.1. Комментарии в шаблонах	238
23.4.2. Переменные в Smarty	239
23.4.3. Файлы конфигурации шаблонов	240
23.4.4. Служебная переменная <code>{Smarty}</code>	241
23.4.5. Модификаторы переменных	243
23.4.6. Стандартные (встроенные) функции Smarty	246
Функции <code>{include}</code> и <code>{insert}</code>	247
Функция <code>{foreach}</code>	247
Функции <code>{if}</code> , <code>{elseif}</code> , <code>{else}</code>	249
Функция <code>{capture}</code>	251
Функция <code>{php}</code>	251
Функция <code>{strip}</code>	251
23.4.7. Пользовательские функции Smarty	252
Функция <code>{assign}</code>	252
Функция <code>{cycle}</code>	252
Функция <code>{fetch}</code>	253
Функции <code>{html_checkboxes}</code> и <code>{html_radios}</code>	253
Функция <code>{html_image}</code>	254
Функция <code>{html_select_date}</code>	254
Функция <code>{html_select_time}</code>	257
Функция <code>{html_table}</code>	257
23.5. Smarty для программиста	258
23.5.1. Специальные переменные	258
23.5.2. Полезные методы класса <i>Smarty</i>	261

ГЛАВА 24. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ262

24.1. Основы ООП	262
24.2. Классы и объекты	263
24.3. Конструкторы и деструкторы класса	266
24.4. Наследование классов. Полиморфизм	267
24.5. Новые возможности PHP 5/6	268
24.5.1. Область видимости членов класса	268
24.5.2. Абстрактные классы и методы	269
24.5.3. Служебное слово <i>final</i>	270
24.5.4. Клонирование объектов	271
24.5.5. Обработка исключительных ситуаций	272
24.5.6. Константы-члены класса	273
24.5.7. Статические члены класса	274

24.5.8. Оператор <i>instanceof</i>	274
24.5.9. Итераторы.....	275
24.5.10. Пространства имен.....	275
ГЛАВА 25. МЕХАНИЗМ СЕССИЙ.....	276
25.1. Для чего нужны сессии.....	276
25.2. Как работает механизм сессий.....	278
25.3. Обход Cookies.....	279
25.4. Сценарий аутентификации.....	280
ГЛАВА 26. ВВЕДЕНИЕ В PEAR.....	285
26.1. Серьезные проекты и PEAR.....	285
26.2. Пример использования класса <i>DB</i>	287
ГЛАВА 27. КОНТРОЛЬ ВЕРСИЙ.....	291
27.1. Выбор системы контроля версий.....	291
27.2. Практическое использование TortoiseHG (Mercurial).....	293
27.3. Просмотр внесенных изменений.....	295
ГЛАВА 28. ТЕСТИРОВАНИЕ РНР-СЦЕНАРИЕВ.....	298
28.1. Программа работает, но не так, как нам нужно.....	298
28.2. "Самодельные" точки останова.....	300
28.3. Система автоматического тестирования.....	300
28.4. Директива <i>error_reporting</i>	305
ЧАСТЬ II. ПРАКТИКА.....	307
РАЗДЕЛ 7. РАЗРАБОТКА ОСНОВНЫХ ЭЛЕМЕНТОВ САЙТА.....	309
ГЛАВА 29. ЗАГРУЗКА ФАЙЛОВ НА СЕРВЕР.....	311
29.1. Что нужно знать о загрузке файлов на сервер.....	311
29.2. Реализация загрузки файла.....	314
29.3. Загрузка нескольких файлов.....	317
29.4. Проблемы при загрузке файлов.....	319

ГЛАВА 30. ИСПОЛЬЗОВАНИЕ FTP-ФУНКЦИЙ.....	321
30.1. Функции для работы с FTP	321
30.2. Примеры использования FTP-функций	325
ГЛАВА 31. ОТПРАВКА И ПРИЕМ ПОЧТЫ	328
31.1. Отправка почты средствами PHP — функция <i>mail()</i>	328
31.2. Отправка писем с вложениями — класс <i>HtmlMimeMail</i>	330
31.2.1. Отправка сообщения.....	330
31.2.2. Проблемы при отправке сообщения	334
31.2.3. MIME-типы.....	336
31.3. Получение писем по протоколу POP3	337
ГЛАВА 32. РАБОТА С RSS: ПОЛУЧАЕМ НОВОСТИ НА САЙТ	342
32.1. Краткие сведения о RSS.....	342
32.2. Формат RSS-файла	343
32.3. Написание сценария импорта новостей	345
32.4. Подключение файла <i>import.php</i> к сайту.....	349
32.5. Создание граббера новостей.....	349
ГЛАВА 33. ПОИСК С ИСПОЛЬЗОВАНИЕМ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ.....	354
33.1. Нужно что-то найти... ..	354
33.2. Язык регулярных выражений <i>RegExp</i>	355
33.3. Управляющие конструкции	357
33.3.1. Квантификаторы	357
33.3.2. Альтернативный оператор <i> </i>	358
33.3.3. Скобки.....	358
33.4. Псевдосимволы	359
33.5. Практическое использование <i>RegExp</i> -функций	359
ГЛАВА 34. РАБОТАЕМ С MP3	364
34.1. Формат MP3	364
34.2. Библиотека <i>PEAR</i>	365
34.3. Вывод ID3-тегов.....	366
34.4. Редактирование ID3-тегов	368
34.5. Удаление тега	369
ГЛАВА 35. ТЕХНОЛОГИЯ AJAX.....	370
35.1. Что такое Ajax	370
35.2. Ваше первое Ajax-приложение.....	371

РАЗДЕЛ 8. ОБЕСПЕЧЕНИЕ БЕЗОПАСНОСТИ САЙТА377**ГЛАВА 36. СОЗДАНИЕ ПРОСТЕЙШЕГО ДВИЖКА САЙТА379**

36.1. Планирование движка	379
36.1.1. Зачем нужно разрабатывать собственный движок	379
36.1.2. Необходимые нам функции движка.....	381
36.1.3. "Принципиальная схема" движка.....	383
36.2. Основные функции движка	385
36.2.1. Разработка TPL-шаблона	385
36.2.2. Файл настроек	386
36.2.3. Основной файл CMS — index.php.....	386
36.2.4. Проектирование базы данных	389
Таблица static	390
Таблица cats	390
Таблица pages	392
36.2.5. Иерархическая структура сайта.....	393
Алгоритм работы меню	393
Сценарий menu.php	395
Вывод содержимого раздела и страницы	402
36.3. Дополнительные функции движка.....	406
36.3.1. Вывод информации из таблицы static	406
36.3.2. Функция вывода содержимого HTML-файла	408
36.3.3. Версия для печати	409
36.4. Где взять листинги этой главы	410

ГЛАВА 37. СОЗДАНИЕ ФОТОГАЛЕРЕИ412

37.1. Постановка задачи	412
37.2. Загрузка изображений на сервер	412
37.3. Вывод галереи	417

ГЛАВА 38. ГОСТЕВАЯ КНИГА420

38.1. Пережиток прошлого?.....	420
38.2. Разработка базы данных и структура гостевой книги.....	421
38.3. Вывод гостевой книги	421
38.4. Добавление записей в гостевую книгу	425
38.5. Сервисный сценарий gb_service.php	429

ГЛАВА 39. ИНТЕГРАЦИЯ ГАЛЕРЕИ LIVEJOURNAL И ВАШЕГО САЙТА.....431

39.1. Что такое Живой журнал	431
39.2. Интеграция фотогалереи LiveJournal и сайта.....	433
39.3. Настройка внешнего вида галереи.....	437

ГЛАВА 40. СОЗДАНИЕ СЧЕТЧИКА САЙТА	439
40.1. Постановка задачи	439
40.2. Файл конфигурации	440
40.3. Разработка таблиц counter и ipaddr	440
40.4. Сценарий counter.php	442
40.5. Сценарий reset_counter.php	447
ГЛАВА 41. СТАТИСТИКА САЙТА.....	448
41.1. Методы сбора статистики	448
41.2. Программы-анализаторы журналов Web-сервера	450
41.3. Системы статистики	451
ГЛАВА 42. ГОЛОСОВАНИЯ (ОПРОСЫ).....	455
42.1. Разработка собственной системы голосования	455
42.2. Разработка сценария poll_form.php	457
42.3. Сценарий poll_process.php	459
42.4. Сценарий poll_results.php	461
РАЗДЕЛ 9. СЛОЖНЫЙ ПРОЕКТ: САЙТ ПО ПРОДАЖЕ НЕДВИЖИМОСТИ	465
ГЛАВА 43. ПОСТАНОВКА ЗАДАЧИ	467
43.1. Функции будущего проекта	467
43.2. Разработка базы данных.....	468
ГЛАВА 44. РАЗРАБОТКА ОСНОВНОЙ ЧАСТИ САЙТА	479
44.1. С чего начать	479
44.2. Реализация основных функций системы	481
44.2.1. Аутентификация пользователей	481
44.2.2. Вывод VIP-объявлений	485
44.2.3. Вывод объявлений об услугах	488
44.2.4. Вывод рекламных баннеров.....	489
44.2.5. Постраничный вывод объявлений	489
44.3. На что обратить внимание	493
ГЛАВА 45. РАЗРАБОТКА ПАНЕЛИ АДМИНИСТРИРОВАНИЯ	501
45.1. Функции панели управления	501
45.2. Управление VIP-объявлениями.....	502

45.3. Массовая отправка электронного сообщения.....	503
45.4. Общие операции с базой данных	504
45.5. Редактирование статей, новостей и контактов	505
РАЗДЕЛ 10. ОБЕСПЕЧЕНИЕ БЕЗОПАСНОСТИ САЙТА.....	507
ГЛАВА 46. SSL-СОЕДИНЕНИЯ.....	509
46.1. Защищаем передаваемые данные.....	509
46.2. Настройка SSL в DirectAdmin.....	510
46.3. SSL-переменные.....	512
ГЛАВА 47. ЗАЩИТА PHP С ПОМОЩЬЮ КОНФИГУРАЦИОННОГО ФАЙЛА	515
47.1. Конфигурационный файл php.ini	515
47.2. Отключение потенциально опасных функций.....	517
47.3. Рекомендованные значения некоторых конфигурационных директив	517
ГЛАВА 48. ЗАЩИТА САЙТА ОТ АТАК.....	519
48.1. Сайт в опасности.....	519
48.2. Два самых распространенных метода взлома.....	520
48.3. Межсайтовый скриптинг	521
48.4. SQL-инъекции	523
48.5. Флуд	527
48.6. Защита форума PHPBB2 от спамеров	528
ПРИЛОЖЕНИЕ. ОПИСАНИЕ CD.....	531
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ	533

Введение

PHP (Hypertext Preprocessor) — один из самых популярных языков программирования, используемый для разработки Web-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 во 2-м издании

Во 2-м издании довольно много изменений, даже не знаю, с чего и начать. Начну с прилагаемого диска. В 1-м издании диска не было, а листинги из книги нужно было скачивать с моего сайта (www.dkws.org.ua). Во 2-м издании диск есть. Кроме листингов на диске вы найдете все необходимое для программирования на PHP программное обеспечение — от Web-сервера и самого PHP до текстового редактора. Подробное описание диска приводится в *приложении*, поэтому не будем сейчас тратить время на его описание.

Раздел 9, в котором рассматривалась интеграция сайта с форумом phpBB (версий 2 и 3), полностью переработан. Теперь в нем рассматривается разработка системы продажи недвижимости — полностью реальный проект, разработанный мною в 2009 г. Полный исходный код системы доступен на прилагаемом диске. Такая перестановка "слагаемых" не означает, что читатели 2-го издания не узнают, как "прикрутить" сайт к форуму или форум к сайту. В каталоге Edition_1 прилагаемого диска вы найдете все главы 1-го издания, которые были удалены из 2-го издания. Также на диск "переехало" описание программы phpMyAdmin (ранее это было единственное приложение).

В 2-м издании книги появился абсолютно новый материал. Кроме мелких изменений в различных главах (вроде создания механизма генерирования уникальных имен при загрузке файлов) было добавлено три новых главы (26–28). В новых главах описывается репозиторий PEAR, система контроля версий РНР-файлов, а также тестирование РНР-сценариев.

И еще хочется сделать одно очень важное замечание. Так сказать, о наболевшем. В 2009 г. на сайте www.bhv.ru одним из читателей был размещен не очень лестный отзыв, в котором говорилось, что в книге очень много опечаток. Отзыв сразу мне показался странным, поскольку все листинги тщательно тестировались как мной лично, так и редактором. Ради "спортивного интереса" при подготовке 2-го издания я проверил еще раз все листинги — ни в одном из них не было найдено ошибок. Если у вас не запускается какой-нибудь сценарий, то это связано либо с неправильными настройками РНР/Apache, либо с тем, что *вы* допустили ошибку в самом сценарии. В любом случае, добро пожаловать на мой форум www.dkws.org.ua, где можно обсудить любые вопросы, связанные с этой книгой, а также с программированием на РНР в целом. Если интересно, то отзывы о моих книгах можно прочитать (и, конечно же, разместить собственный), можно по адресам:

□ <http://www.dkws.org.ua/phpbb2/viewtopic.php?t=3333>;

□ <http://www.dkws.org.ua/phpbb2/viewtopic.php?t=4425>.

Немного истории

В 1994 г. датский программист Расмус Лердорф создал надстройку над Perl/CGI (набор скриптов) 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 С новый интерпретатор и назвал его РНР/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 РНР, такие как автоматическая обработка форм, встраиваемость в HTML и др.

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

В 1998 г. появился PHP 3.0. Именно с этой версии PHP стал 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, так как сценарии выполнялись медленно. Поэтому разработку версии PHP 4 начали с переработки ядра. Начиная с PHP 4 интерпретатор транслирует PHP-код во внутреннее представление, а потом выполняет (а не выполняет сценарий строка за строкой, как PHP 3), благодаря чему существенно повысилась производительность сценариев. Четвертая версия PHP появилась в мае 2000 г. Обновления для этой версии PHP выпускались до конца 2007 г. А с августа 2008 г. поддержка PHP 4 была прекращена.

Пятая версия PHP появилась в июле 2004 г. Было переработано ядро Zend, что увеличило эффективность интерпретатора, добавлена поддержка XML, переработана поддержка ООП. Теперь PHP является полноценным объектно-ориентированным языком программирования, а его объектная модель во многом схожа с моделью Java. В данное время доступна версия 5.3, которая пока еще будет поддерживаться, несмотря на наличие шестой версии PHP.

PHP 6

Итак, что же нового в PHP 6? Прежде всего, в шестой версии значительно улучшена поддержка Unicode. Получить список всех функций, поддерживающих Unicode, можно по адресу http://www.php.net/~scoates/unicode/render_func_data.php.

Поддержка Unicode очень важна. Раньше, когда в знаковый генератор можно было загрузить только 256 символов, поддержка разных языков была ограничена, поэтому и были созданы разные кодировки (иногда даже несколько) для одного языка. Русский язык не стал исключением. Для него было разработано три кодировки: CP866 (DOS), Windows-1251 (или CP-1251, Windows) и KOI8-R (UNIX/Linux). При использовании разных кодировок всегда есть проблема их конвертирования. Например, символ "Б" во всех этих кодировках имеет разный код.

Первая версия Unicode, представленная в 1991 г., позволяла хранить 65 536 символов. Текущая версия Unicode позволяет хранить 2 147 483 648 символов, чего вполне достаточно для хранения символов всех языков, в частности кириллицы.

Когда все программное обеспечение будет переведено на Unicode (а это произойдет в ближайшем будущем), проблема несовместимости кодировок исчезнет. Подробно о Unicode вы можете прочитать по адресу <http://ru.wikipedia.org/wiki/Unicode>.

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, 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 спору, 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.

MySQL

Ни один серьезный Web-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. На сегодняшний день доступна уже шестая версия этого популярного сервера. Кроме MySQL в этой книге мы рассмотрим еще одну альтернативную систему управления базами данных — SQLite. Почему именно

SQLite? На заре пятой версии PHP ходили слухи, что новая (то есть пятая 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 стал SQLite. К счастью, поддержка MySQL не была удалена из PHP 5, и обе последние версии (5 и 6) 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



РАЗДЕЛ 1

Быстрый старт



Установка необходимого программного обеспечения

1.1. Нужно ли устанавливать программное обеспечение

При написании этой книги я вообще сомневался в целесообразности данной 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 то, что PHP-хостинг сейчас стоит очень дешево, как и безлимитный Интернет, то все, что нужно для PHP-разработки, — это какой-нибудь текстовый редактор с поддержкой PHP-синтаксиса.

Все, что вам нужно, — это написать сценарий. В свою очередь, сценарий — это обычный текстовый файл. Как только сценарий будет готов, вам 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 MacOS.

Ранее имело смысл устанавливать на свой компьютер Apache + PHP + сервер баз данных MySQL. Сейчас 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 будут различны, поэтому при переносе сценариев могут возникнуть нюансы. 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 вообще могут возникнуть глобальные проблемы. Буквально вчера (на момент написания этих строк, понятно) я столкнулся со следующей проблемой. У меня был отлично работающий проект, отправляющий прайс-листы клиентам одной из поддерживаемых мною организаций. Сценарий был предельно простым: пользователь загружал новый прайс-лист, а сценарий отправлял его всем клиентам, адреса которых были в базе данных. Для отправки сообщений использовался класс `PHPMailer`. После переноса на другой сервер сценарий перестал запускаться, причем даже не выдавал сообщение об ошибке. Первым делом я запустил функцию `phpinfo()`:

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

Результат меня немного поразил. На сервере была установлена четвертая (!) версия PHP. Учитывая, что уже есть шестая версия, PHP 4 выглядит антиквариатом. Решить проблему удалось путем загрузки класса `PHPMailer` для PHP 4. Хорошо, что такой до сих пор 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 ваша 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, какая версия PHP вам нужна? Все зависит от того, как вы писали свой код. Если вы не использовали функции PHP 5, например `scandir()`, `file_get_contents()` и др., то вам, по сути, хватит PHP 4. А вот если вы использовали пространства имен и другие возможности PHP 6, тогда вам нужна только шестая версия PHP. Все новые версии PHP поддерживают все функции предыдущих версий. Однако не нужно забывать об особенностях PHP 6. Если вы ранее программировали на PHP, то могу поспорить, что вы привыкли работать с включенной директивой `register_globals`. В PHP 6 ее нет, поэтому большую часть ваших сценариев придется переписать с учетом этой особенности. Если вы не программировали до этого на PHP, то не беспокойтесь по этому поводу: все сценарии в этой книге будут написаны с учетом особенностей PHP 6.

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

1.2. Выбор PHP-редактора и FTP-клиента

Из всех PHP-редакторов, которыми я пользовался, мне больше всего 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 на прилагаемом диске в каталоге software.

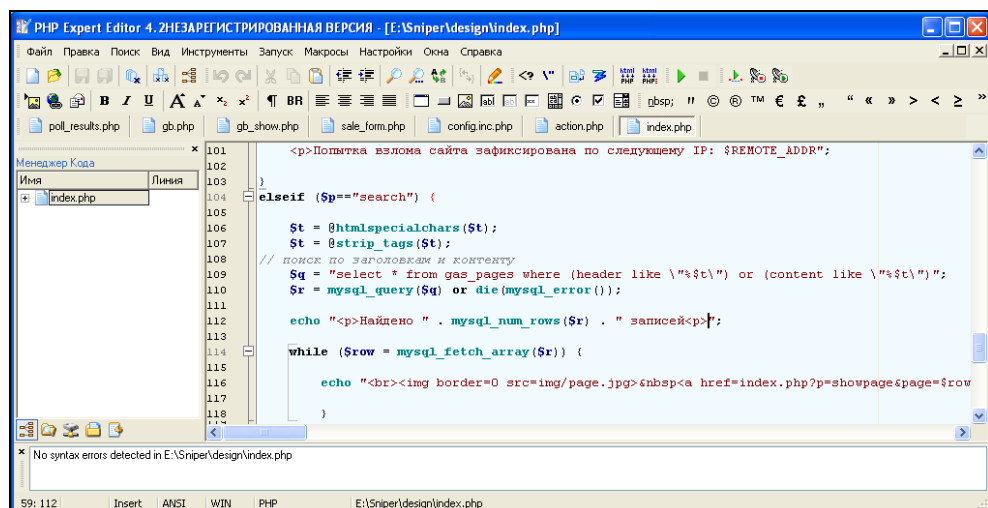


Рис. 1.1. Программа PHP Expert Editor

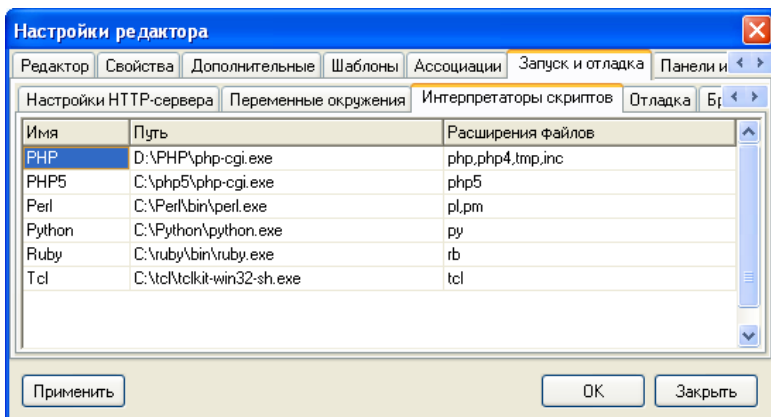


Рис. 1.2. Путь к интерпретатору PHP

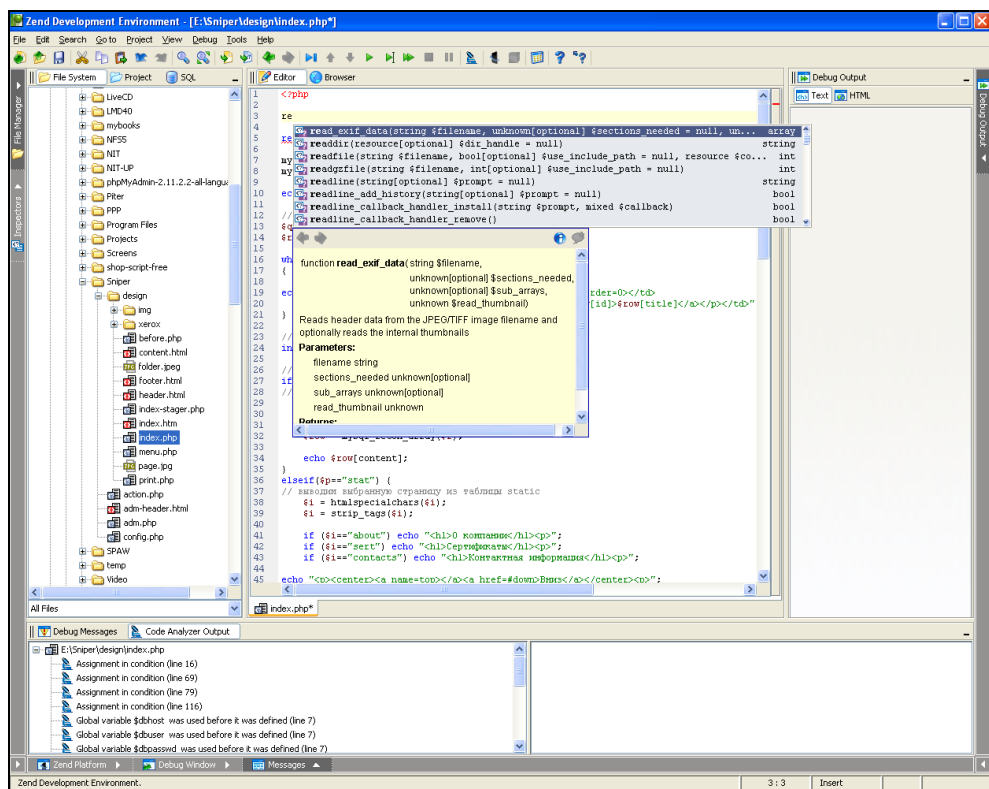


Рис. 1.3. Zend выводит не только имена функций, но и краткую справку по каждой функции

Для этого выполните команду **Запуск | Настройки**, в появившемся окне перейдите на вкладку **Интерпретаторы скриптов** (рис. 1.2) и укажите путь к интерпретатору PHP. Для проверки синтаксиса нажмите клавишу <F7>.

Возможности Zend Studio куда шире (рис. 1.3). Чего стоит только автодополнение кода, когда программа "читает" ваши мысли и дописывает за вас PHP-код. Если вы работали с Delphi и другими продуктами Borland, вы 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. Версия 3.3.1 этого FTP-клиента находится на прилагаемом диске в каталоге software.

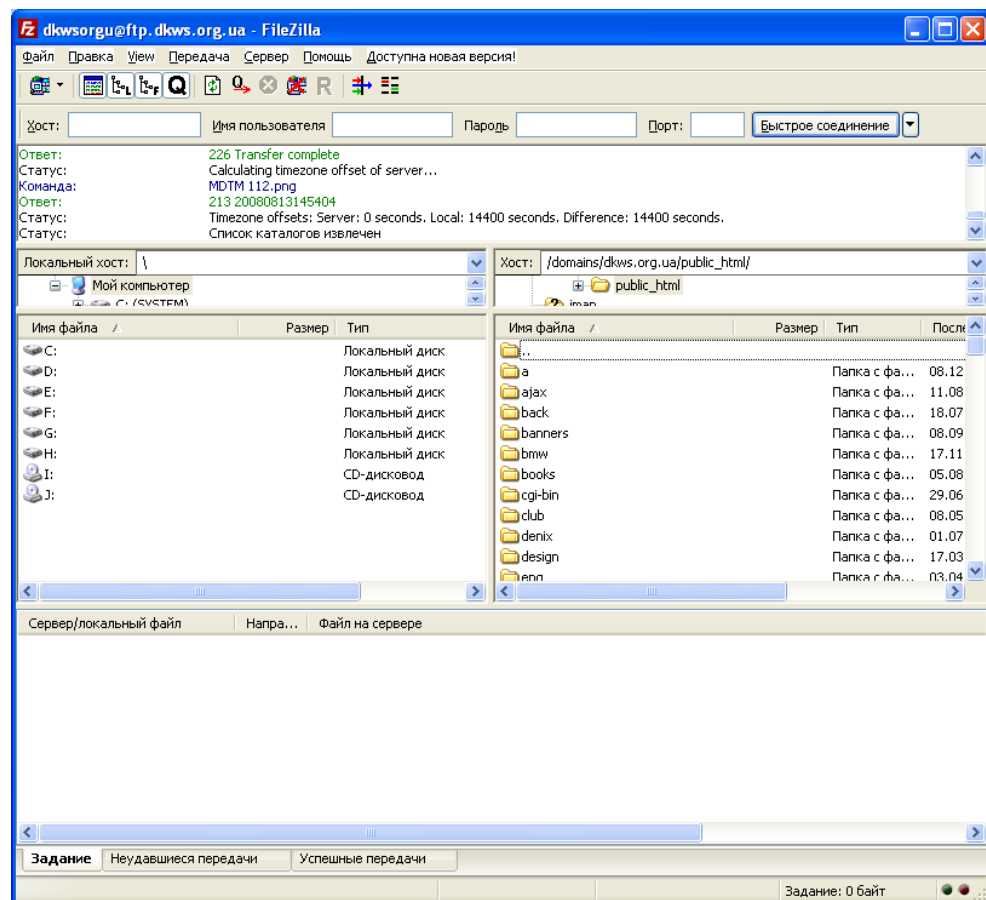


Рис. 1.4. Программа FileZilla

1.3. Установка Apache + PHP + MySQL в Windows

Если вы-таки решили настроить собственный сервер, то приступим к установке. Первым делом вам нужно скачать с сайта **www.apache.org** Web-сервер Apache. Вы скачаете файл `apache_2.2.X-win32-x86-no_ssl.msi`. Запустите его. Процесс установки обычно проходит без осложнений, но мы все же рассмотрим его.

ПРИМЕЧАНИЕ

На прилагаемом диске в каталоге `software` вы найдете обычную (no-SSL) и SSL-версию Apache для Windows. Linux-версии на диске нет, поскольку Apache входит в состав любого дистрибутива Linux, вам нужно только установить его. Несмотря на наличие в каталоге `software` установочных файлов Apache, не забудьте посетить страницу <http://archive.apache.org/dist/httpd/binaries/win32/> — вполне возможно, что к моменту покупки книги вышла новая версия Apache. На диске находится версия 2.2.14, а также модуль `mod_ftp` для Windows.

УСТАНОВКА 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 будет доступен всем пользователям;
- ☐ **only for the Current User** — сервер нужно будет запускать вручную и он будет использовать порт 8080, то есть при обращении к серверу нужно будет указывать порт в URL, например **`http://localhost:8080/`**.

Затем выберите Custom-тип установки 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.

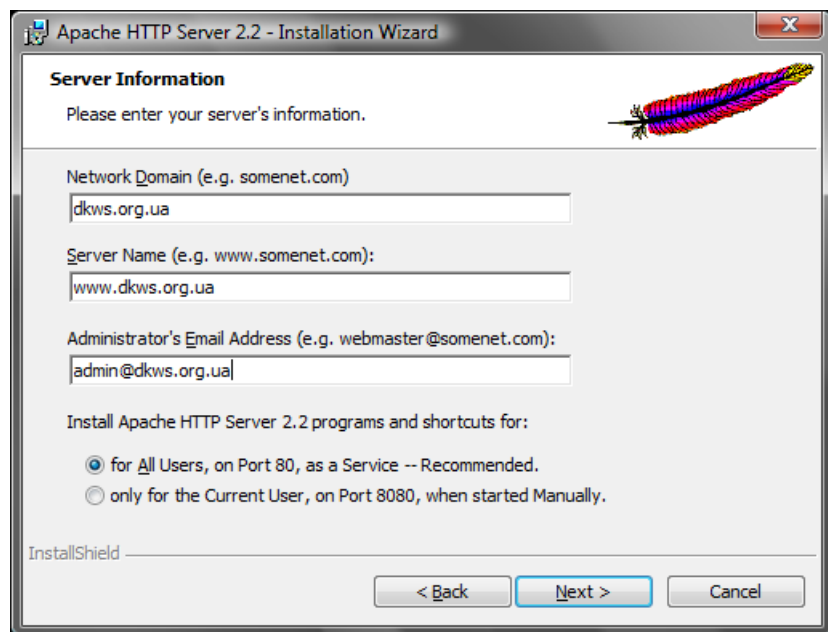


Рис. 1.5. Параметры сервера

После установки Apache запустите браузер и введите следующий URL:

http://localhost

Если вы увидите "It works!" (рис. 1.7), значит, установка Apache прошла успешно.

По умолчанию сервер Apache не поддерживает PHP. Поэтому скачайте с сайта **www.php.net** инсталлятор PHP (он тоже выполнен в виде MSI-пакета) и установите его. Обратите внимание на версию PHP! Если вы хотите использовать последние возможности PHP, нужно скачать шестую версию PHP.

PHP 5 и PHP 6

В каталоге software прилагаемого диска вы найдете установочный файл PHP 5.2. А в каталоге php6 вы найдете архивы как с откомпилированной Windows-версией, так и исходный код PHP 6.

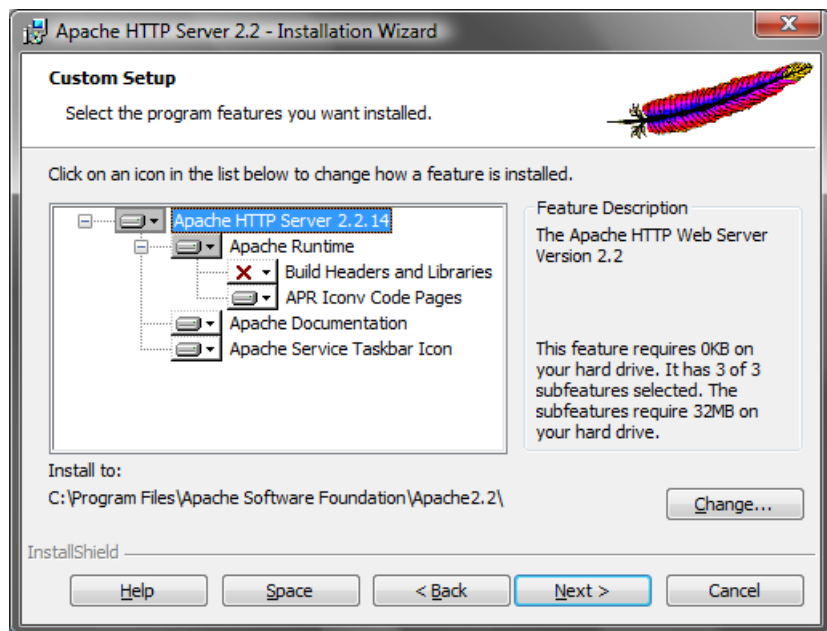


Рис. 1.6. Каталог для установки Apache

При установке PHP обязательно выберите версию вашего Web-сервера. Мы установили версию 2.2, поэтому ее нужно выбрать при установке PHP (рис. 1.8). Далее нужно указать путь к конфигурационному каталогу Apache, чтобы инсталлятор PHP внес необходимые изменения в файлы конфигурации Apache. Если вы установили Apache в D:\WWW, то вам нужно указать следующий каталог: D:\WWW\conf.

Теперь нужно перезагрузить Apache. Для этого выполните команду **Пуск | Выполнить** и введите команду `services.msc`. Отыщите в списке служб Apache 2.2 и выберите команду **Перезапустить службу** (рис. 1.9). В случае возникновения ошибки перезапустите компьютер.

Теперь вам осталось скачать и установить сервер баз данных MySQL (<http://mysql.com/>). Его установка проходит вообще без каких-либо нюансов, поэтому мы не будем ее подробно рассматривать. В каталоге software вы найдете версию 5.1 (на данный момент это рекомендуемая, не экспериментальная версия MySQL) для 32- и 64-битных систем. В главе 2 мы напишем сценарий test.php, тестирующий созданную вами конфигурацию. Данный сценарий нужно поместить в подкаталог D:\WWW\htdocs.

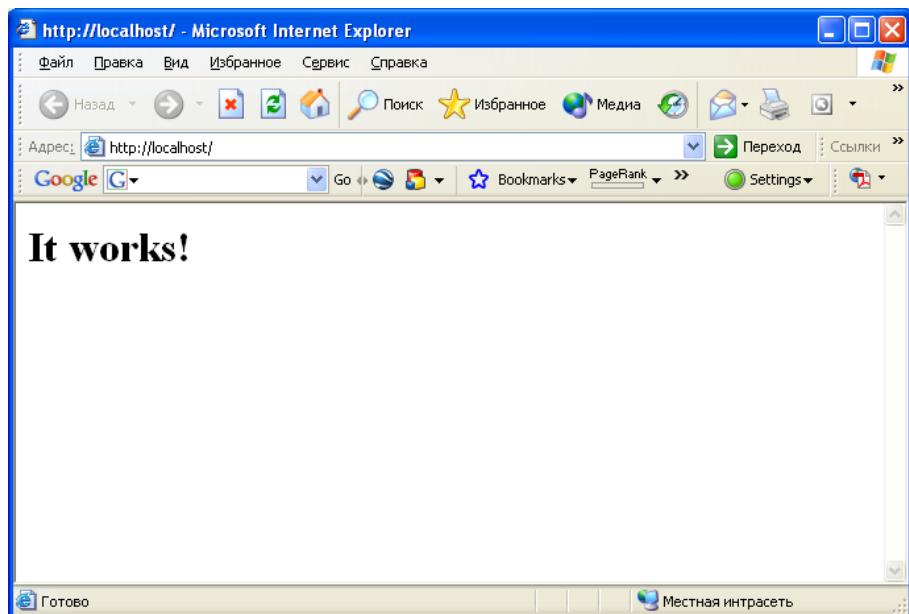


Рис. 1.7. Web-сервер работает!

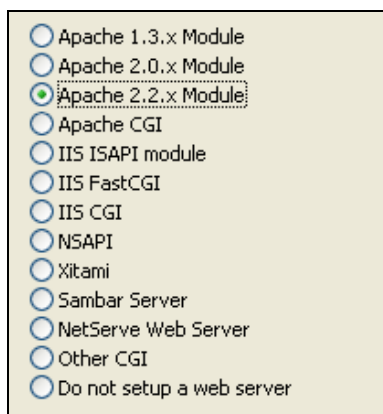


Рис. 1.8. Выбор версии Web-сервера

ПРИМЕЧАНИЕ

На момент написания этих строк MySQL 6 все еще находится на стадии тестирования. Windows-версию MySQL 6 вы найдете на прилагаемом диске в каталоге php6.

Если вам лень устанавливать все отдельно, вы можете установить пакет XAMPP, сочетающий Apache, PHP и MySQL "в одном флаконе". Скачать последнюю версию XAMPP можно по адресу <http://www.apachefriends.org/en/xampp.html>. На прилагаемом диске в каталоге software вы найдете последнюю (на момент написания этих строк) версию XAMPP — 1.7.2.

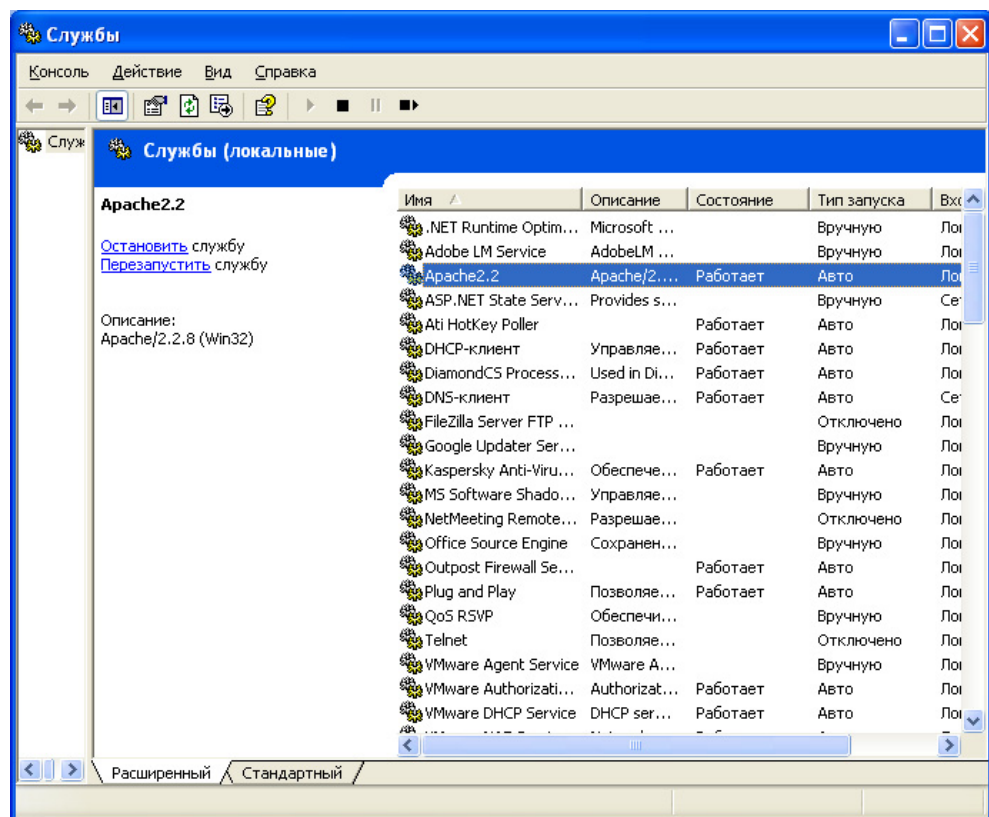


Рис. 1.9. Управление службами Windows

Основное преимущество XAMPP — простота установки и управления всеми сервисами, входящими в его состав (Web-сервер, FTP-сервер и MySQL). Но у XAMPP есть и недостаток — версии программного обеспечения, входящие в его состав, отстают от текущих. Например, на момент написания этих строк пока еще не было XAMPP с PHP 6, а только с PHP 5.

1.4. Установка

Apache + PHP + MySQL в Linux

Процесс установки связки Apache + PHP + MySQL зависит от вашего дистрибутива. В общих чертах вам нужно установить следующие пакеты:

- ❑ `apache2` — Web-сервер Apache;
- ❑ `php6-cli` — интерпретатор PHP, работающий в режиме командной строки (command-line interpreter);
- ❑ `php6-imap` — поддержка протоколов POP/IMAP для PHP;
- ❑ `php6-gd` — поддержка графических функций PHP;
- ❑ `php6-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 используется команда `yum install:`

```
yum install apache2
```

Сразу после установки MySQL нужно задать пароль администратора:

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

Для управления Web-сервером Apache можно использовать 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 используются следующие команды:

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

1.5. Несколько советов

Перед тем как приступить к написанию сценариев, позволю себе дать несколько советов. Если вы при запуске сценария ничего не видите, хотя сценарий должен производить 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`:

```
mysql_connect("localhost","root","");
```

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()`.

ГЛАВА 2



Первая PHP-программа

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";  
?>
```

Разберемся, что мы узнали из этой программы:

- ❑ 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 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 используется инструкция `echo`;
- ❑ после каждого оператора нужно указывать точку с запятой.

2.2. Запуск PHP-программы

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

`http://сервер/test.php`

ПРИМЕЧАНИЕ

Файл `test.php` нужно загрузить именно в корневой каталог Web-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 Web-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 файл `html.php` на основании листинга 2.2.

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-код 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 `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).

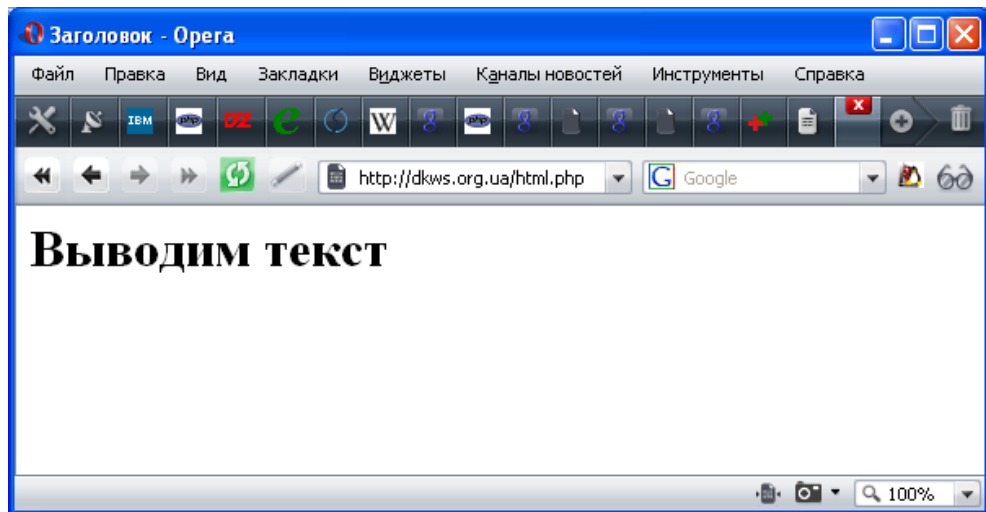


Рис. 2.1. Результат выполнения листинга 2.3

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>";
?>
```

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 передается заказчику, его будет поддерживать Web-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 наперед, 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 RНР

3.1. Переменные

3.1.1. Правила объявления переменных. Имена переменных

Если вы писали до этого на других языках программирования, то наверняка знакомы с понятием переменной, которое присутствует и в RНР. 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 будете работать, будут храниться в переменных.

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

```
var  
    I : integer;
```

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

То, что RНР не требует объявления переменной, не освобождает вас от ее инициализации, то есть от задания переменной начального значения. Инициализация не обязательна, но весьма желательна. Предположим, что вы используете переменную \$i как счетчик с шагом 2, то есть 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, то есть `$a` и `$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, что 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, `$b` станет обычной целой переменной и будет содержать значение переменной `$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 посвящен целый раздел книги;
- ❑ `object` — РНР 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 используются 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

Таблица 3.1 (окончание)

Функция	Описание
<code>gettype(\$x)</code>	Возвращает строку, которая описывает тип переменной: ☐ <code>"boolean"</code> — логическое значение; ☐ <code>"integer"</code> — целое значение; ☐ <code>"double"</code> — вещественное значение; ☐ <code>"string"</code> — строка; ☐ <code>"array"</code> — массив; ☐ <code>"object"</code> — объект; ☐ <code>"resource"</code> — ресурс (например, результат обработки MySQL-запроса); ☐ <code>"NULL"</code> — переменная не имеет значения (не инициализирована), следовательно, невозможно установить ее тип; ☐ <code>"unknown type"</code> — 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, что наш сценарий 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.

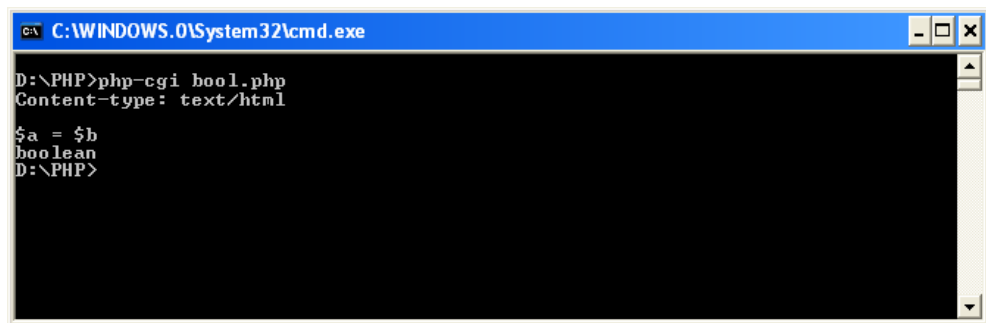


Рис. 3.1. Сравнение логической и целой переменных

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 используется 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 С, то вы 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 будет 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 `$a`. Затем мы объявляем ссылку `$link_on_a` на переменную `$a`. Синтаксис объявления ссылки следующий:

```
ссылка = &переменная;
```

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

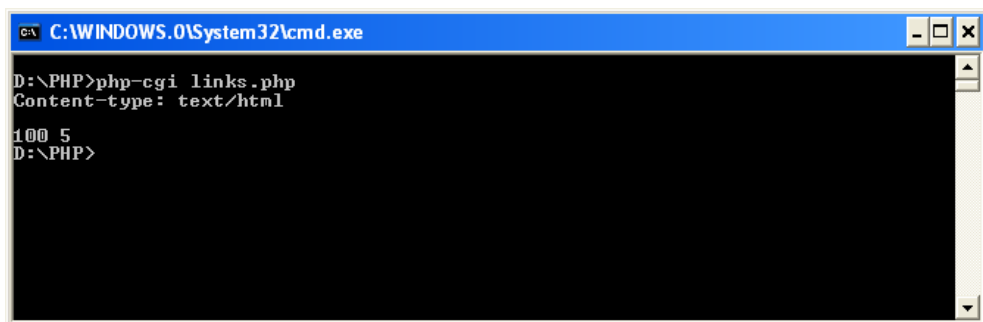


Рис. 3.2. Пример использования ссылок

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 нужно создать переменную, на которую будет указывать ссылка. Затем нужно другой переменной присвоить имя первой переменной. Чтобы создать мягкую ссылку, нужно 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, то есть `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;
- ❑ `PHP_OS` — название операционной системы, под которой выполняется PHP;
- ❑ `PHP_VERSION` — версия PHP.

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 использовать следующие арифметические операции:

- `+` — сложение;
- `-` — вычитание;
- `*` — умножение;
- `/` — деление;
- `A % B` — остаток от деления `A` на `B`.

Кроме того, вы можете 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 используются 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 РНР выполняются в соответствии с приоритетом выполнения операций, 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	=, +=, . = , *=, /=, %=, >=, <=, &=, ^=, =
12	слева направо	
11	слева направо	&&
10	слева направо	
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 Web-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 используется 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 PHP воспринимает `$a` и `$b` как логические. А 0 и пустая строка соответствуют `false`, поэтому он 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
```

Значение переменной `string2` будет 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 под 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 рассмотрим первые три цикла, а последний цикл будет рассмотрен в *главе 9*.

3.5.1. Цикл со счетчиком

Цикл со счетчиком 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 напомним цикл, который будет вызывать эту функцию некое 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, то есть 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 этот оператор. 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;
}
```



РАЗДЕЛ 2

Передача параметров RHP-программам

ГЛАВА 4



Методы GET и POST

4.1. Интерфейс CGI

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

Итак, предположим, что у нас есть Web-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 каталоге cgi-bin мы помещаем самую простейшую программу, работающую со стандартным вводом и выводом. Наша простейшая 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 ей на ее стандартный ввод данные. Следовательно, Web-сервер может запустить нашу программу, передать ей информацию, затем прочесть вывод нашей программы и передать 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 программирова-

ния. 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 URL:

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

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

При получении такого URL наш Web-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. Web-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. Потом программа 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. Если у языка программирования есть средства для разработки CGI-приложений, то разработка такого приложения 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, то есть (в нашем случае):

```
name=denis
```

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

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

Наше приложение несовершенно. Нам предстоит еще сделать разбор параметров, то есть разобрать строку параметров на пары параметр—значение.

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

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

Таблица 5.1 (окончание)

Переменная	Значение
SERVER_NAME	Доменное имя сервера
SERVER_PORT	Порт сервера, обычно 80
SERVER_PROTOCOL	Версия HTTP-протокола

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>";
```

?>

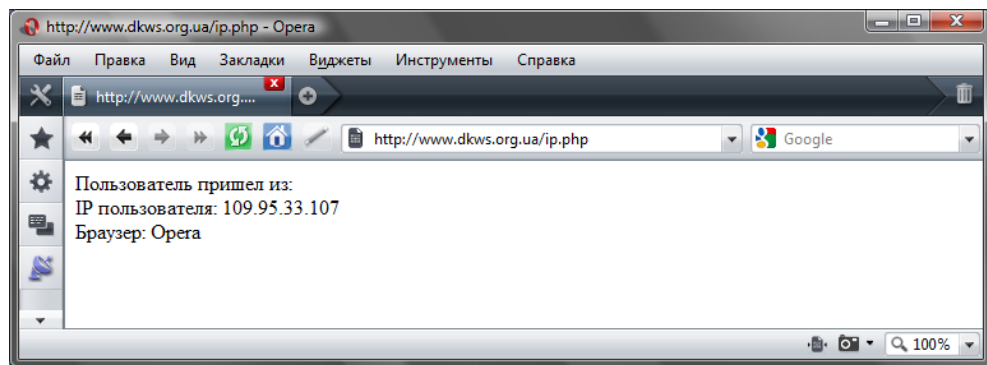


Рис. 5.1. Определение IP-адреса и браузера пользователя

5.2. Заголовки протокола HTTP

Сейчас мы поговорим о заголовках протокола HTTP, которые вам рано или поздно придется использовать в своих сценариях. Например, заголовок `Location`

tion используется для перенаправления пользователя на другую страницу, а заголовки GET/POST используются 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. Данный заголовок используется редко, и если вы не собираетесь разрабатывать собственный браузер, вы не будете его использовать
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 <i>главе 7</i>
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

Таблица 5.2 (окончание)

Заголовок	Значение
Pragma	Используется для всевозможных целей, чаще всего для запрета кэширования страницы: Pragma: no-cache
Referer	Содержит адрес сайта, откуда пришел пользователь на ваш сайт
User-Agent	Содержит информацию об используемом браузере, например: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) Opera 7.54 [en]

5.3. Коды ответов протокола HTTP

Наверняка каждый пользователь Интернета сталкивался с ошибкой 404, которая генерируется, если запрашиваемая страница не найдена. Вообще, это не ошибка, а обычный ответ сервера, который сообщает вам, что запрошенная страница не существует. 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 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.

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

Если вам интересно, то коды 1xx называются информационными — они передают различную информацию, которая так или иначе относится к запросу, а коды 3xx — это коды перенаправления.



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, ровно как и способы обработки данных (что может делать программа с данными). Нас сейчас интересуют входные данные. Чтобы получить результат, нужно раздобыть входные данные. 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 говоря, и эти параметры необязательны, 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 передаёт параметры
method	Определяет метод передачи параметров (GET или POST)
enctype	Тип кодирования данных формы. Обычно этот параметр определять не нужно. Только если собираетесь передавать файлы, тогда нужно установить значение <code>multipart/form-data</code> для этого параметра
name	Задаёт имя формы, через которое можно обращаться к её 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, как на рис. 6.1.

Как только пользователь введёт своё имя и нажмёт кнопку **ОК**, форма методом GET передаст параметр `username` сценарию `hello.php`.



Рис. 6.1. Форма в окне браузера

Теперь поговорим о сценарии `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 вы передаете параметры, или же просто хотите сделать более универсальный сценарий, чтобы он работал с обоими методами передачи параметров, используйте 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 ли 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'];  
  
?>
```

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

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

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

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

Используемая нами конструкция будет работать на самых древних версиях PHP, а вот функция `file_get_contents()` появилась только в PHP 5. Учитывая, что на современных хостах используются последние версии PHP, вы можете использовать функцию `file_get_contents()`.

Можно было форму вывести 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 будет 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 **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">
```

Но 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 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">';
```

Поэтому, если значения HTML-параметров не содержат пробелов, от использования кавычек можно отказаться. 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
<code>type</code>	Задаёт тип текстового поля: ☐ <code>text</code> — обычный текст; ☐ <code>hidden</code> — скрытое поле, содержит значение, но не отображается на форме, и пользователь не может его изменить; ☐ <code>password</code> — поле для ввода пароля
<code>value</code>	Задаёт значение по умолчанию
<code>size</code>	Размер поля (в знаках)
<code>maxlength</code>	Максимальная длина строки, которую можно ввести в поле
<code>name</code>	Имя, по которому можно обращаться к элементу ввода в сценарии

Рассмотрим небольшой пример:

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

Данный тег создаёт поле для ввода пароля (введённый текст будет заменяться звёздочками) длиной восемь знаков и максимальной длиной значения — 16 символов. То есть наше поле — небольшое, оно отображает только 8 символов (8 звёздочек, точнее), хотя в него можно ввести до 16 символов.

Теперь поговорим о теге `TEXTAREA`. Тег `INPUT` позволяет ввести всего одну строку, а тег `TEXTAREA` позволяет 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` вы обязаны его указать (`/TEXTAREA`), 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 используется 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 полу-  
чать рассылку
```

```
<input type=checkbox value=yes name=email checked> Да, я хочу полу-  
чать 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";
```

Теперь перейдем к зависимым переключателям. Переключатели, изображенные 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:

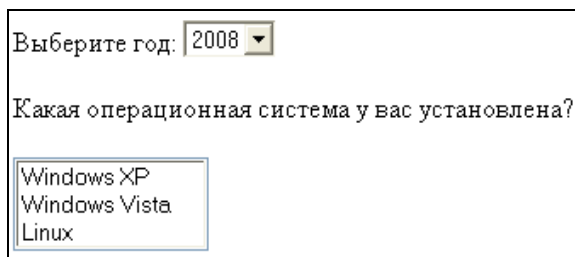
```
// проверяем, был ли передан параметр  
if (isset($_REQUEST['sex'])) {  
    // параметр передан, проверяем значение  
    if ($_REQUEST['sex'] === "male") echo "Мужчина";  
    else echo "Женщина";  
}
```

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=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>
```



Выберите год: 2008 ▼

Какая операционная система у вас установлена?

- Windows XP
- Windows Vista
- Linux

Рис. 6.3. Два списка выбора

Первый список не представляет ничего интересного — это обычный выпадающий список. Форма передает нашему сценарию параметр `year` со значе-

нием, 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>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 тегам `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, 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 есть две квадратные скобки. Это означает, что мы передаем массив параметров, то есть 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 — кнопку **Submit**. Существуют два варианта кнопок: 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 получилось такого, что пользователь запустил сценарий без параметров.
2. Если проверять значения всех параметров вам не хочется (тем более, что их может быть очень много, что отрицательно скажется на читабельности сценария), тогда поможет инициализация переменных, то есть установка параметров по умолчанию. Этот способ будет даже лучше. Даже если форма не передаст параметр, у вас будут параметры по умолчанию — 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 другое (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

Сейчас мы 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, то есть если пользователь введет адрес **user@domain.com**, то функция сообщит, что адрес правильный, поскольку он соответствует правилам написания электронных адресов. Но 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;  
    }  
}
```

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

- ❑ 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 карты все больше входят в обиход, и даже у нас теперь не боятся оплачивать покупки в интернет-магазинах кредиткой. Обработкой транзакции, то есть 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 опять 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  
    $cards = array(  
        '/^4\d{12}(\d\d\d){0,1}$/' => '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;
}
}
```

В начале функции мы задаем массив регулярных выражений, позволяющих определить тип кредитной карточки — VISA или MasterCard (если вы знаете форматы номеров карточек других систем, вы 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`.

Затем 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 "mastercard").

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:

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

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



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, то есть 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, то есть 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 используется 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.

Давайте поговорим о 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, то есть получается, что перед выполнением `setcookie()` уже был 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 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

Сейчас 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 7.1), выводящий форму регистрации и проверяющий правильность имени пользователя и пароля, если пользователь с помощью формы передал их.

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');
}

?>
```

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 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 ...])
```

Данная функция принимает переменное число аргументов, то есть 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, то есть до обращения к функции `session_start()`:

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

```
session_start();
```

Если имя сессии не задано (то есть 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 делает 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, нужно использовать функцию `unserialize()`:

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



Отдельное слово о директиве *register_globals*

8.1. Почему опасно использовать *register_globals*

В файле конфигурации `php.ini` есть 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 `$page` со значением `main`, которая будет доступна в любой части сценария `index.php`. Это очень удобно, поскольку не нужно больше анализировать массивы `_GET` и `_POST` на предмет переданных данных. Если же директива `register_globals` выключена (`register_globals = Off`), обратиться к переменной `$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. Многие хостеры отключают эту директиву, сославшись на опасность ее использования, причем не все могут объяснить, в чем именно заключается эта самая опасность. А у пользователей хостера вместо чувства защищенности возникает раздражение, поскольку нужно переписывать уже готовые сценарии. Хорошо, если сценарии были написаны профессионально и ориентированы на работу без `register_globals`. А вот если сценарии писались с максимальным комфортом для программиста, тогда все их нужно будет переработать. Стоит ли оно того? Ради чего придется потратить целый день на модификацию уже готовых сценариев?

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

```
<?php
```

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

```
?>
```

Запустите данный 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 создается 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 `$_GET`. Ведь локальная переменная `$a` не будет создана, следовательно не будет и ссылки на нее в массиве `$GLOBALS`.

Когда мы знаем, как работает `register_globals`, можно поговорить о 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  
$pswd = GetPasswordFor($login);  
  
// проверяем пароль пользователя  
if ( md5($password) === $pswd ) $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. Затем 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, тогда пользователь видит надпись "Аутентификация прошла успешно!", а далее он получает доступ к своей панели управления.

Наш сценарий содержит огромную дыру в системе безопасности. Смотрите, если имя пользователя и пароль передаются методом GET, то 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 с методами `GET` и `POST`. То есть если вы просто укажете адрес **`http://localhost/login.php?auth=1`**, наш метод "обхода" аутентификации сработает (переменная `$auth` будет определена со значением 1, что соответствует логическому значению `true`).

Посмотрим, как будет работать наш сценарий:

1. Он попытается вычислить пароль пользователя.
2. Затем он проверит пароли.
3. После отобразит надпись "Аутентификация прошла успешно!" Да, сценарий не устанавливал переменную `$auth` в `true`, за него это сделала директива `register_globals`.

Если бы `register_globals` была выключенной, тогда переменную `auth` нельзя было передать указанным способом. Следовательно, переменная `$auth` приняла бы значение `true` только в случае правильного имени пользователя и пароля. Конечно, в этом случае не были бы переданы и переменные `$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!";  
    ...  
}
```

Сейчас вы полностью согласны с тем, что `register_globals` опасна, но наверняка у вас возник 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, чат, а используем уже готовые продукты. А они доступны всем. Злоумышленник может скачать тот же форум, проанализировать его на предмет подобных уязвимостей и взломать ваш форум. Вот так-то вот.

Далеко не все хостеры отключают `register_globals`. А ведь даже если ваш сценарий не будет ориентирован на `register_globals` (см. листинг 4.2), но эта директива будет включена, все равно будет 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, поэтому до версии 4.3 все писали сценарии, ориентируясь на то, что данная директива будет включена. 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 ваши сценарии уже не работают?

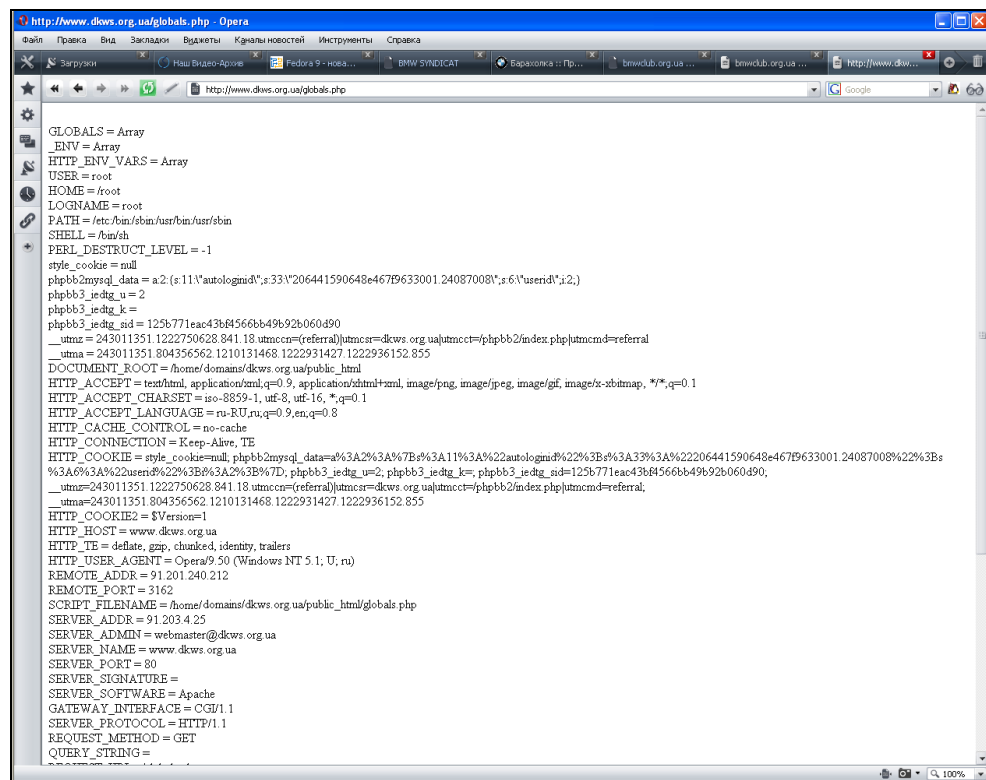
Если директива `register_globals` выключена, то получить доступ к внешним переменным (которые пользователь не определил явно в сценарии) возможно через ассоциативные массивы `HTTP_***_VARS` (`HTTP_POST_VARS`, `HTTP_GET_VARS` и т. д.) или через суперглобальные массивы (табл. 8.1). Немного удобнее использовать суперглобальные массивы. Да и разработчики PHP рекомендуют отключить `register_globals` и работать с переменными только через суперглобальные массивы. Все нужные вам суперглобальные массивы представлены в табл. 8.1. В этой таблице также приведены соответствующие суперглобальным ассоциативные массивы.

Таблица 8.1. Суперглобальные массивы

Название массива	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 Web-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

Таблица 8.1 (окончание)

Название массива	Ассоциативный массив	Описание
<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

Рис. 8.1. Массив `$_GLOBALS`

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 очень просто:

```
$_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:

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

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

```
foreach($HTTP_POST_VARS as $key => $value)
    eval("$".$key."='".$value.'";");
foreach($HTTP_GET_VARS as $key => $value)
    eval("$".$key."='".$value.'";");
```

Что делает код, думаю, особо комментировать не нужно. Он просто устанавливает все переменные из массивов `$HTTP_POST_VARS` и `$HTTP_GET_VARS`.

8.3. PHP 6 и *register_globals*

Все сказанное ранее верно для PHP версий 4 и 5. Я прекрасно понимаю, что PHP 6 относительно недавно появился на свет, поэтому на серверах хостинг-провайдеров все еще установлена пятая версия PHP.

Начиная с версии PHP 4.2 директива `register_globals` по умолчанию отключена. Но многие ее включают, невзирая на все предупреждения. Да, с ней намного удобнее. Но в PHP 6 данная директива вообще отсутствует. Она исключена из состава PHP 6. Теперь доступ к переменным осуществляется только через суперглобальные массивы.



РАЗДЕЛ 3

Массивы и списки

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. Как вы уже убедились, 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, то есть индекс уникален для каждого элемента.

Давайте создадим массив:

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

Массив `$os` содержит пять элементов, нумерация элементов 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()`. Данная функция возвращает количество элементов массива.

РНР обладает очень удобной функцией: при добавлении в массив нового элемента совсем не обязательно указывать его индекс, РНР будет использо-

вать следующий свободный индекс. Приведенный массив можно было бы создать так:

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

Нумерация элементов начинается с нуля.

В PHP есть несколько типов массивов: ассоциативные массивы и списки.

Список — это обычный массив, где в качестве индексов элементов выступают числа. В нашем случае массив `$os` является списком.

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>";
```

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>";
```

Но цикл `foreach` использовать гораздо удобнее.

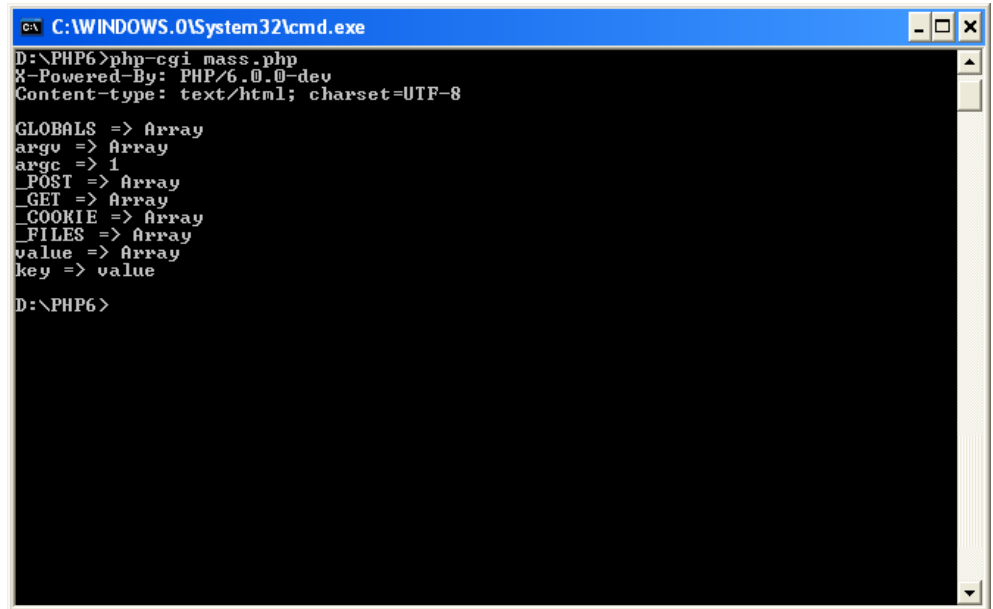
Вот пример перебора суперглобального массива `$GLOBALS`, содержащего все глобальные переменные текущего сценария:

```
<?php

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

?>
```

Вывод сценария изображен на рис. 9.1.



```
C:\WINDOWS.0\System32\cmd.exe

D:\PHP6>php-cgi mass.php
X-Powered-By: PHP/6.0.0-dev
Content-type: text/html; charset=UTF-8

GLOBALS => Array
argv => Array
argc => 1
_POST => Array
_GET => Array
_COOKIE => Array
_FILES => Array
value => Array
key => value

D:\PHP6>
```

Рис. 9.1. Выбор массива `$GLOBALS`

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()` используется для создания массивов, как списков, так и ассоциативных:

```
// пример создания списка
```

```
$os = array("UNIX", "Linux", "Windows XP", "Windows Vista",  
"MacOS");
```

```
// ассоциативный массив
```

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

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

Элементы массива тоже могут быть массивами, то есть 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"];
```

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";
```

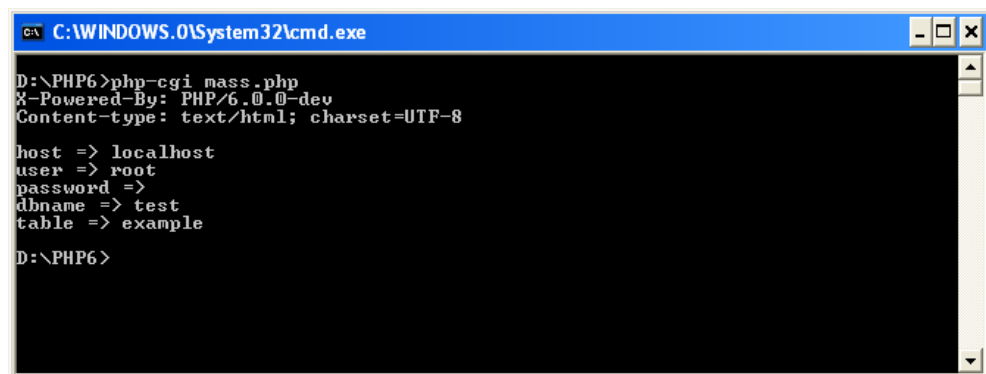
Вывод изображен на рис. 9.2.

А теперь попробуем объединить два списка, например:

```
$A1 = array(1, 2, 3);
```

```
$A2 = array(4, 5, 6);
```

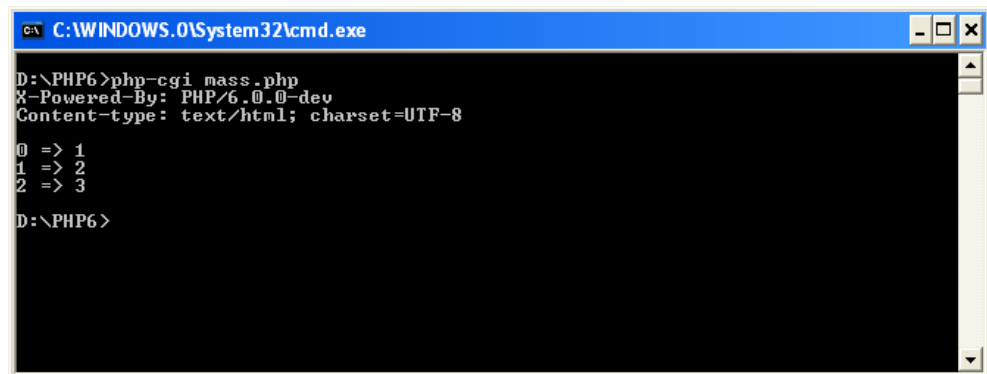
```
$A3 = $A1 + $A2;
```



```
C:\WINDOWS\system32\cmd.exe  
D:\PHP6>php-cgi mass.php  
X-Powered-By: PHP/6.0.0-dev  
Content-type: text/html; charset=UTF-8  
  
host => localhost  
user => root  
password =>  
dbname => test  
table => example  
D:\PHP6>
```

Рис. 9.2. Вывод массива `$DBINFO`

Однако результат нас удивит (рис. 9.3). Наш массив \$a3 вместо значений 1, 2, 3, 4, 5, 6 будет содержать только значения 1, 2, 3.



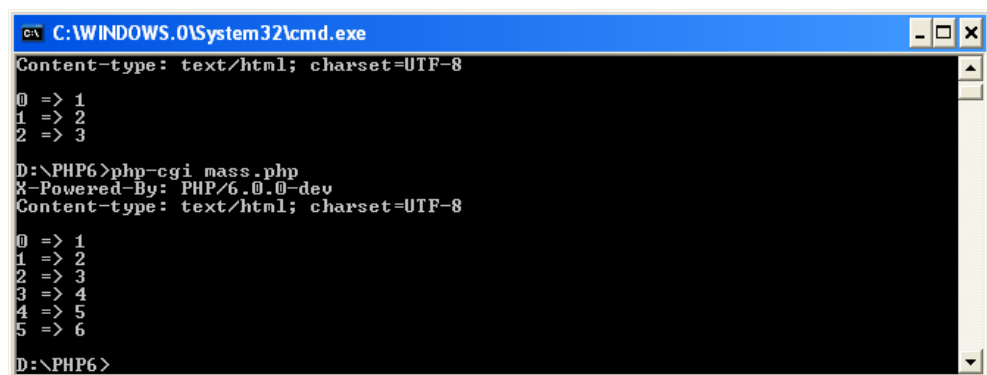
```
C:\WINDOWS.0\System32\cmd.exe
D:\PHP6>php-cgi mass.php
X-Powered-By: PHP/6.0.0-dev
Content-type: text/html; charset=UTF-8

0 => 1
1 => 2
2 => 3

D:\PHP6>
```

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 вот.



```
C:\WINDOWS.0\System32\cmd.exe
Content-type: text/html; charset=UTF-8

0 => 1
1 => 2
2 => 3

D:\PHP6>php-cgi mass.php
X-Powered-By: PHP/6.0.0-dev
Content-type: text/html; charset=UTF-8

0 => 1
1 => 2
2 => 3
3 => 4
4 => 5
5 => 6

D:\PHP6>
```

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

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 использовать `array_merge()`:

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

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

9.5. Функция *print_r()*

Иногда нужно вывести массив в отладочных целях — просто чтобы посмотреть его элементы.

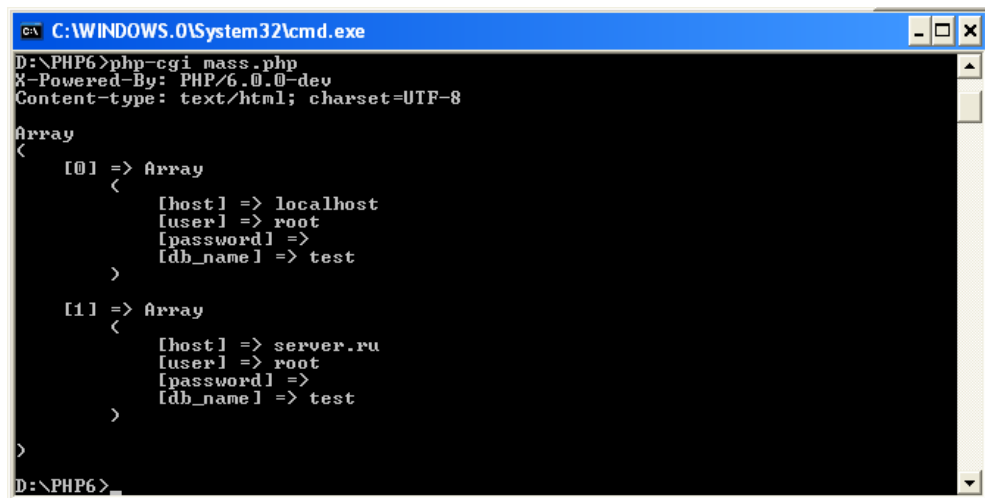


Рис. 9.5. Вывод массива с помощью функции `print_r()`

В этом случае некогда 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 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.



Функции сортировки массивов

10.1. Функции для сортировки 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 рассмотрим лишь самые востребованные, а также рассмотрим создание собственной функции сортировки.

10.2. Функция `sort()` — сортировка списка

Функция `sort()` используется для сортировки обычных массивов, то есть списков. Прототип этой функции выглядит так:

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

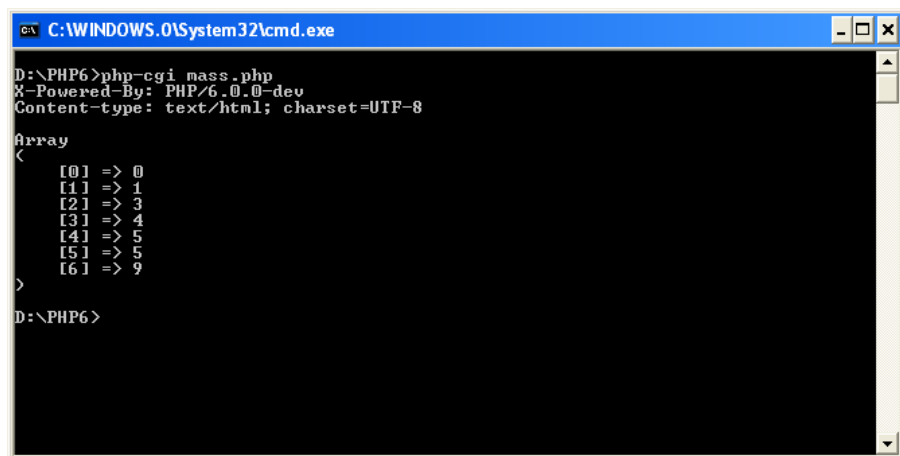
Рассмотрим пример использования этой функции:

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

Результат сортировки представлен на рис. 10.1.

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.



```
C:\WINDOWS.0\System32\cmd.exe

D:\PHP6>php-cgi mass.php
X-Powered-By: PHP/6.0.0-dev
Content-type: text/html; charset=UTF-8

Array
(
    [0] => 0
    [1] => 1
    [2] => 3
    [3] => 4
    [4] => 5
    [5] => 5
    [6] => 9
)

D:\PHP6>
```

Рис. 10.1. Результат сортировки массива

- ❑ 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).

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 следующий массив. 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. Функция `krsort()` выполняет сортировку по ключам в обратном порядке.

Рассмотрим небольшой 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 будет 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?

В этом случае 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, она 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`).

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 используется 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 следующих переменных:

```
$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 используется 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, то есть `$host`, `$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 сериализовать массив, то есть 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.



РАЗДЕЛ 4

Функции в РНР



Полезные стандартные функции

12.1. Генератор случайных чисел

В Web-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) . " ";
```

Сейчас мы 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
$мax = count($мotd);

// получаем случайное число
$х = mt_rand(0, $мax);

// выводим случайную фразу
echo $мotd[$х];

?>
```

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 г. С одной стороны, такой формат не очень удобен для понимания пользователем, 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 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 использования этой функции.

Таблица 12.2. Примеры использования функции `strtotime()`

Функция <code>strtotime()</code>	Значение <code>timestamp</code>
<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 <code>timestamp</code> будет добавлено 86 400 секунд)
<code>strtotime("+1 week")</code>	Неделя спустя с текущего момента (к текущему <code>timestamp</code> 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 вторник
<code>now</code>	Текущая дата

Данная функция довольно удобна: вам не нужно высчитывать `timestamp` вручную, а нужно только указать дату или ее смещение. Конечно, если вы не знаете английский, то вам будет полезно знать 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. 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. Для проверки даты лучше использовать функцию `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 — 1–12, день — в зависимости от выбранного месяца (функция также учитывает високосные годы).

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, используется 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 нужно процитировать 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
<code>%</code>	Час, 12-часовой формат, без лидирующего нуля	1–12
<code>%G</code>	Час, 24-часовой формат, без лидирующего нуля	0–23
<code>%h</code>	Час, 12-часовой формат, с лидирующим нулем	01–12
<code>%H</code>	Час, 24-часовой формат, с лидирующим нулем	00–23
<code>%a</code>	До/после полудня (нижний регистр)	am или pm
<code>%A</code>	До/после полудня (верхний регистр)	AM или PM
<code>%i</code>	Минуты с лидирующим нулем	00–59
<code>%s</code>	Секунды с лидирующим нулем	00–59
<code>%d</code>	День месяца, две цифры с лидирующим нулем	01
<code>%D</code>	Текстовое представление дня (трехбуквенная аббревиатура)	Mon
<code>%j</code>	День месяца (1 или 2 цифры)	1
<code>%l</code>	Название дня недели (полное)	Monday
<code>%w</code>	Номер дня недели	0 (Вс) до 6 (Сб)
<code>%z</code>	Номер дня года	0–365
<code>%W</code>	Номер недели в году	42
<code>%F</code>	Полное название месяца	December
<code>%M</code>	Короткое название месяца (трехбуквенное)	Dec
<code>%m</code>	Номер месяца с лидирующим нулем	01
<code>%n</code>	Номер месяца без лидирующего нуля	1
<code>%t</code>	Количество дней в текущем (или заданном, если указан timestamp) месяце	от 28 до 31

Таблица 12.3 (окончание)

Модификатор	Описание	Диапазоны/ примеры возвращаемых значений
L	Високосный ли год	1, если год високосный, 0 — в противном случае
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 говорю, что PHP не создан 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 $\$x$



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;
- ❑ `substr()` — возвращает подстроку;
- ❑ `strcmp()` — сравнивает две строки;
- ❑ `strcasecmp()` — сравнивает две строки без учета регистра символов;
- ❑ `str_repeat()` — используется для повтора строки;
- ❑ `str_replace()` — производит замену 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 она возвращает 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 `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 используется 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.

Данная функция учитывает регистр символов. Если вам нужно сравнивать строки без учета регистра, тогда используйте функцию `strcasecmp()`. Ее параметры и возвращаемые значения такие же, как у `strcmp()`.

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] )
```

Данная функция производит замену подстроки `search` строкой `replace` в строке `subject`. Последний параметр задает количество замен, которые должна произвести функция. Обратите внимание, что последний параметр доступен начиная с PHP 5, поэтому если у вас до сих пор PHP 4, то самое время обновиться!

Заметьте, что данная функция не изменяет исходную строку `subject`, а работает с ее копией, а потом возвращает результат.

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 нужно 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.

13.3. Преобразование строки

К этой группе относятся следующие функции:

- ❑ `strip_tags()` — удаляет HTML-теги из строки;
- ❑ `strtolower()` — переводит символы строки в нижний регистр;
- ❑ `strtoupper()` — переводит символы строки в верхний регистр;
- ❑ `trim()` — удаляет пробельные символы до и после строки;
- ❑ `ltrim()` — удаляет пробельные символы в начале строки;

- ❑ `chop()` — удаляет пробельные символы в конце строки;
- ❑ `urlencode()` — кодирует строку для ее передачи в URL;
- ❑ `urldecode()` — раскодирует строку, закодированную функцией `urlencode()`;
- ❑ `wordwrap()` — разбивает текст согласно указанным параметрам.

Конечно, это далеко не все строковые функции, которые есть в арсенале PHP, но это 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 данная 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`, вы 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);
```

Функции `strtolower()` и `strtoupper()` подробно рассматривать не будем — они слишком просты. Все что нужно — это передать функции строку, регистр которой нужно изменить. Как и остальные строковые функции PHP, данные функции не изменяют исходную строку. Если вам нужно изменить исходную строку, можно вызвать функцию так:

```
$str = strtolower($str);
```

Довольно часто приходится удалять пробелы в данных, оставленных пользователем в форме. Функции `trim()`, `ltrim()` и `chop()` удаляют пробелы и пробельные символы (вроде табуляции) из строки. 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          Иванов"
```

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 заменяются 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>");
```

Заметьте, что функция `wordwrap()` распознает слова и не разбивает их. Если же вы хотите вставлять указанный разделитель через указанное 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 используется 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. Второй параметр — 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`.

13.5. Функции для работы с отдельными символами 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 предыдущем примере 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
```

13.6. Функция `md5()`

Функция `md5()` кодирует заданную строку с помощью алгоритма MD5. Особенность данного 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";
```

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. Например, 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 нашей строке. В этом случае функция прочитает только указанное нами количество полей.



Работаем с файлами и каталогами

14.1. Права доступа в UNIX

Прежде чем приступать к рассмотрению функций для работы с файлами и каталогами, мы должны рассмотреть права доступа к файлам и каталогам. Если вы работаете в данный момент в Windows, вы можете пропустить этот раздел, но вы обязательно вернетесь к нему, когда будете 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 РНР.

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 (то есть для себя), для группы владельца (то есть 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 видно, 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 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 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. 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 восьмеричной системы в двоичную

Восьмеричная система	Двоичная система
0	000
1	001

Таблица 14.1 (окончание)

Восьмеричная система	Двоичная система
2	010
3	011
4	100
5	101
6	110
7	111

Аналогично произведем разбор прав для членов группы владельца. Получится 100, то есть 4. С третьим набором (---) все вообще просто: это 000, то есть 0.

Записываем полученные числа в восьмеричной системе в порядке владелец — группа — остальные. Получится число 640, это и есть права доступа. Для того чтобы установить эти права, 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, то есть сделать какого-то пользователя владельцем файла, то вам нужно использовать команду `chown`:

```
chown пользователь файл
```

Учтите, что, возможно, после изменения владельца файла вы сами не сможете получить к нему доступ, ведь владелец уже не вы.

А теперь поговорим о нюансах доступа к файлам из PHP-сценариев. Предположим, что у нас есть сценарий гостевой книги `gb.php` и файл `gb.txt`, в котором хранятся записи гостевой книги. Вы загрузили оба файла на сервер по протоколу FTP. Следовательно, вы — владелец этих двух файлов. По умолчанию (в большинстве случаев) будут использованы права доступа 644.

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 Web-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 использовать таблицы MySQL для хранения данных, 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()`;
- ☐ построчное чтение файла с помощью функции `file()`;
- ☐ чтение всего файла сразу функцией `file_get_contents()`.

14.2.1. Использование функций `fopen()` и `fread()`

Функции `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 права доступа, как вы могли подумать. Это 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, то есть при записи в файл новые данные будут записаны поверх уже имеющихся. Например, если в файле была строка "Мама мыла раму", то после записи в файл строки "Папа" получится строка "Папа мыла раму". 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. 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.

Вы должны не просто вызвать функцию `fopen()`, а получить дескриптор файла, который вы потом будете использовать в других функциях (чтобы та же функция `fread()` знала, из какого файла нужно произвести чтение, если вы открыли несколько файлов). Вот примеры использования `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 прочитать из него. Для этого используется функция `fread()`, которой нужно передать два параметра —

дескриптор файла и количество байтов, которое должно быть прочитано из файла:

```
// 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($handle, 1)` можно использовать функцию `fgetc($handle)` для чтения одного символа.

По окончании обработки файл нужно закрыть с помощью функции `fclose()`:

```
fclose($handle);
```

Вообще PHP при завершении сценария автоматически закрывает все открытые файлы, но самостоятельное закрытие файла считается хорошим тоном.

14.2.2. Использование функции *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 используются 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 мы использовали четыре функции для организации построчного чтения. 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";  
}
```

Как видите, использовать `file()` намного проще. 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 allow-  
able_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. Намного удобнее объединить массив в строку и вывести 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');
```

Данная функция появилась в PHP 5, поэтому если у вас древняя версия PHP, то вам придется использовать функции `join()` и `file()`.

14.3. Запись файла

Настало время поговорить о записи информации в файл. Для этого можно использовать следующие функции:

- ❑ `file_put_contents()` — записывает строку в файл;
- ❑ `fwrite()` (или `fputs()`) — используется 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] )
```

Намного проще использовать `file_put_contents()`: вам не нужно получать дескриптор файла, проверять этот дескриптор, вызывать `fwrite()`. Вам нужно просто указать имя файла, данные и флаги. Самый полезный флаг `FILE_APPEND`: если вы будете его использовать, данные будут записаны 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 используете `fwrite()` и хотите убедиться, что 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 хочется. Во-первых, тогда нам нужно контролировать имя файла — чтобы ненароком не перезаписать существующий файл. Во-вторых, нам нужно удалить временный файл после обработки. Намного проще использовать функцию `tmpfile()`, создающую временный файл в каталоге `/tmp`.

Вот пример использования:

```
$temp = tmpfile();  
fwrite($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.

После сохранения этой таблицы в формате CSV мы получим следующий файл:

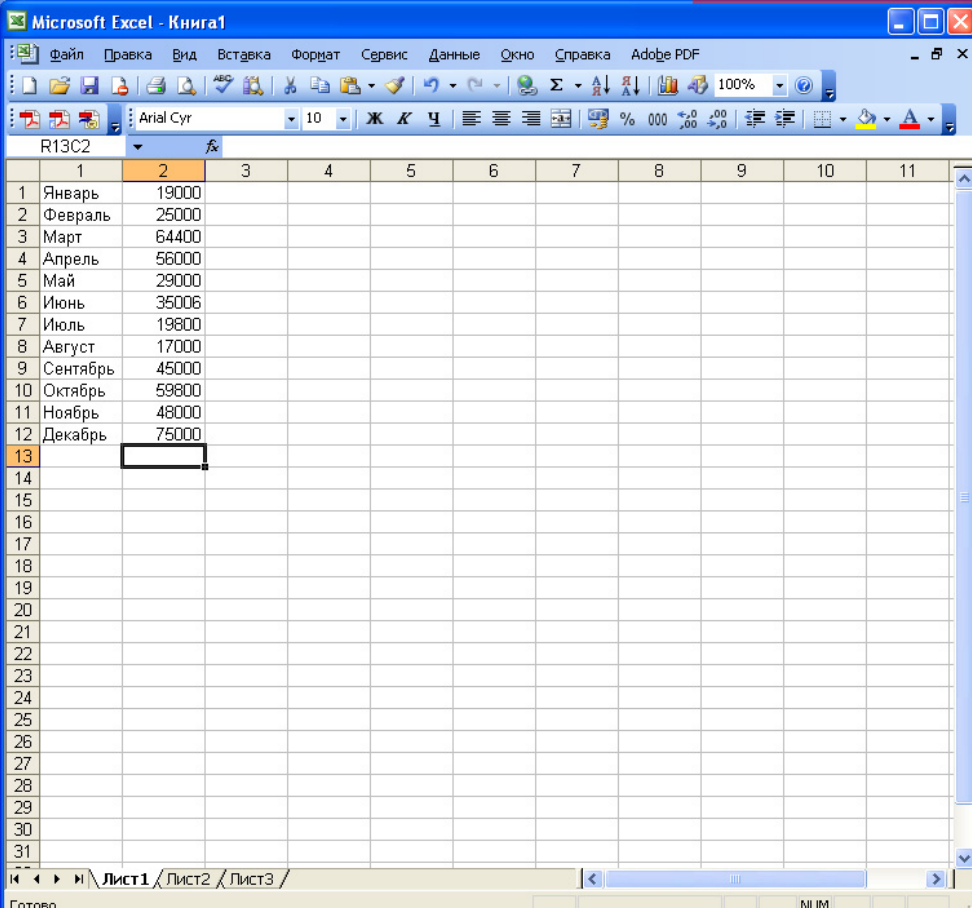
```
Январь;19000  
Февраль;25000  
Март;64400  
Апрель;56000  
Май;29000  
Июнь;35006  
Июль;19800  
Август;17000
```

Сентябрь;45000

Октябрь;59800

Ноябрь;48000

Декабрь;75000



The screenshot shows a Microsoft Excel window titled "Microsoft Excel - Книга1". The interface includes a menu bar (Файл, Правка, Вид, Вставка, Формат, Сервис, Данные, Окно, Справка, Adobe PDF), a toolbar, and a formula bar showing "R13C2". The spreadsheet has 11 columns and 31 rows. The data is as follows:

	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											

The status bar at the bottom shows "Готово" and "NUM".

Рис. 14.1. Примерная электронная таблица

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 запя-

тую в качестве разделителя, поэтому если мы работаем с CSV-файлами, полученными из Excel, нужно задать разделитель:

```
fgetcsvg($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 (($data = fgetcsvg($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 используется функция `fputcsvg()`:

```
int fputcsvg ( 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 довольно просто использовать, 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()` — позволяет изменить группу владельцев файла;
- ❑ `chmod()` — изменяет права доступа к файлу или каталогу;
- ❑ `chown()` — позволяет изменить 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 директорией;
- ❑ `is_executable()` — определяет, является ли файл исполняемым;
- ❑ `is_file()` — позволяет определить, является ли файл обычным файлом;
- ❑ `is_link()` — позволяет определить, является ли файл символической ссылкой;
- ❑ `is_readable()` — доступен ли файл для чтения;

- ❑ `is_uploaded_file()` — позволяет определить, был ли файл загружен с помощью HTTP POST;
- ❑ `is_writable()` — 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 используется 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`, `false` — в противном случае.

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 используется 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 используется 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 RНР мог скопировать файл в указанный вами каталог, пользователь, от имени которого запущен RНР, должен иметь право записи в этот каталог;
- ❑ чтобы прочитать файл (например, при копировании), у RНР 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;

- ❑ `filetime()` — 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 (открытие) и модификации (изменение), 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 только 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 непредсказуемый. Чтобы его предсказать и точно знать, что одновременно отправленные данные записаны в нужном порядке (а не вперемешку) и что сценарий точно сможет прочитать файл, нужно использовать механизм блокировок. Данный механизм доступен только в UNIX, то есть он будет работать на сервере хостинг-провайдера, но не будет работать на вашем домашнем компьютере под управлением Windows.

Для установки блокировки используется 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 установить записывание на файл, но и заблокировать выполнение сценария, пока файл не станет доступным.

Рассмотрим значения второго параметра:

- ❑ `LOCK_EX` — исключительная блокировка, ее нужно использовать 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 используется 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:

```
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 используется 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 использовались 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>



Вывод графических изображений средствами PHP

15.1. Библиотека GD

Библиотека GD предназначена для работы с изображениями и графическими файлами форматов JPEG, GIF, PNG, WBMP и XPM. Используя данную библиотеку, можно программно создавать изображения, сохранять их в разных форматах или сразу выводить в браузер.

Если честно, 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 используется функция `exif_imagetype()`, которой нужно передать имя графического файла:

```
int exif_imagetype (string filename)
```

Функция возвращает числовое значение, соответствующее формату файла (табл. 15.1).

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

Если формат файла определить не удалось, то функция возвращает `false`.

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.

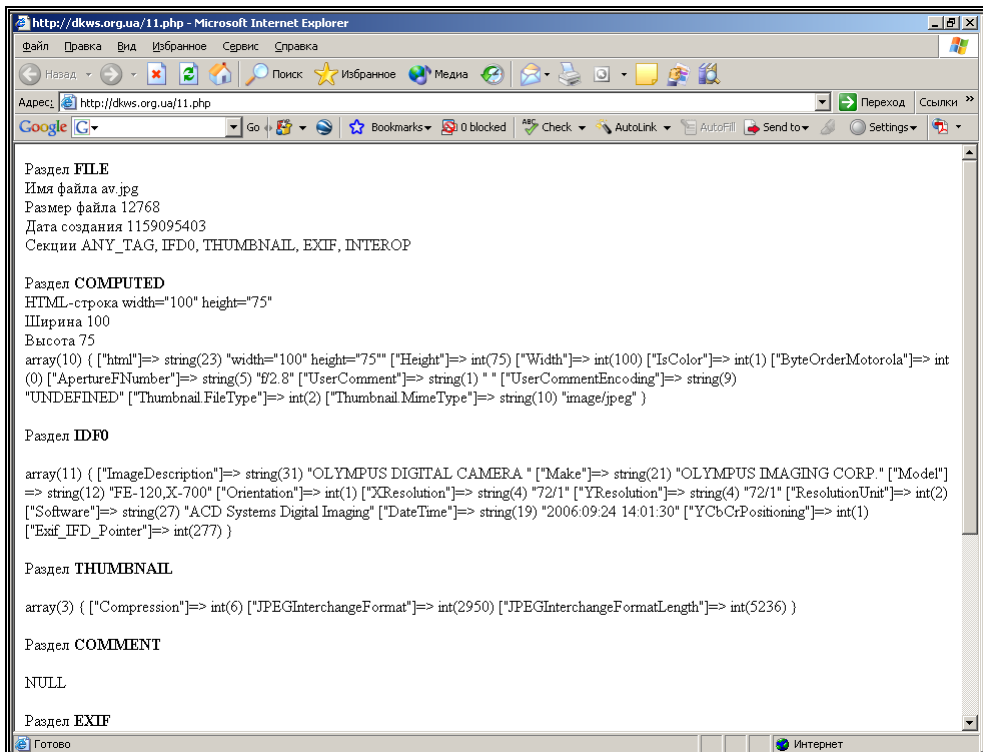


Рис. 15.1. Информация о JPEG-файле

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
IFD0	Все данные с тегами IFD0

Таблица 15.2 (окончание)

Раздел	Параметры
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 данной функции (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>IDF0</b><p>";
var_dump($info['IDF0']);

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']);

?>
```

Результат выполнения сценария представлен на рис. 15.1.

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.

Также существует функция `png2wbmp()` преобразования формата PNG в формат Windows BMP. 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);

?>
```

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, тогда 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
<code>resource imagecreatefromgd2 (string filename)</code>	Загружает изображение формата GD2 из файла
<code>resource imagecreatefromgif (string filename)</code>	Загружает изображение из файла формата GIF
<code>resource imagecreatefromjpeg (string filename)</code>	То же, JPEG
<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, то есть 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

Таблица 15.3 (окончание)

Функция	Описание
<pre>int imagejpeg (resource image [, string filename [, int quality]])</pre>	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
<pre>int imagepng (resource image [, string filename])</pre>	Сохраняет изображение на диск в формате PNG
<pre>int imagewbmp (resource image [, string file name [, int foreground]])</pre>	Сохраняет изображение на диск в формате WBMP. Параметр <code>foreground</code> задает цвет переднего плана (по умолчанию черный, то есть <code>foreground = 0</code>). Для вывода такого изображения в браузер нужно передать заголовок: <code>header('Content-type: image/vnd.wap.wbmp ');</code>
<pre>int imagexbm (resource image, string file name [, int foreground])</pre>	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

Для вывода строки используется функция `imagestring()`:

```
int imagestring (resource image,
int font,
int x,
int y,
string s,
int color)
```

Данная 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.

Задать цвет можно с помощью функции `imagecolorallocate()`:

```
int imagecolorallocate (resource image,  
                       int red,  
                       int green,  
                       int blue)
```

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 TTF-шрифтом (в каталоге `Windows\Fonts` вы найдете TTF-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 (листинг 15.3) вывода русскоязычного текста.

Листинг 15.3. Вывод русскоязычного текста

```
<?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
imagefttext($image, 20, 0, 10, 10, $color, "arial.ttf", "Привет");

// выводим картинку в формате PNG в браузер
header('Content-type: image/png');

ImagePng($image);
```

?>

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.

15.2. Изменение размера изображения

Вспомним, как организована любая графическая галерея. Сначала выводятся миниатюрные изображения. Как правило, размер миниатюры равен 100×80 пикселей. При нажатии на миниатюру выводится увеличенное 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);  
}
```

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.

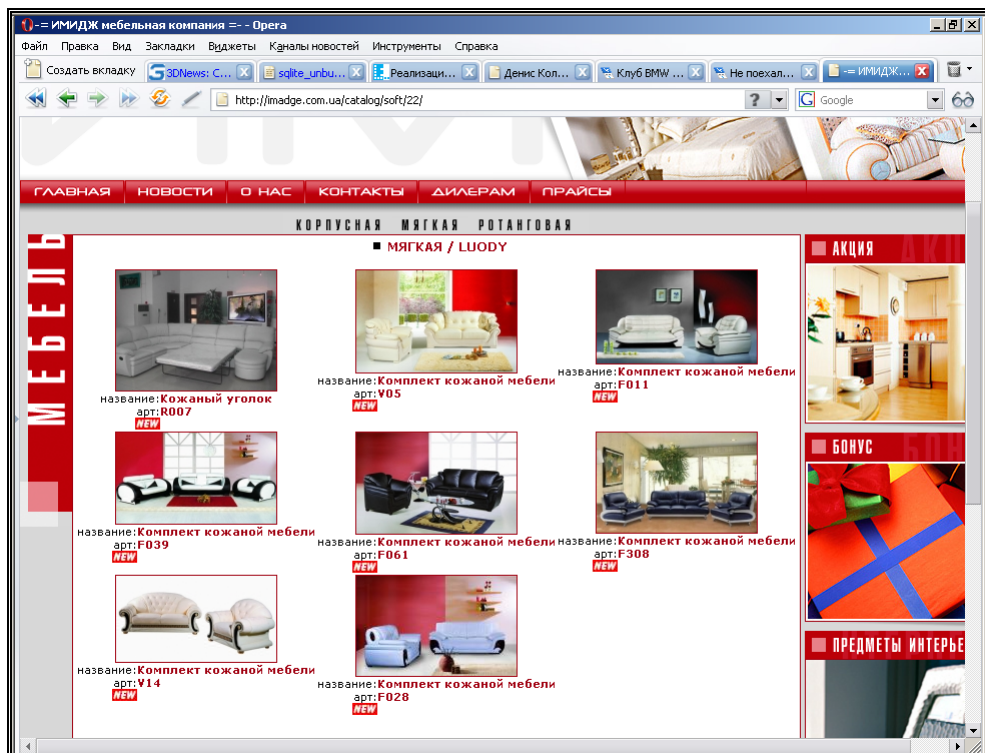


Рис. 15.2. Вывод миниатюр

Наша функция работает только с JPEG-файлами, но при желании вы без проблем можете переделать ее для работы с GIF- и PNG-файлами.

Если разобраться с кодом нашей функции, то видно, что фактически изменяет размер функция `imagecopyresampled()`, копирующая прямоугольные области с одного изображения на другое. Наша функция только подготавливает все для ее работы.

Вот пример вызова нашей функции:

```
imageresize('big_file.jpg', 'big_file_thumb.jpg', 160, 100, 75);
```

На рис. 15.2 изображен результат работы нашей функции на примере реально работающего проекта.

15.3. Создание водяных знаков

Водяные знаки защищают банкноты от подделки. "Водяные знаки" на изображениях позволяют защитить изображения от несанкционированного копирования. Однажды с подобной просьбой ко мне обратился один 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. А если нанести надпись внизу или сверху (слева или справа — вертикально), от нее будет 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 картинки: 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. Класс `watermark` (файл `api.watermark.php`), а также все листинги из этой главы вы найдете на прилагаемом диске в каталоге `Glava_15`.

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);

?>
```

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 должна быть 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 изображен на рис. 15.3.

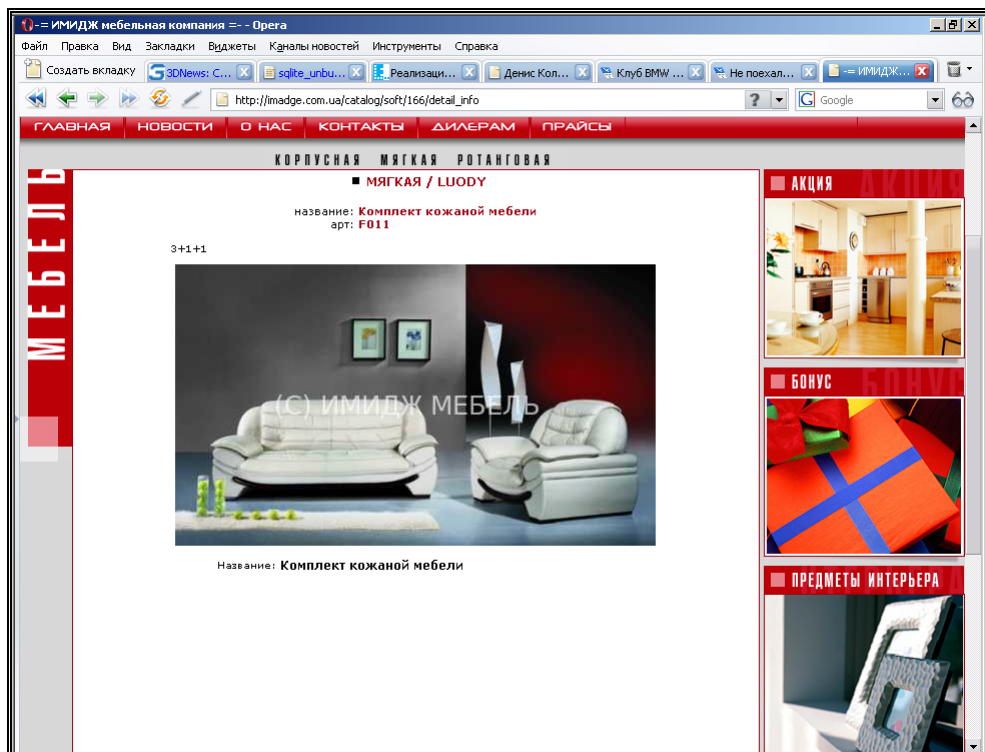


Рис. 15.3. Изображение с водяным знаком



Работа с сетевыми сокетами в PHP. Сетевые функции

16.1. Еще раз о том, что такое сокет

Все мы знаем, как работать с файлами в PHP. 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, то есть служит для передачи файлов по сети. Примечательно, что с сетевым соединением в PHP 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 TCP-серверу, например к Web-, FTP-серверу, почтовому и т. д. Конечно, функции PHP и так могут работать с Web-, FTP-серверами, а для работы с почтовыми серверами уже есть готовые классы. Так для чего нужны сокеты? Для реализации нестандартного обмена. Предположим, что вам необходимо обмениваться 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, то есть 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 — это 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 приведены 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, то есть и для чтения, и для записи.

16.3. Пример использования сокетов

Рассмотрим небольшой пример, демонстрирующий общую концепцию работы с сокетами. Мы подключимся к Web-серверу, передадим ему запрос и прочитаем ответ сервера. Как уже было отмечено, socket 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 socket с протоколом HTTP

```
<?
// открываем socket, 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 socket
fclose($sock);
?>
```

Для того чтобы обмениваться информацией с сервером, вам нужно знать протокол обмена. В Интернете есть 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

Документы RFC 2068, 2616, 1870, 1939 можно найти в архиве по адресу <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**. Вызывать сценарий нужно так:

`http://<сервер>/soc.php?ip=<IP-адрес>`

Сценарий представлен в листинге 16.2.

Листинг 16.2. Сценарий soc.php

```
<?php

if ($ip!="")
{
    // подключаемся к whois.ripe.net, порт 43
    $sock = fsockopen ("whois.ripe.net", 43, $errno, $errstr);
    if (!$sock)
    {
        // в случае ошибки выводим сообщение об ошибке
        echo("$errno($errstr)");
        return;
    }
    else
```

Листинг 16.3 иллюстрирует примерный вывод сценария, содержит всю информацию о переданном IP-адресе.

```
% This is the RIPE Whois query server #2.
% The objects are in RPSL format.
%
% Note:          the default output of the RIPE Whois server
% is changed. Your tools may need to be adjusted. See
% http:          //www.ripe.net/db/news/abuse-proposal-20050331.html
% for more details.
%
% Rights restricted by copyright.
% See http:      //www.ripe.net/db/copyright.html
%
% Note:          This output has been filtered.
% To receive output for a database update, use the "-B" flag
% Information related to '192.168.1.0 - 192.168.1.255'
```

```
inetnum:      194.67.57.0 — 194.67.57.255
netname:      SERVERRU-NET2
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 предыдущем листинге (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 сокете 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 младше, то данной функции у вас не будет.

16.5. DNS-функции

Для разрешения IP-адреса в доменное имя и наоборот используются 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);

?>
```



Собственные функции

17.1. Зачем нужны собственные функции

До этого 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 среди стандартных функций PHP нужную вам функцию, тогда вы можете 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 пример такой функции:

```
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 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 можете 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 этой 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, то есть `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 разберемся, что такое локальные и глобальные переменные. Глобальной называется переменная, объявленная вне функции. Локальной называется переменная, объявленная в теле функции.

Предположим у вас есть глобальная переменная `$i`, используемая 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, вам нужно 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!

Сейчас разберемся, 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 не прямой рекурсии.

Рекурсия — довольно мощный инструмент, но в то же время он довольно опасен в неопытных руках. Вообще любой рекурсивный алгоритм можно преобразовать в аналогичный нерекурсивный, то есть 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.



РАЗДЕЛ 5

Серверы баз данных MySQL 6 и SQLite



Основы SQL

18.1. Немного истории

В далеком 1970-м году 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), который создавал компилятор запросов. Их главной целью было создание языка запросов к базе данных, с которым мог бы разобратся любой пользователь, не имеющий никаких навыков программирования.

В 1986 г. Американским национальным институтом стандартизации (ANSI, American National Standards Institute) был принят первый стандарт языка SQL. Принятие стандарта должно было способствовать дальнейшему развитию языка запросов. В 1987 г. стандарт был уточнен (появился так называемый SQL level 1). Еще одно изменение в стандарте произошло в 1989 г. (SQL level 2).

Развитие и совершенствование систем управления базами данных послужило причиной принятия в 1992 г. следующего стандарта (он назывался ANSI SQL-92 или просто SQL-2). Последующие стандарты были приняты в 1999 г. (SQL-99) и 2003 г. (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 будет 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 эти приложения.

В случае с SQL все гораздо проще. Вам не нужно 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 придется — 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 управ-

лять ими: добавлять новые данные, изменять и удалять уже имеющиеся, создавать базы данных. Выходит, что кроме знания SQL вам не понадобятся никакие другие дополнительные сведения для работы с СУБД.

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 представления об SQL. Очевидно, этот 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 будет 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, то есть 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 много другого. Вот и получается, что в одной базе данных будет несколько наборов таблиц, связанных только между собой, 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, то есть 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 СНГ очень много, поэтому в нашей таблице может быть два Иванова, и 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 даже для разработки 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 если вам интересно, 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
TINYINT	Целый тип, может принимать значения из диапазона –128...+127
SMALLINT	Целый тип, может принимать значения из диапазона –32 768...+32 767
MEDIUMINT	Целый тип, может принимать значения из диапазона –8 388 608...+8 388 607
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), то есть количество знаковых мест, 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)

Таблица 18.2 (окончание)

Тип	Описание
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 байтов
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, то есть 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,
commenttext,
primarykey (id)
);
```

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, так как 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 пустое значение.

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;
```

Данный 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, то есть 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;
```

`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;
```

Данный 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 следующим 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;
```

18.11. Группировка записей.

Сложные запросы

До этого 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)  
);
```

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, 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.

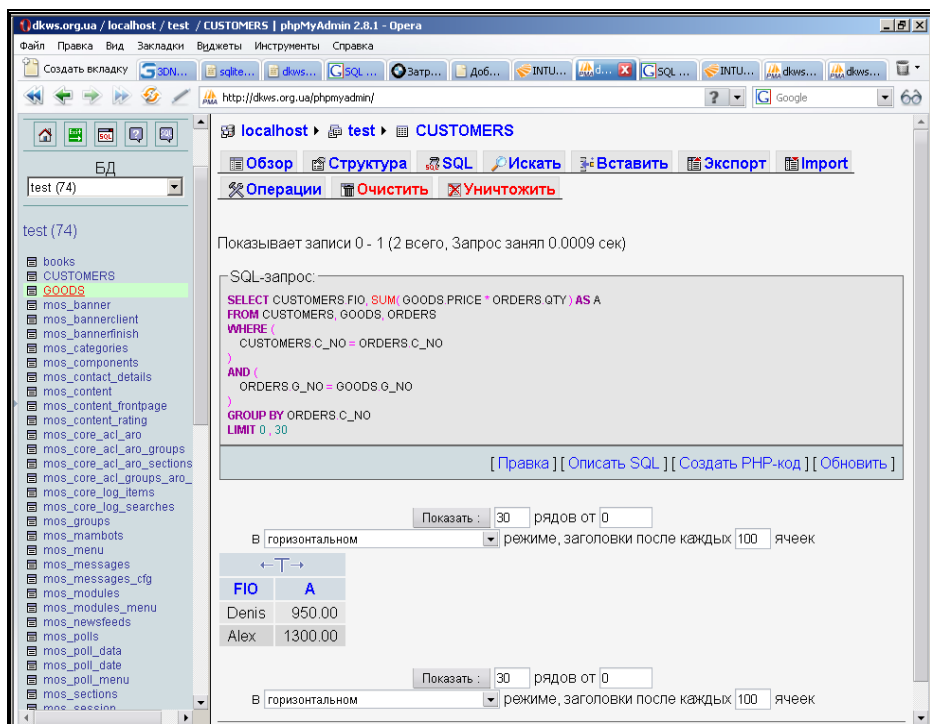


Рис. 18.1. Группировка записей: вывод общей суммы всех заказов с группировкой по клиенту

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.



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 (об этом 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` (по-

сколько первое значение обычно больше второго, таким способом можно увеличить тайм-аут).

Вообще все параметры здесь не являются обязательными. Дело в том, что в `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 нужно 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 Web-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 Web-сервера.
3. Закрытие соединения удаленным сервером MySQL.

Функция `mysql_pconnect()` работает так: проверяет, есть ли идентификатор соединения. Если он есть, то она 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 нужно тратить время на создание соединения. Да, за один вызов сценария вы этого не почувствуете, а вот когда у вас сложный сценарий (например, форум), который использует постоянные соединения, тогда результат будет виден без всяких замеров производительности (ведь соединение потребует 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 ней (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 используем 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()` существует 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

Таблица 19.1 (окончание)

Функция	Описание
<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: <code>MYSQL_ASSOC</code> — ассоциативный массив (по умолчанию); <code>MYSQL_NUM</code> — численный массив; <code>MYSQL_BOTH</code> — 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 (листинг 19.2) вывода всех записей этой таблицы с помощью функции `mysql_fetch_array()`.

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 (листинг 19.3) вывода этой же таблицы, но с помощью функции `mysql_fetch_object()`.

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 быстрдействию 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 параграфе. 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.

Если вам "лень" выполнять `mysql_select_db()` для переключения к другой базе данных, то можно 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 ( строка 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\")";
```

Данный оператор формирует запрос, добавляющий в таблицу `tab` значение переменной `$content`. Вроде бы все нормально и этот запрос должен выполниться без особых проблем. Когда в переменной `$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());
```



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 MySQL).

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 Web-интерфейса. Однако если у вас имеется собственный сервер, то вы можете установить на него Oracle и настроить PHP для работы с этой СУБД. Если вам интересно, как это сделать, то 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.

Теперь поговорим о mSQL. Автор этой СУБД описывает ее как (буквально) "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 первая строчка результата будет: "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 могло так получиться, что пятая версия PHP могла остаться вообще без поддержки MySQL. Нужно было чем-то заменить MySQL. Такой заменой стал SQLite. Почему именно SQLite? Он также полностью бесплатный 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. Сейчас объясню разницу. К MySQL-серверу, если это разрешил администратор, можно подключаться с других узлов сети (можно указать, с каких именно). Таким образом, физически наш сценарий может быть размещен на одном сервере (Web-сервер), а база данных — на другом. В случае с SQLite сервер баз данных обязательно должен быть установлен на Web-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 будут 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, 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

SQLite работает немного иначе. Поскольку это 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 — это имя файла базы данных. Если указанный файл не существует, функция попытается его создать. Другими словами, данную функцию можно использовать не только для открытия базы данных, но и для ее создания. Второй параметр — это режим доступа к базе данных. По умолчанию установлен режим 0666 (режим здесь — это права доступа к файлу в UNIX). 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), то есть позволяет перейти к произвольной записи результата. Это очень удобно, когда нужно обработать результат не 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);
}
?>
```

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
<pre>array sqlite_array_query (resource db, string query [, int res_type [, bool de- code_binary]])</pre>	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>
<pre>void sqlite_busy_timeout (resource dbhandle, int milliseconds)</pre>	Устанавливает максимальное время ожидания готовности базы данных (в миллисекундах). По умолчанию это время равно 60 с
<pre>int sqlite_changes (resource dbhandle)</pre>	Возвращает количество записей, измененных в результате последнего запроса
<pre>array sqlite_current (resource result [, int re- sult_type [, bool de- code_binary]])</pre>	Возвращает текущую запись
<pre>string sqlite_error_string (int err_code)</pre>	Возвращает строку с описанием ошибки. Функции нужно передать код ошибки

Таблица 20.1 (продолжение)

Функция	Описание
string sqlite_escape_string (string item)	Экранирует спецсимволы в строке SQL-запроса. Обратите внимание: функцию addslashes() нельзя использовать для экранирования запросов к SQLite
bool sqlite_exec (string query, resource dbhandle)	Выполняет запрос, который не возвращает записей, например, INSERT-запрос. Функция возвращает TRUE, если запрос выполнен. Применение этой функции совсем необязательно, ведь есть sqlite_query()
object sqlite_fetch_object (resource result [, string class_name [, array ctor_params [, bool de- code_binary]])	Возвращает текущую запись результата в виде объекта. Аналогична по своему действию функции mysql_fetch_object()
array sqlite_fetch_all (resource result [, int result_type [, bool decode_binary]])	Выбирает все записи и возвращает их в виде массива
sqlite_fetch_string()	Псевдоним для функции sqlite_fetch_single()
string sqlite_field_name (resource result, int field_index)	Возвращает имя поля с указанным номером поля. Нумерация начинается с нуля
bool sqlite_has_more (resource result)	Возвращает TRUE, если в результате еще есть доступные записи
bool sqlite_has_prev (resource result)	Возвращает TRUE, если доступна предыдущая запись результата
int sqlite_key (resource result)	Возвращает индекс текущей записи результата

Таблица 20.1 (окончание)

Функция	Описание
<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), то есть когда нет необходимости доступа к произвольной записи результата



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 `koï8-r`, а на сайте — `cp1251`. Получается, что содержимое (в том числе и навигация) сайта выводится в кодировке `cp1251`, а информация из базы данных — в `koï8-r`. В результате одновременно прочитать вывод из базы данных и страницу сайта не удастся, кодировки-то разные (рис. 21.1).

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;');
```



Рис. 21.1. Кодировки сайта и базы данных разные

21.2. Сортировка: вывод новинок. Вывод случайных записей

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 будут отображаться в обратном порядке, то есть 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 bid DESC  
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 данная функция присутствует практически во всех серьезных PHP-проектах, поэтому она обязательно должна быть и у вас.

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 запи-

сей, 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>
```

The screenshot shows the forum page for dkws.org.ua. The forum is titled "Linux" and has moderators "КУБыч, Krow, Anderson". It shows a list of forum topics with columns: Темы, Ответов, Автор, Просмотров, and Последнее сообщение. The first topic is "Важная: Пользователей Linux приглашают зарегистрироваться". The pagination links at the bottom right of the topic list are circled in red: "На страницу 1, 2, 3, 4, 5 След."

Темы	Ответов	Автор	Просмотров	Последнее сообщение
Важная: Пользователей Linux приглашают зарегистрироваться	0	den	490	04 Сен 2006 01:27 pm den
Школы Пермской области перейдут на Linux	1	den	122	10 Фев 2007 05:54 pm Krow
Ubuntu и Linspire создали технологический союз	0	den	95	09 Фев 2007 06:06 am den
Автомобильный концерн Peugeot Citroen переходит на Linux	1	den	141	08 Фев 2007 01:13 pm den
Novell может запретить продавать Linux	1	den	157	08 Фев 2007 01:12 pm den
Эмуляция без ограничений	0	den	107	08 Фев 2007 01:11 pm den
Linux расширяет возможности недорогих однопроцессорных мобил	0	den	92	08 Фев 2007 01:09 pm den
Linux для криминалистов	0	den	85	08 Фев 2007 01:09 pm den
В Бельгии появилась первая школа по Linux и Open Source	0	den	99	03 Фев 2007 07:30 am den
Крупнейший продавец бензина Швеции выбрал MySQL и Linux	0	den	110	03 Фев 2007 07:30 am den

Рис. 21.2. Постраничный вывод форума PHPBB

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 будет выводить объявления из таблицы `ob`. Структура таблицы очень проста: поле `dt` — дата публикации объявления, а поле `t` — текст самого объявления. Сценарий будет 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.

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, то есть 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>";
}

?>
```

Готовый сценарий представлен на рис. 21.3.

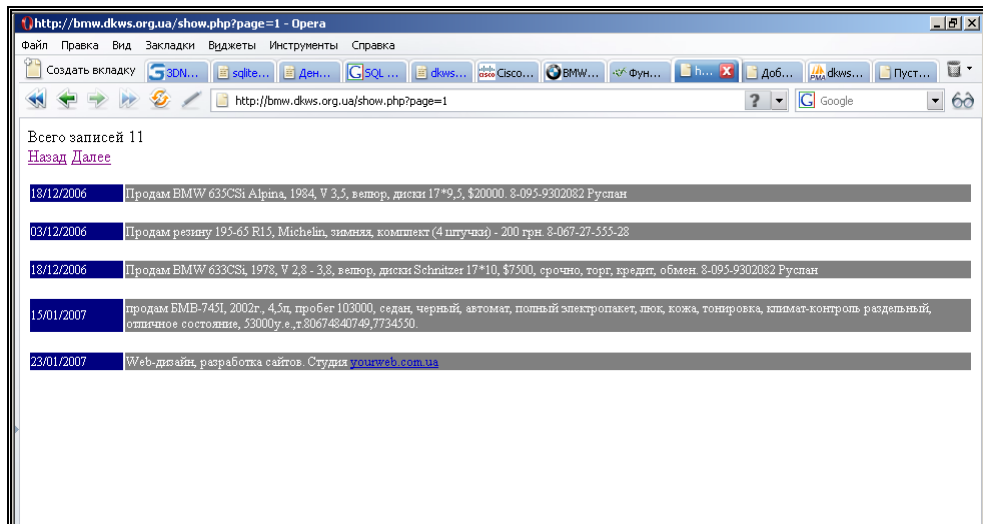


Рис. 21.3. Постраничный вывод таблицы



РАЗДЕЛ 6

Инструменты для создания сложных проектов



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.

Начнем с сортировки файлов по отдельным каталогам. Понимаю, что когда у вас 30 или даже больше 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 изменились, поэтому нужно будет изменить несколько десятков тегов `img`, чтобы дизайн сохранил свою целостность.

После этого вынесем все HTML-файлы в каталог `html`. Но это еще не все. HTML-файлы ведь содержат какую-то информацию. Поэтому целесообразно отсортировать их по содержанию. 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, отсортировать 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 — отсортировать РНР-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 `index.php` и `config.php` (о нем поговорим позже). Все остальные файлы следует отсортировать 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 РНР-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.

У нас получилась структура каталогов, подобная изображенной на рис. 22.1.

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 Vista, тогда файл этой статьи можно назвать, например, так: `windows_vista_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`

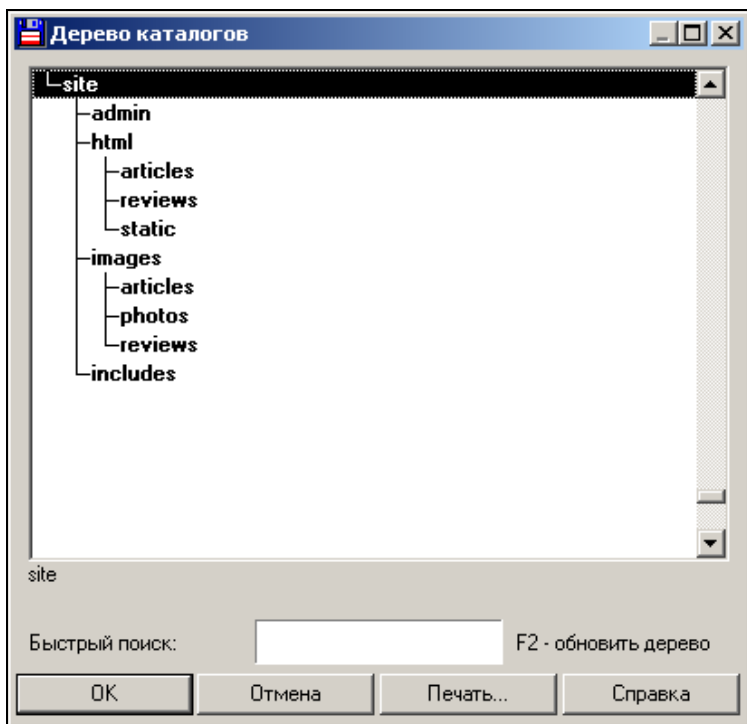


Рис. 22.1. Примерная структура каталогов

В данной 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 символическое 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).

Не знаю, как вам, но мне проще ориентироваться, когда название файла отображает его содержимое, поэтому для своего сайта я выбрал именно этот способ. Но он не всегда оптимальный. В данном случае он лучше всего подходит для меня. Но если вы пишете серьезный проект, например систему управления контентом сайта (CMS), тогда первый вариант отпадает. Сейчас поясню. Заходит на сайт пользователь и хочет опубликовать статью с назва-

нием "Установка Windows Vista, или Проверяем, как будет тормозить система на 512 Мбайт оперативки". Вот теперь подумайте, какой алгоритм придется написать, чтобы это название было трансформировано в "windows_vista_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 целесообразнее будет такой формат имени файла для статьи:

article_ДД_ММ_ГГГГ_Номер

Согласитесь, такое имя файла сформировать намного проще. С другой стороны, можно 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 доставать 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 данные 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 этих 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';

?>
```

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 включаются 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, то есть 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 этих инструкций есть определенная особенность: подключенный файл становится неотъемлемой частью сценария. Сейчас объясню. 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");

?>
```

Допустим, что произошла ошибка подключения к серверу баз данных. Будет выполнена инструкция `die()`, которая прервет выполнение не дополнительного, а основного сценария, поскольку дополнительный инструкцией `include` просто встраивается в основной сценарий. В результате мы увидим сообщение об ошибке, но не увидим нижнюю часть дизайна. Это нехорошо. Выход из данной ситуации можно найти. Есть даже два выхода: во-первых, можно изменить сценарий `connect.php` приблизительно так:

```
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 данные файлы с помощью инструкции `file()`. Но в нашем случае инструкцию `file()` использовать нельзя. А вот при подключении сценария, который не выполняет сервисных или подготовительных действий для основного сценария (ведь сценарий `connect.php` подключает основной сценарий к базе данных), 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'));
```

Обратите внимание: нужно указывать URL файла, а не путь к файлу, иначе вместо результата работы будет выведен исходный код файла.

Еще раз повторяю, 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 `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, был ли он подключен или нет. Ведь ваш проект может стать настолько огромным, что 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, исключили лишний код из PHP-файлов в отдельные файлы — так намного удобнее. Теперь настал черед вынести HTML-код из наших сценариев.

Даже если вы начинающий PHP-программист, то, скорее всего, вы 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, в нашем PHP-сценарии есть HTML-код, который нельзя поместить в обычный HTML-файл, поскольку его нужно модифицировать и передавать в браузер.

Если вы сами планируете поддерживать ваш сценарий, то вроде бы ничего страшного. 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 наш сценарий — это не эти простые пять строк, а сотни строк PHP-кода, в котором 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. В этом случае наш HTML-код будет вынесен в файл `welcome.tpl`, приведенный в листинге 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>
```

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.

В этой главе мы напишем очень простой шаблонизатор, то есть 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;
?>
```

Вот теперь полный порядок — из РНР-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).

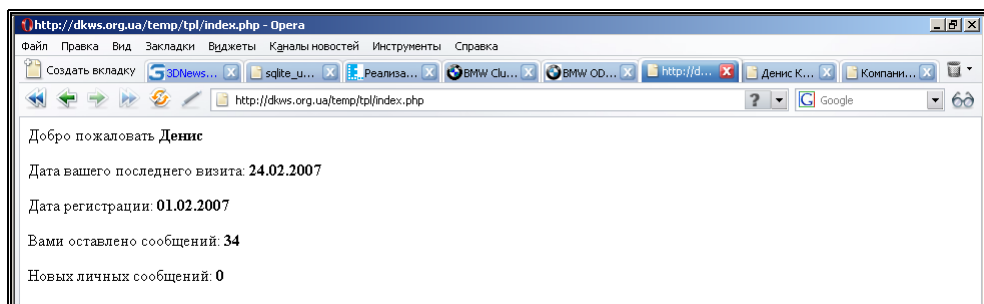


Рис. 22.2. Шаблонизатор в действии

Осталось рассмотреть только сам файл `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;

?>
```

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.



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, как исключить HTML-код 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 код, и сценарий потом не будет работать.

У нашего простенького шаблонизатора есть существенный недостаток: если шаблоны будут объемными, то он станет работать довольно медленно. У Smarty данного 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**. Последние версии (стабильная и самая новая, но бета) Smarty доступны в каталоге Glava_23 прилагаемого диска. В этом же каталоге вы найдете официальное руководство по Smarty на русском языке в формате PDF.

Для установки Smarty потребуется Web-сервер с установленным PHP 4.0.6 или более свежей версией. Распакуйте содержимое каталога `libs` на ваш 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 Web-сервера, например в каталог `/usr/local/lib/php/`. Кроме этого, 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 Web-сервера. У меня как раз второй случай, поэтому я распаковал каталог `libs` в корневой каталог документов Web-сервера.

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 Web-сервер. Если у вас Web-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 четырем каталогам. Их имена сугубо условные, то есть 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 используются 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');
?>
```

Константа `SMARTY_DIR` содержит путь к Smarty. Если вы не знаете полный путь (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`.

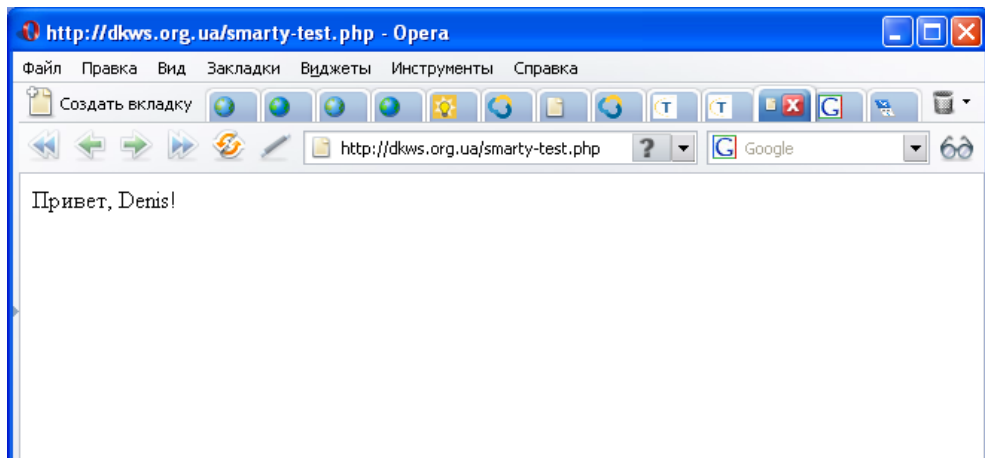


Рис. 23.1. Smarty работает!

23.3. Создание 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(SMARTYDIR . '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.

23.4. Разработка шаблонов Smarty

В этом разделе мы поговорим о правилах написания TPL-шаблонов Smarty. С основными правилами мы уже знакомы, поскольку уже применяли шаблонизатор. Но наш 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, то есть 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 РНР-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 редактирования этих значений Web-дизайнеру придется открывать РНР-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.

Для примера создайте файл конфигурации `main.conf`. Создать его следует в каталоге конфигураций (у нас это каталог `config`). Пример этого файла приведен в листинге 23.5.

Листинг 23.5. Пример файла конфигурации (`main.conf`)

```
title = "Мой сайт"
BgColor = "black"
BorderSkze = 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 данной 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 данных 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, с 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>

Таблица 23.1 (продолжение)

Модификатор	Описание
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 <code>\n</code> или комбинации символов <code>\r\n</code> на тег <code>
</code>
regex_replace	Выполняет поиск подстроки с помощью заданного регулярного выражения и производит замену найденной подстроки на строку, заданную вторым параметром: <pre>{ \$переменная regex_replace : "рег_выражение" : "строка" }</pre>
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: <pre>{ \$articleTitle strip } { \$articleTitle strip : "&nbsp;" }</pre>
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".

Таблица 23.1 (окончание)

Модификатор	Описание
	Третий параметр (может принимать true или false) определяет, как нужно обрезать строку — по границе слова (false) или по символу (true): <pre>{ \$articleTitle truncate}</pre> <pre>{ \$articleTitle truncate:30}</pre> <pre>{ \$articleTitle truncate:30:"..."}</pre> <pre>{ \$articleTitle truncate:30:"...", false}</pre>
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 абсолютно 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 `тер {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, то есть 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. Данные функции для описания условий используют квалификаторы (`<`, `>` и т. д.), которые должны быть отделены от остального содержимого пробелами. Квалификаторы аналогичны существующим в PHP, но допустим также 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

Таблица 23.2 (окончание)

Квалификатор	Альтернативный синтаксис	Эквивалент РНР	Пример	Описание
!	not	!	not \$a	отрицание
%	mod	%	\$a mod \$b	остаток от деления
is [not] div by		\$a % \$b == 0	\$a is not div by 4	возможно деление без остатка
is [not] even		\$a % 2 == 0	\$a is not even	[не]четно
is [not] even by		(\$a / \$b) % 2 == 0	\$a is not even by \$b	[не]четно значению
is [not] odd		\$a % 2 != 0	\$a is not odd	[не]нечетно
is [not] odd by		(\$a / \$b) % 2 != 0	\$a is not odd by \$b	[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 РНР-функции, например:

```
{if isset($var) }
    {$var}
{/if}
```

Функция {capture}

Используется для чтения в одну переменную содержимого шаблона для последующей обработки данного 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. Если вам интересно, 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 должен. А вот 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".

Функция `{assign}`

Рассмотрим самые полезные пользовательские функции. Одну мы уже знаем — `assign`. Напомню ее синтаксис:

```
{assign var="имя_переменной" value="значение"}  
{$имя_переменной}
```

Функция `assign` позволяет создать Smarty-переменную и присвоить ей значение.

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}`

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}`

Данная функция 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 данной функции. 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 out-  
put=$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 `{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 только один зависимый переключатель.

Функция `{html_image}`

Данная функция генерирует HTML-код для картинки. Вот пример использования:

```
{html_image file="photo.jpg"}
```

На выходе получим (при условии, что размер фото составляет 640×480):

```

```

Функция `{html_select_date}`

Очень удобная функция, генерирующая форму для выбора даты. У данной 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
<code>prefix</code>	string	<code>Date_</code>	Префикс названий переменных для передачи даты (переменные года, месяца, дня)
<code>time</code>	timestamp	текущее время	Время
<code>start_year</code>	string	текущий год	Начальный год. Можно указать относительно текущего года
<code>end_year</code>	string	аналогично <code>start_year</code>	Конечный год. Можно указать относительно текущего года
<code>display_days</code>	boolean	<code>true</code>	Если <code>true</code> , то будет выведен список дней
<code>display_months</code>	boolean	<code>true</code>	Если <code>true</code> , то будет выведен список месяцев
<code>display_years</code>	boolean	<code>true</code>	Если <code>true</code> , то будет выведен список лет
<code>month_format</code>	string	<code>%B</code>	Задаёт формат названий месяцев (см. модификаторы функции <code>date()</code>)
<code>day_format</code>	string	<code>%02d</code>	Задаёт формат названий дней (см. модификаторы функции <code>date()</code>)
<code>day_value_format</code>	string	<code>%d</code>	Задаёт формат отображения дней (см. модификаторы функции <code>date()</code>)
<code>year_as_text</code>	boolean	<code>false</code>	Если <code>true</code> , то значение года будет выведено текстом
<code>reverse_years</code>	boolean	<code>false</code>	Если <code>true</code> , то годы будут выведены в обратном порядке
<code>field_array</code>	string	<code>null</code>	Название переменной (например, <code>var</code>), которая будет содержать выбранные значения в виде массива: <code>var [Day], var [Year], var [Month]</code>

Таблица 23.3 (окончание)

Параметр	Тип	По умолчанию	Описание
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}
```

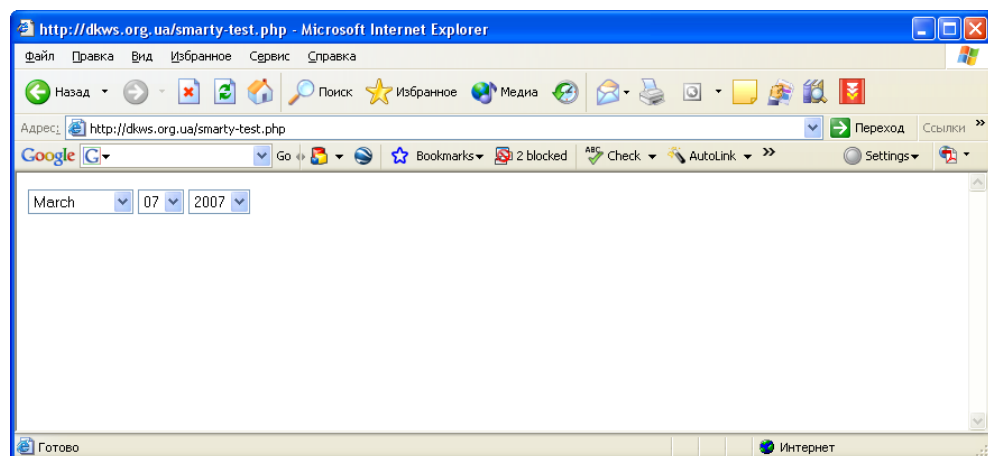


Рис. 23.2. Результат работы функции {html_select_date}

Функция `{html_select_time}`

Данная функция 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
<code>prefix</code>	string	<code>Date_</code>	Префикс названий переменных для передачи даты (переменные года, месяца, дня)
<code>time</code>	timestamp	текущее время в формате	Время
<code>display_hours</code>	boolean	<code>true</code>	Если <code>true</code> , то будут выведены часы
<code>display_minutes</code>	boolean	<code>true</code>	Если <code>true</code> , то будут выведены минуты
<code>display_seconds</code>	boolean	<code>true</code>	Если <code>true</code> , то будут выведены секунды
<code>display_meridian</code>	boolean	<code>true</code>	Если <code>true</code> , то будут показаны символы <code>am/pm</code> (для 12-часового формата)
<code>use_24_hours</code>	boolean	<code>true</code>	Использовать 24-часовой формат времени
<code>minute_interval</code>	integer	1	Интервал минут
<code>second_interval</code>	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 размер-

ность 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 нужно 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 рекомендуют ее не использовать, а все расчеты делать в сценарии, а в шаблон выводить только результат. Также не рассмотрели 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 они понадобятся очень редко.

23.5. Smarty для программиста

В этом разделе мы посмотрим на шаблонизатор Smarty глазами программиста, а именно рассмотрим специальные переменные и полезные методы класса Smarty.

23.5.1. Специальные переменные

Обращение к переменным Smarty осуществляется через объект `Smarty`, то есть 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>	Задает имя каталога для шаблонов. По умолчанию используется значение <code>./templates</code> , то есть вы вообще можете не указывать значение этого поля при создании сценария, если ваши шаблоны находятся в каталоге <code>templates</code>
<code>compile_dir</code>	Задает 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, то есть кэширование выключено. Рекомендуется включить кэширование, если ваши шаблоны генерируют большие объемы PHP-кода, в результате вы получите ощутимый прирост производительности
<code>cache_dir</code>	Задает каталог для кэша. По умолчанию <code>./cache</code>
<code>cache_lifetime</code>	Время жизни кэша в секундах. Для использования этого параметра нужно установить <code>caching</code> в 1. Если установить для <code>cache_lifetime</code> значение <code>-1</code> , то время жизни кэша будет неограниченное
<code>config_overwrite</code>	Если <code>true</code> , то переменные из конфигурационного файла будут перекрывать все остальные. По умолчанию <code>false</code>

Таблица 23.5 (продолжение)

Переменная	Описание
<code>security</code>	Если установлен в <code>true</code> , позволяет предотвратить взлом вашего сценария через шаблоны. Рекомендуется устанавливать в <code>true</code> , если шаблоны редактируют не совсем надежные люди. Параметр запрещает выполнение PHP-кода с помощью функции <code>{php}</code> ; в функции <code>{if}</code> запрещает использование PHP-функций, кроме указанных в <code>\$security_settings</code> ; шаблоны могут быть подключены только из каталога, который указан в <code>\$secure_dir</code> ; запрещено использовать PHP-функции в виде модификаторов, кроме тех, которые указаны в <code>\$security_settings</code>
<code>secure_dir</code>	Массив всех каталогов, которые считаются безопасными
<code>security_settings</code>	Задаёт параметры безопасности. Массив, содержащий следующие элементы: <code>PHP_HANDLING</code> — если <code>true</code>, то параметр <code>\$php_handling</code>, который определяет, как будет выполняться PHP-код, не проверяется на безопасность; <code>IF_FUNCS</code> — массив имен PHP-функций, которые могут быть использованы в условиях <code>IF</code>; <code>INCLUDE_ANY</code> — если <code>true</code>, то любой шаблон может быть подключен независимо от списка <code>secure_dir</code>; <code>PHP_TAGS</code> — если <code>true</code>, то теги <code>{php}</code> разрешены в шаблонах; <code>MODIFIER_FUNCS</code> — массив имен PHP-функций, которые можно использовать в качестве модификаторов; <code>ALLOW_CONSTANTS</code> — если <code>true</code>, то допускаются константы вида <code>{\$smarty.const.name}</code>. Из соображений безопасности установлен в <code>false</code>: <pre>\$smarty->security_settings [PHP_HANDLING]=false; \$smarty->security_settings [IF_FUNCS]=array('isset');</pre>

Таблица 23.5 (окончание)

Переменная	Описание
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 больше и не потребуются какие-либо другие методы. 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/>.



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 3 раза — для каждого сотрудника. В этом тривиальном случае ошибку допустить практически невозможно, если быть внимательным, 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 РНР 4, хотя весьма ограниченная. И только РНР 5 обладает уже полноценной поддержкой ООП.

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();

?>
```

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, то есть может использоваться как для свойств, так и для методов класса.

24.3. Конструкторы и деструкторы класса

Конструктор — это специальный метод, выполняющий инициализацию объекта, 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 5/6 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, то есть 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 5/6.

24.5. Новые возможности PHP 5/6

24.5.1. Область видимости членов класса

В PHP 5/6 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 производных (дочерних) классов. Вы не можете установить значение защищенного свойства непосредственно. 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 5/6 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 данный 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()`. Данный метод есть у каждого класса по умолчанию, который 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;              //клонировем класс

// выведет 0, потому что __clone() присвоит свойству id ноль
print $obj->id . "\n";
?>
```

24.5.5. Обработка исключительных ситуаций

В PHP5 появилась возможность обработки исключительных ситуаций. Для обработки исключительной ситуации используется 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');

}
?>
```

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()`).

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 использовать 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 5/6 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.

Давайте рассмотрим пример использования пространства имен. Пространство имен объявляется инструкцией `namespace`:

```
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 6 все проходит гладко, только при использовании нашего класса нужно указывать наше пространство имен:

```
$xml = new DenSpace::XMLWriter();
```



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 с помощью формы передал их.

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');  
}  
  
?>
```

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 РНР-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 РНР 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 ...])
```

Данная функция принимает переменное число аргументов, то есть 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, то есть до обращения к функции `session_start()`:

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

```
session_start();
```

Если имя сессии не задано (то есть 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 делает 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 зарегистрирует 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). Данные сценарии будут 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 MD5: просто при сравнении не будем вызывать MD5 для шифрования этого пароля, ведь мы будем сравнивать зашифрованное значение с зашифрованным. Сказанное иллюстрирует листинг 25.2.

Листинг 25.2. Полная версия `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');  
}  
  
>
```

Для упрощения сценария я не стал исключать из него HTML-код, 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 наша ссылка.

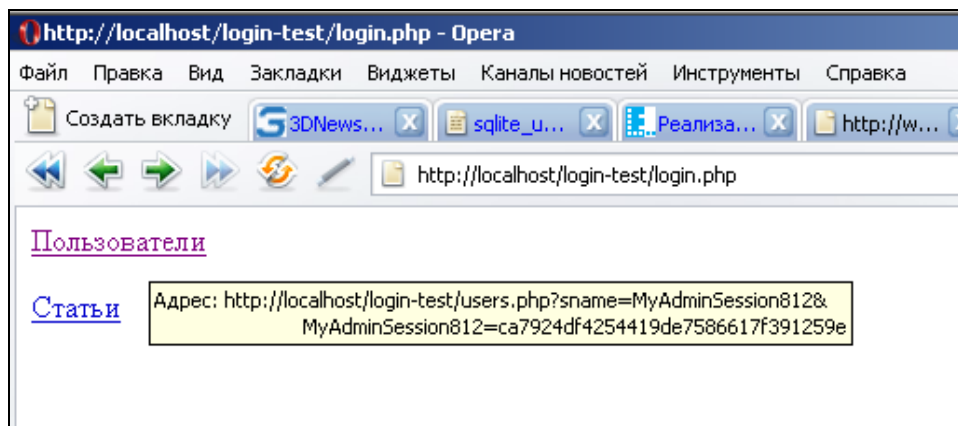


Рис. 25.1. Передача имени и идентификатора сессии другому сценарию

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, так как 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('Данный 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');

?>
```

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 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, вам интересно, 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 сессии, желательно шифровать.



Рис. 25.2. Результат работы сценария `users.php`

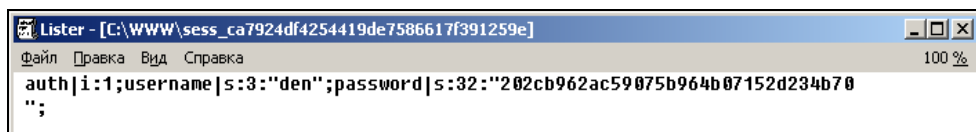


Рис. 25.3. Наш файл сессии



Введение в PEAR

26.1. Серьезные проекты и PEAR

PEAR (PHP Extension and Application Repositor) — это база расширений и приложений для PHP. Чтобы понять, что представляет собой PEAR, нужно рассмотреть один пример. Предположим, что у вас есть довольно серьезный проект, состоящий, скажем, из 100 PHP-сценариев (на самом деле это не очень много). Ваш проект использует сервер 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 изменять все 100 сценариев. Причем простая замена имен функций с `mysql_` на `ora_` не пройдет.

Что же делать? Первое, что приходит в голову, — создать класс, напоминающий представленный ниже:

```
<?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 методы этого класса при обращении к БД. Тогда для перехода на другой сервер баз данных вам нужно будет изменить только этот файл. Это намного проще, чем редактировать все ваши 100 сценариев. К тому же ваш проект станет намного более универсальным, ведь можно написать несколько разных клас-

сов — 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 вы можете использовать класс `DB` из базы PEAR! Все уже сделано за вас — вам нужно 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*

Первым делом 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>

На прилагаемом диске в каталоге `Glava_26` вы найдете версию 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 Web-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 за-

хламлять сервер ненужными файлами. Нужно отметить, что это неполная установка PEAR, с полной установкой вы можете ознакомиться по адресу:

<http://pear.php.net/manual/ru/installation.php>

После этого скачайте последнюю версию пакета DB (как обычно, текущая версия доступна в каталоге Glava_26 прилагаемого диска):

<http://pear.php.net/package/DB/download>

Распакуйте архив DB. Опять мы не будем копировать на сервер все файлы, а скопируем только те, которые нам необходимы. На Web-сервер нужно скопировать только подкаталог DB (он появится, когда вы распакуете архив) и файл DB.php.

Далее откройте файл DB.php и в самом начале найдите строчку:

```
require_once 'PEAR.php'
```

Удалите или прокомментируйте ее. Данная 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
        ...
```

Приступим к редактированию файла test_pear.php. Листинг этого файла с моими комментариями представлен ниже.

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');

//подсоединяемся 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']);
    }
}
?>
```

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'));
```

Далее мы 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) я разместил документацию на русском языке в каталоге Glava_26 прилагаемого диска в архиве pear_manual_ru.zip.



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. Можно комментировать старый код, после чего писать новый. Всегда можно потом удалить новый код и раскомментировать старый, но зачем себя 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 пока не все, да и оболочку системы контроля версий использовать удобнее.
- ❑ В случае групповой работы над проектом вы сможете видеть, 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 мо-

мент многими программистами) является CVS (Concurrent Versions System). Прочитать о CVS можно 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 были предназначены для работы в UNIX-подобных системах (Linux, FreeBSD), а для Windows разработаны лишь графические клиенты для этих систем. Это означает, что вам понадобится выделенный UNIX-сервер для хранения репозитория — специального хранилища, где хранятся ваш проект и история изменений. Для небольших компаний (которые обычно используют Windows на своих рабочих станциях) или фрилансеров выделенный сервер для проектов экономически нецелесообразен. Можно, конечно, использовать бесплатные интернет-серверы для хранения своего репозитория, 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. Да и 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. К тому же не нужно забывать 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, 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

Последняя на момент написания этих строк версия TortoiseHG находится в каталоге software прилагаемого диска. 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 появившемся окне нажмите кнопку **Create** (рис. 27.2). После чего вы увидите сообщение, что репозиторий создан.

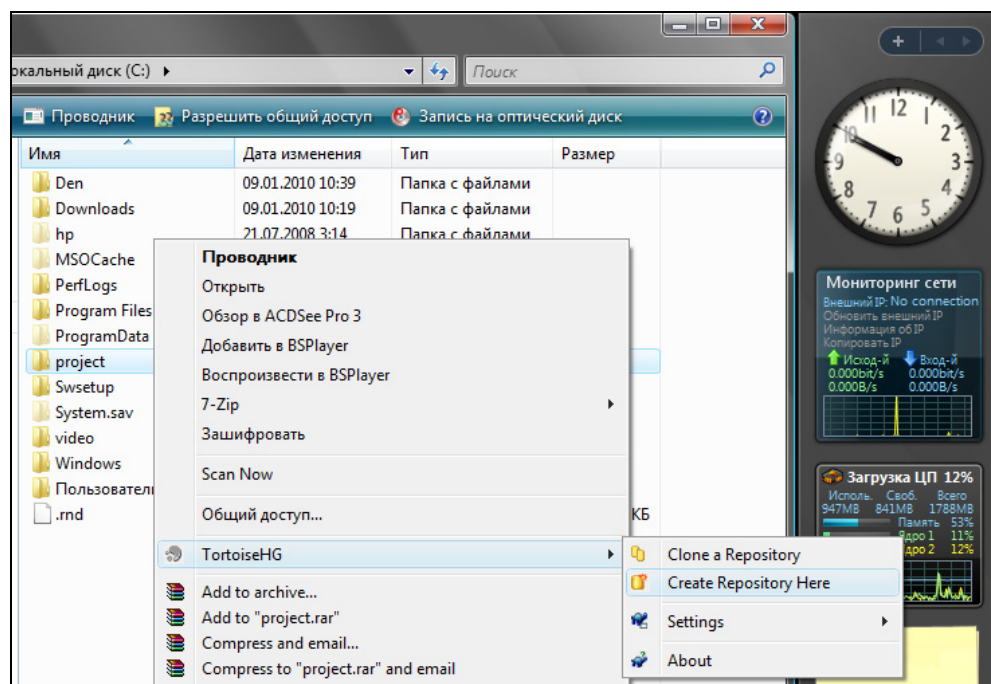


Рис. 27.1. Создание репозитория проекта

Следующий шаг — создание первой версии проекта. Для этого опять щелкните правой кнопкой на папке project и выберите команду **HG Commit** (рис. 27.3).

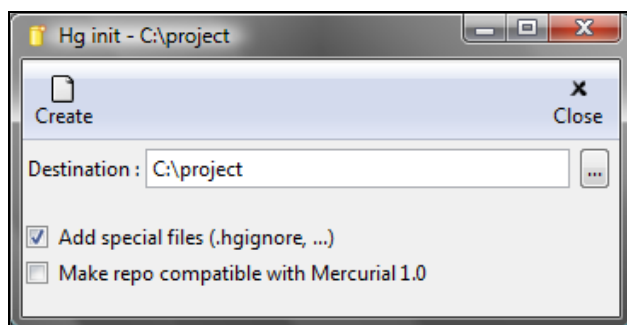


Рис. 27.2. Нажмите кнопку **Create**

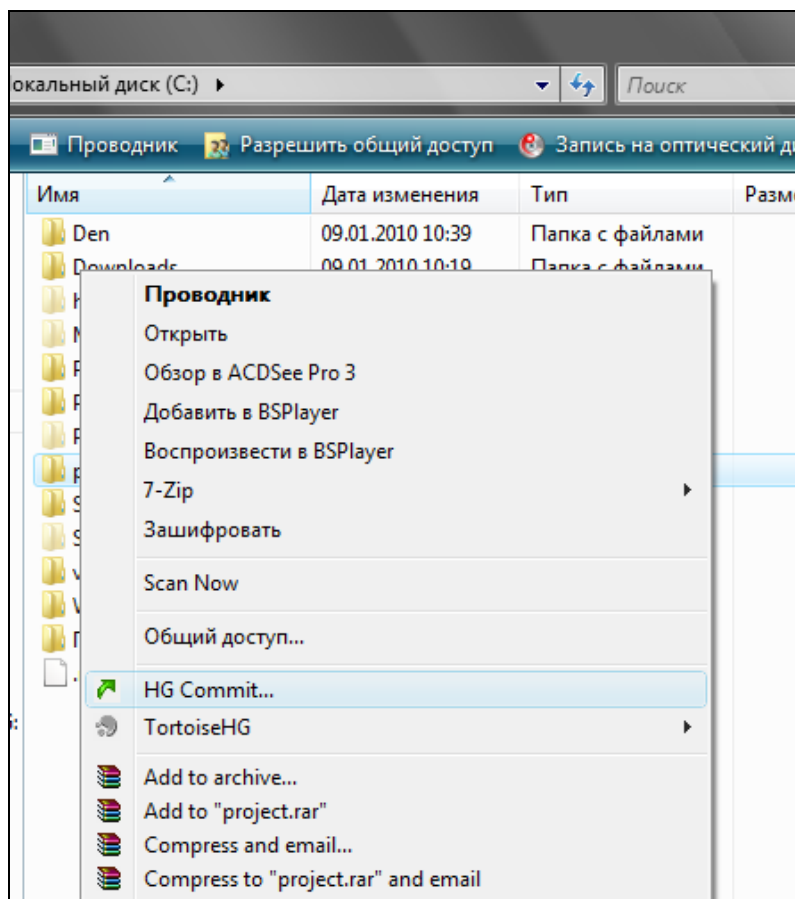


Рис. 27.3. Выберите команду **HG Commit**

В появившемся окне (рис. 27.4) выберите файлы, по которым нужно отслеживать изменения. У нас пока есть только один файл, поэтому выберем его (`index.php`). После этого нажмите кнопку **Commit**.

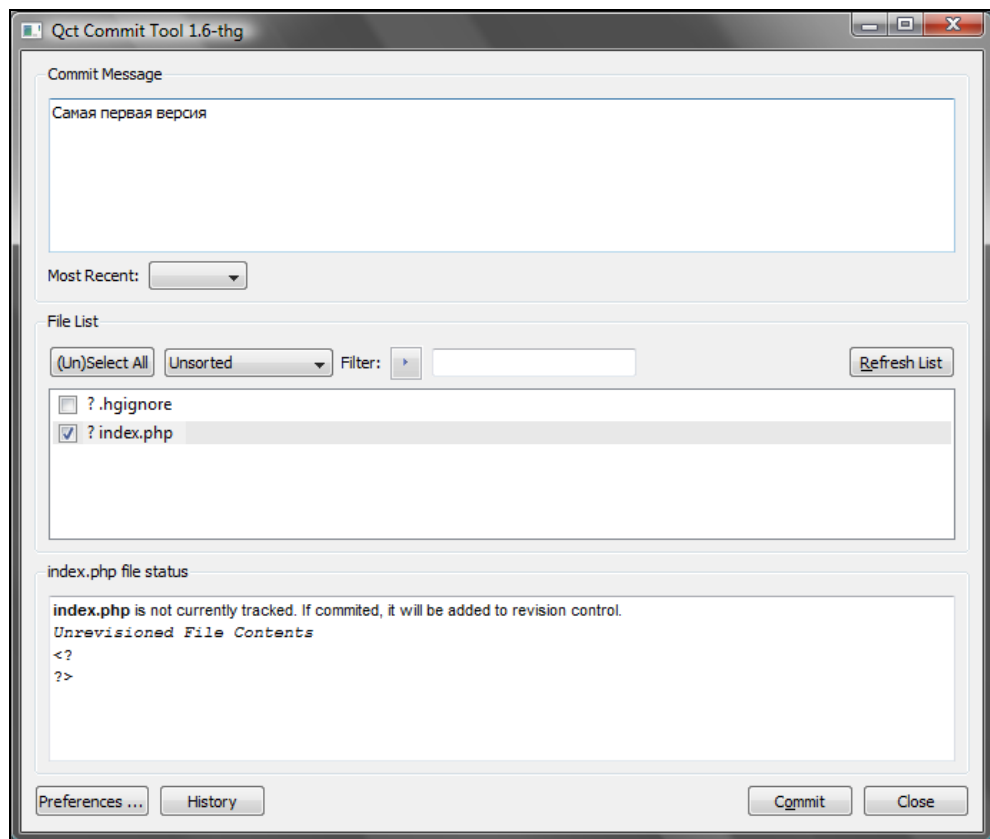


Рис. 27.4. Выберите файлы, по которым нужно отслеживать изменения

27.3. Просмотр внесенных изменений

Откройте любой текстовый редактор и внесите изменения в файл `index.php`. После этого щелкните правой кнопкой мыши на каталоге `project` и выберите команду **TortoiseHG | Visual Diff** (рис. 27.5).

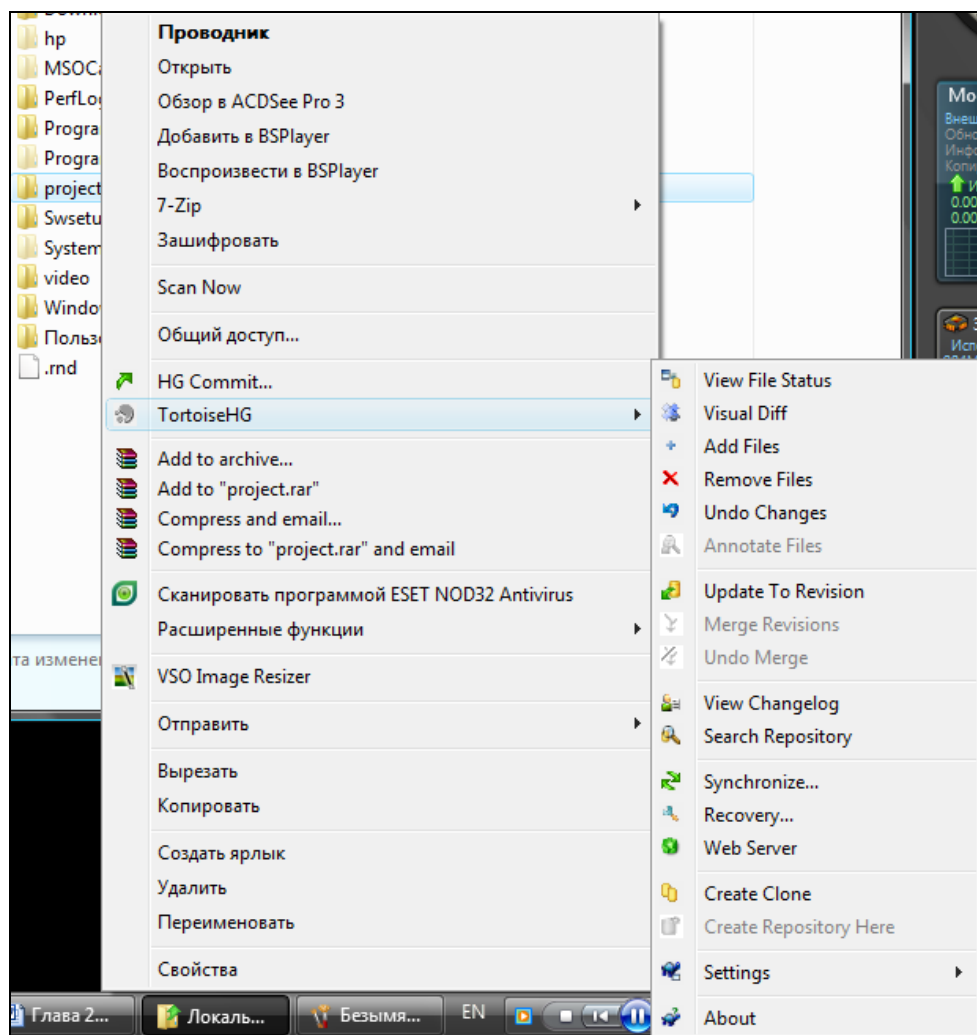


Рис. 27.5. Контекстное меню TortoiseHG

Появившееся 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 появившемся окне выберите файлы, в которых нужно отменить изменения.

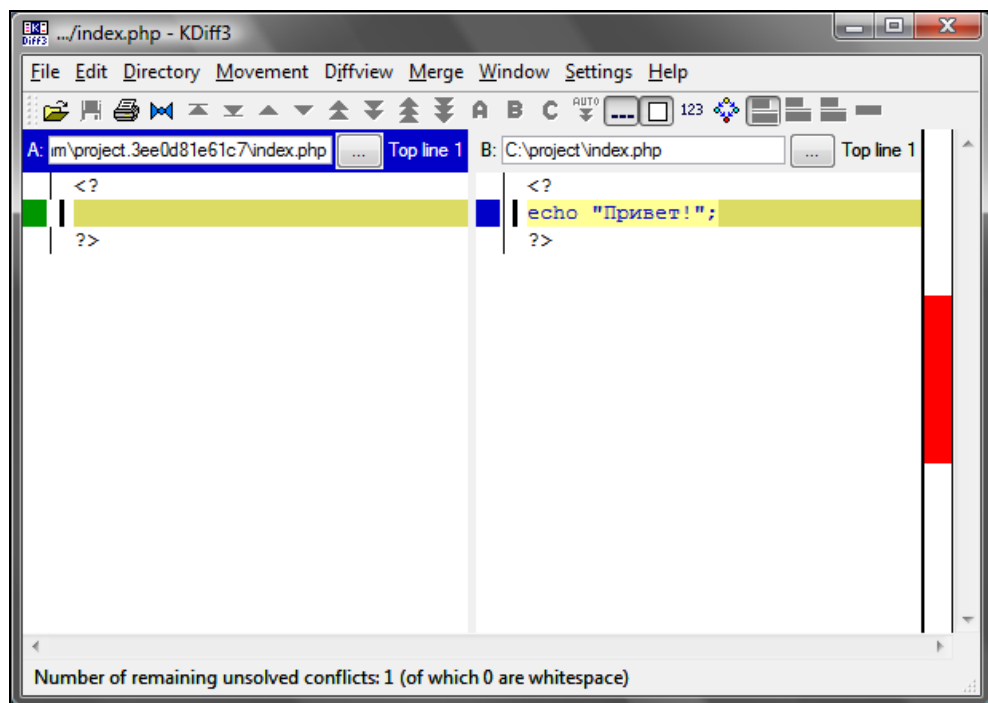


Рис. 27.6. Просмотр изменений в файле

Для добавления новых файлов в проект используется 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.



Тестирование РНР-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 РНР будет намного труднее. Во-первых, РНР 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 ста-

ла 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 RHP — 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 RHP начинается имя каждой переменной. Почему это так, думаю понятно. В том же C++ или Pascal нужно сначала объявить переменные, то есть указать имя переменной и ее тип (можно сразу задать и ее значение). А в RHP, 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, устанавливали точку останова. Отладчик выполнял программу до заданной точки останова. Затем 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, и, 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/>

По данному адресу доступна как сама система, так и модуль для системы Eclipse, если вы ею пользуетесь. Если Интернет в данный момент недоступен, то версию SimpleTest 1.0.1 вы найдете на прилагаемом диске в каталоге Glava_28.

Разберемся, как работает система. Распакуйте скачанный архив в каталог simpletest. В каталоге simpletest/docs находится подробное описание системы, а в simpletest/test много различных примеров.

Мы рассмотрим простейший пример использования данной системы. Создайте файл ex.php и поместите его в каталог simpletest. Код данного файла приведен в листинге 28.1. Внимательно читайте комментарии, чтобы понять, что и как происходит. Можно, конечно, было описать методы отдельно, но вместе с реальным кодом комментарии воспринимаются проще: как говорится, лучше один раз увидеть, чем сто раз услышать.

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);  
  
?>
```

Запустите сценарий `ex.php` в окне браузера. Результат его выполнения представлен на рис. 28.1.

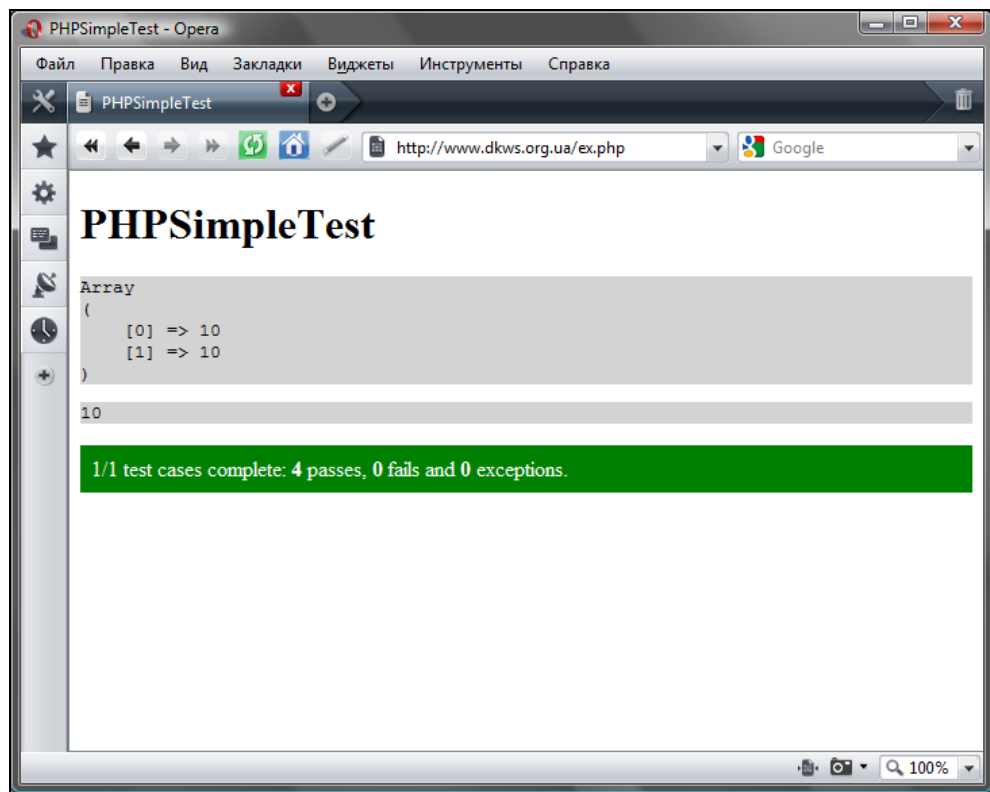


Рис. 28.1. Результат проверки кода

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 "общего развития" проделайте следующий "эксперимент": перед строкой

```
$this->assertTrue($a==10);
```

добавьте строку

```
$a = 9;
```

Другими словами, мы сделали все возможное, чтобы наша функция "провалила" тест. Запустите еще раз сценарий и посмотрите на результат (рис. 28.2).

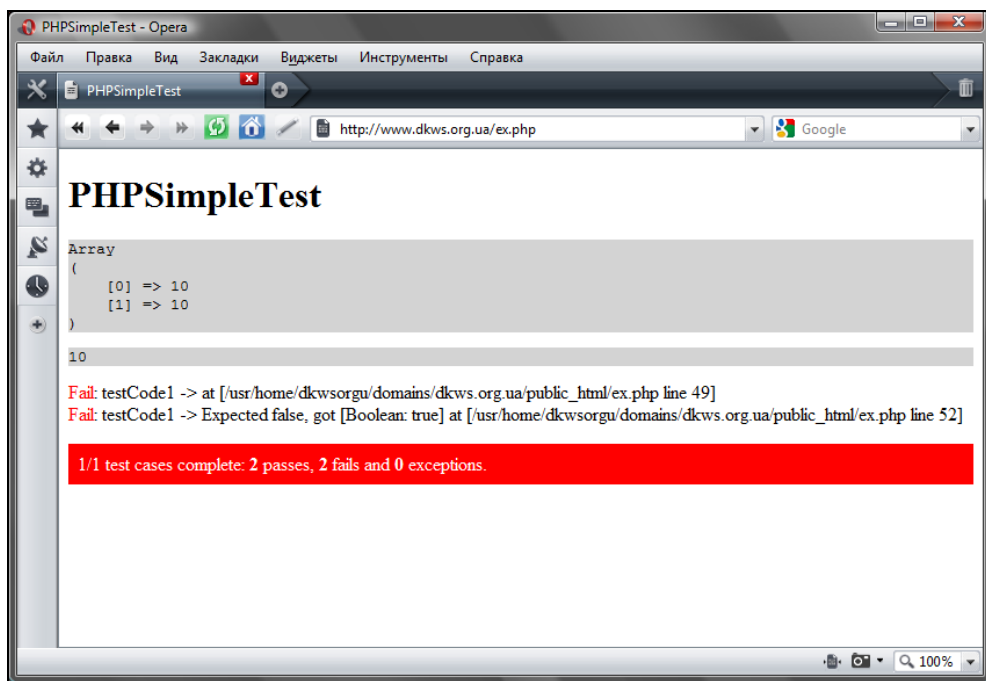


Рис. 28.2. Функция "провалила" два теста

Поскольку мы присвоили переменной `$a` значение 9, то два последних теста оказались не выполненными:

- ❑ 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 ваших 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 данный файл может находиться в каталоге `/etc` или в `/etc/php5/cli` (все 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



РАЗДЕЛ 7

Разработка основных элементов сайта



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;
- ☐ Web-интерфейс почтовой службы — ведь сначала нужно загрузить файл на сервер, а только после этого прикрепить его к письму;
- ☐ Web-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. В этой главе мы поговорим о том, как реализовать загрузку своими силами, не прибегая к использованию сторонних классов для загрузки файлов.

Для загрузки файлов на сервер существуют 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

Код формы загрузки (файл upload.html) и сценария upload.php (см. листинг 29.3) вы найдете на прилагаемом диске в каталоге Glava_29.

Multipart-форма отличается от обычной прежде всего наличием параметра ENCTYPE="multipart/form-data". Но это еще не все. Второй обязательный атрибут 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.

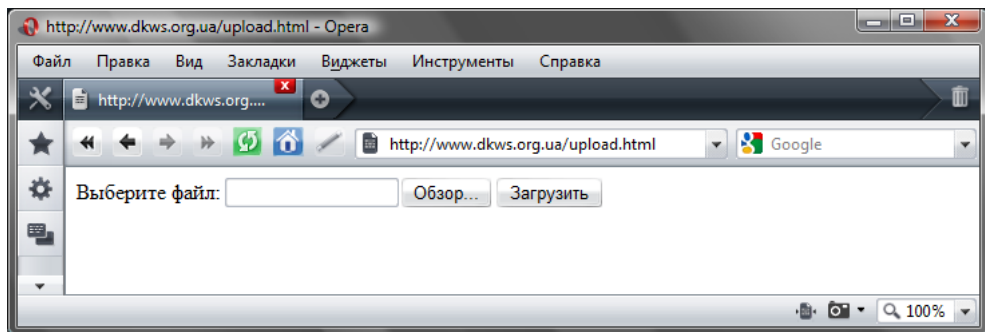


Рис. 29.1. Multipart-форма в окне браузера

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 Web-сервер Apache.

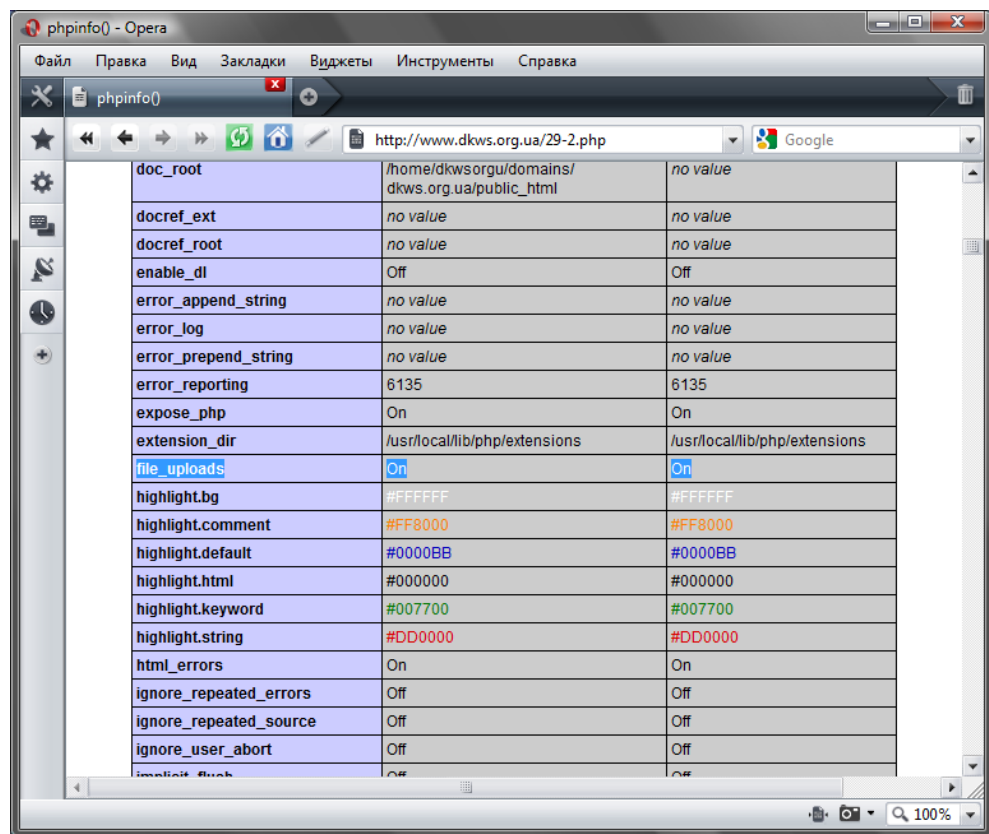


Рис. 29.2. Загрузка файлов включена

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). Чуть далее вы найдете имя временного каталога и максимальный размер загружаемого файла.

Листинг 29.2. Файл `test.php`

```
<?php
```

```
phpinfo();
```

```
?>
```

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);

}

}
```

?>

Теперь проанализируем сценарий. Данный сценарий 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 временного каталога.

29.3. Загрузка нескольких файлов

Возможно, вам потребуется реализовать автоматическую загрузку нескольких файлов. Особых проблем с этим нет. Просто измените нашу 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>
```

При этом наш 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 ли 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
    }

}
...
```

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 примерно таким: `php05hhGi`. Первые три символа будут `"php"`, а остальные символы 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, а этого нельзя делать. Можно проверять существование временного файла и, если такой файл уже существует, попытаться 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 добавить временную метку (timestamp) к имени временного PHP-файла. Получается, что у нас есть случайно сгенерированная последовательность символов (ее генерирует PHP) и временная метка. Вероятность того, что будет сгенерировано две одинаковых 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. Проблемы при загрузке файлов

Если вы сделали все, как было описано в этой главе, трудностей с загрузкой файлов у вас не должно быть.

Но все же могут возникнуть две проблемы:

1. 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.
2. 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
```



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, то есть передать имя пользователя и пароль. Для этого служит функция `ftp_login()`:

```
bool ftp_login (resource ftp_stream,  
               string username,  
               string password)
```

Данной 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 вам 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 используется 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)
```

Данная 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 закачивает 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 закачаться 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 — закидывает 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 используется 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 иницирует 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)
```

Данной 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 27.1) будет подключаться к FTP-серверу, регистрироваться и выводить содержимое текущего каталога (рис. 30.1). Второй пример будет загружать файл с сервера (листинг 27.2).

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);

?>
```

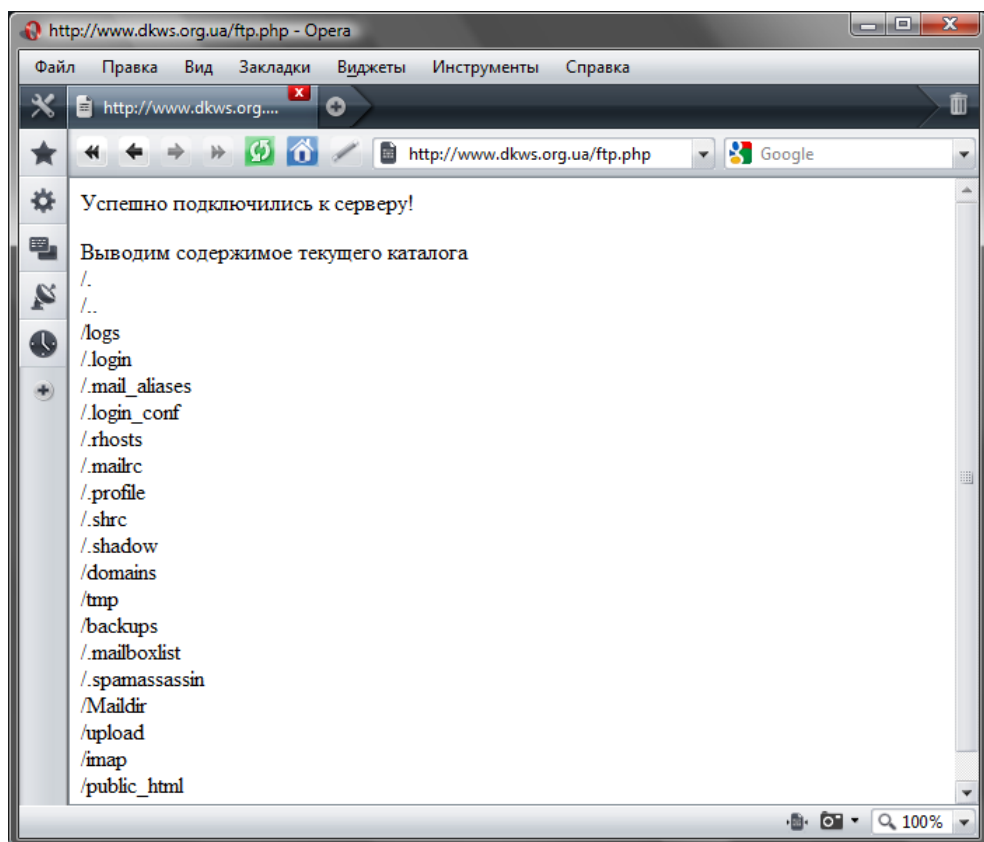


Рис. 30.1. Результат работы сценария (листинг 30.1)

Листинг 30.2 иллюстрирует пример загрузки файла с сервера.

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);

?>
```



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 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. Данный класс, а также все остальные классы и листинги из этой главы можно найти на прилагаемом диске в каталоге `Glava_31`. 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) мы будем использовать тщательно проверенную временем и мною версию `HtmlMimeMail`. К тому же, данная версия мною была немного переработана и усовершенствована. В частности, было исправлено перекодирование русских символов. Архив с исправленной версией называется `mime.zip`. Также в файле `mime.zip` вы найдете 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 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()` используется 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, то есть с помощью этого метода вы указываете, что данная картинка относится к HTML, а не является просто вложением.

С помощью метода `addAttachment()` можно добавить вложение к письму. 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 данного 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'`).

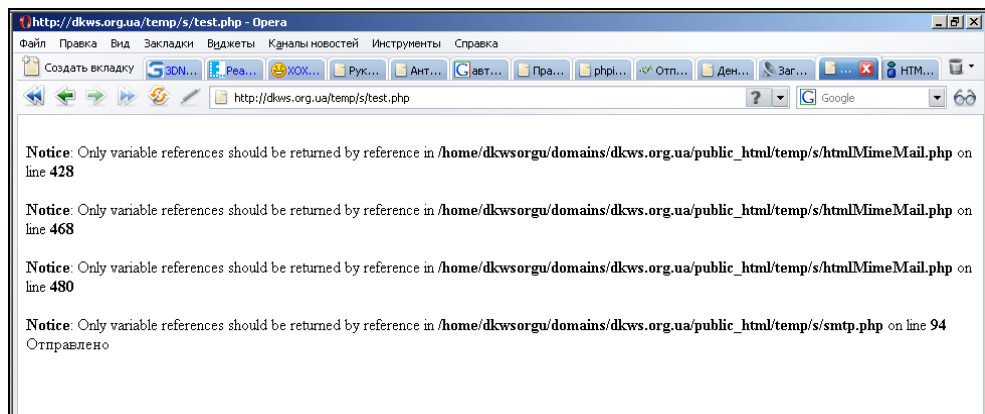


Рис. 31.1. Отправка сообщения

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()`.

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.

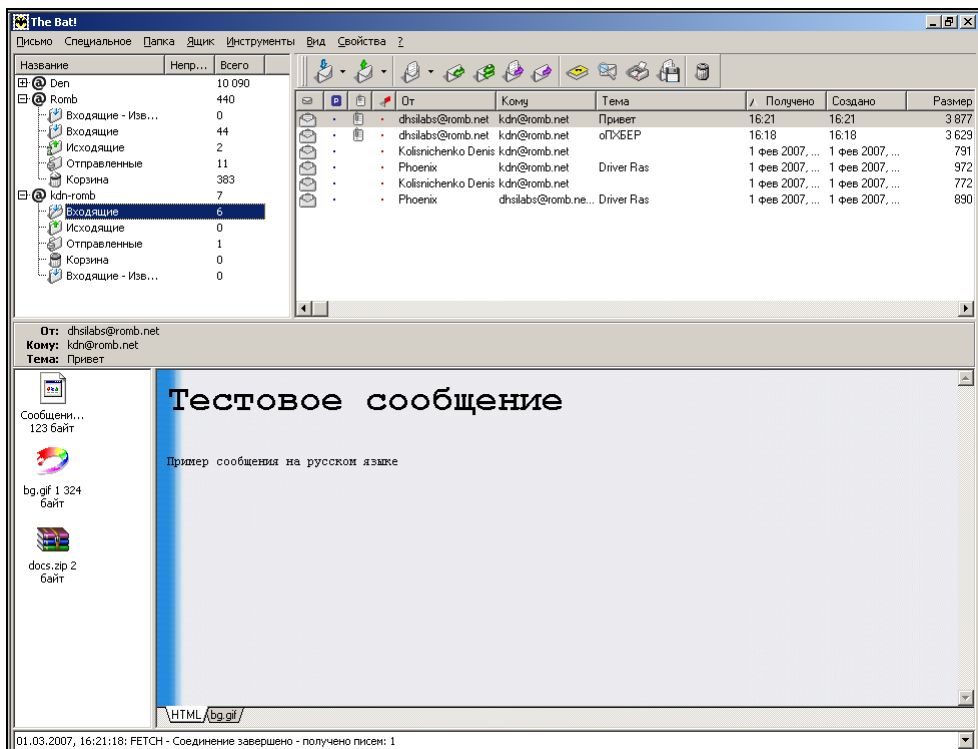


Рис. 31.2. Отправленное сообщение

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 данное решение не поможет, то вам придется задействовать SMTP и редактировать `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. Данные 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 Multipurpose Internet Mail Extension — многоцелевые расширения почты Интернета. Правда, сегодня MIME применяется не только в системе электронной почты. Раньше 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

Таблица 31.1 (окончание)

Базовый тип	Описание	Примеры подтипов
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. 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 не удалять сообщения при получении, 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 накладно сделать — уж очень много нюансов. Гораздо проще применить готовые решения.

Здесь мы рассмотрим класс POP3, доступный в каталоге Glava_31 прилагаемого диска. 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 серверу 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 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 используется 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)
```

`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 31.3.

Листинг 31.3. 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();

?>

```

Результат работы сценария представлен на рис. 31.3.

Надеюсь, что материал данной главы поможет вам в разработке собственного почтового интерфейса.

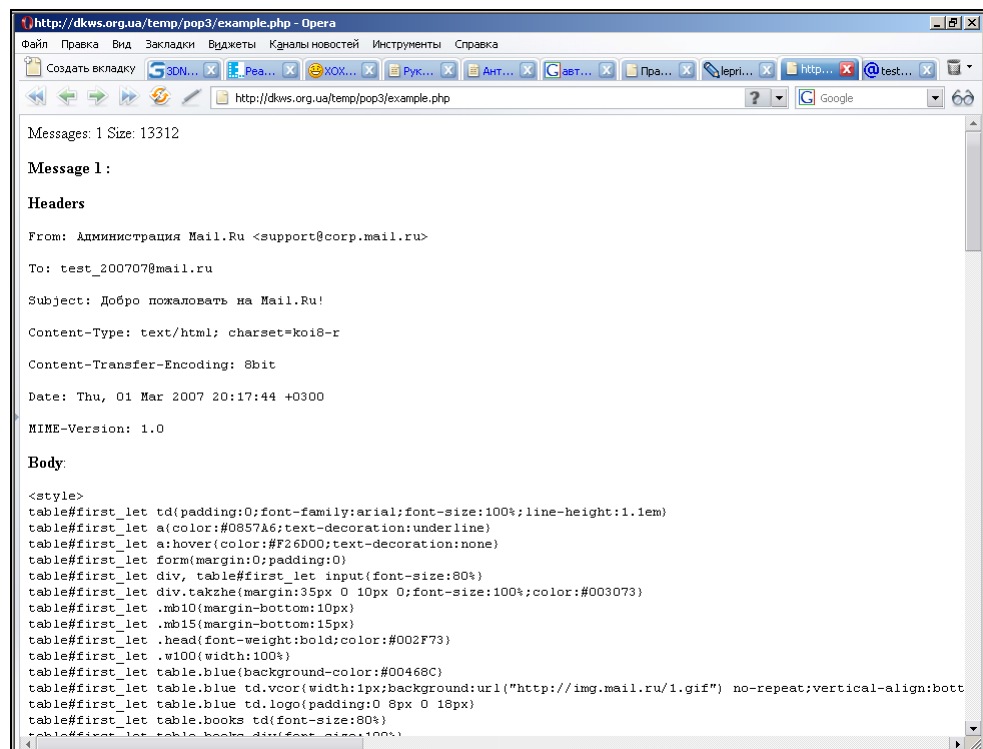


Рис. 31.3. Получение сообщений с помощью класса POP3



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 просмотреть их. Агрегаторы очень удобны, поскольку с их помощью можно получить все нужные вам новости сразу (в настройках агрегаторов можно указать несколько новостных сайтов) и 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, по-
лучив 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 (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 знаем формат RSS-файла, мы можем приступить к написанию сценария парсинга RSS-файла. Но хочется ли вам этим заниматься? Мне — нет. Особенно когда уже есть много готовых PHP-классов, выпол-

няющих всю "грязную" работу. Все, что вам нужно, — это подключить класс к своему сценарию и вывести новости.

Один из таких классов — `lastRSS`, который вы найдете в каталоге Glava_32 прилагаемого диска. Если вас интересует последняя версия, то она 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, поэтому дефицита RSS-каналов нет. Наоборот, есть проблема выбора. Уж слишком их много. В этой главе в качестве примера я буду импортировать новости с сайта [www.itnews.com.ua](http://itnews.com.ua). На сайте <http://itnews.com.ua/export/> есть список доступных RSS-каналов (рис. 32.1).

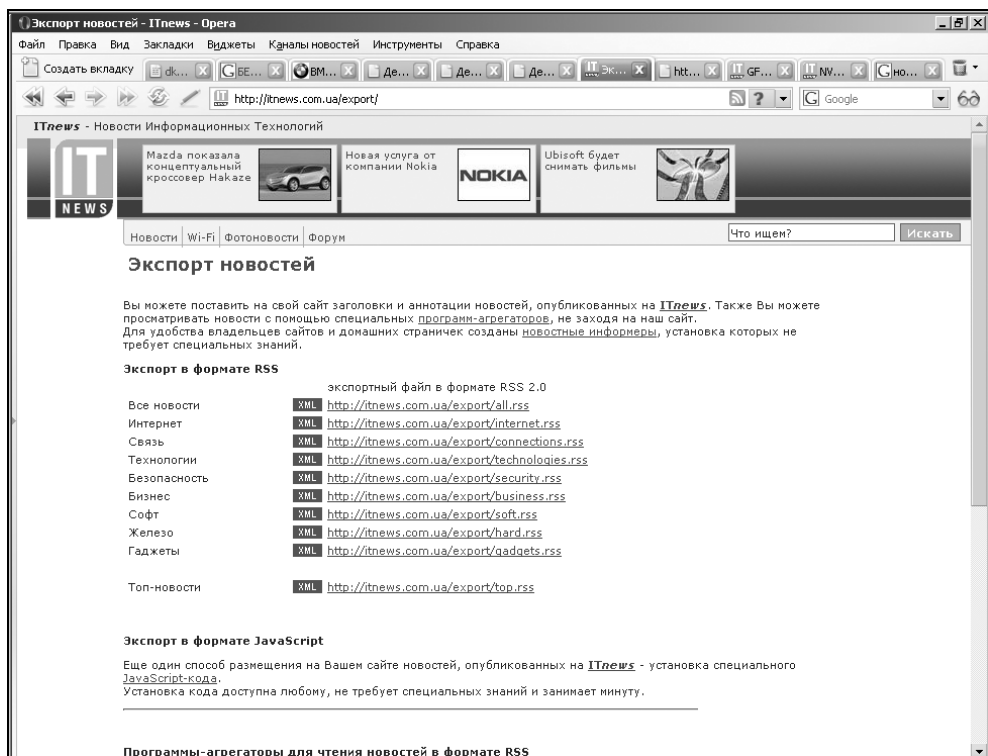


Рис. 32.1. Список доступных RSS-каналов

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, у вас на 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. Данный сценарий будет использовать 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>";
}
}
?>
```

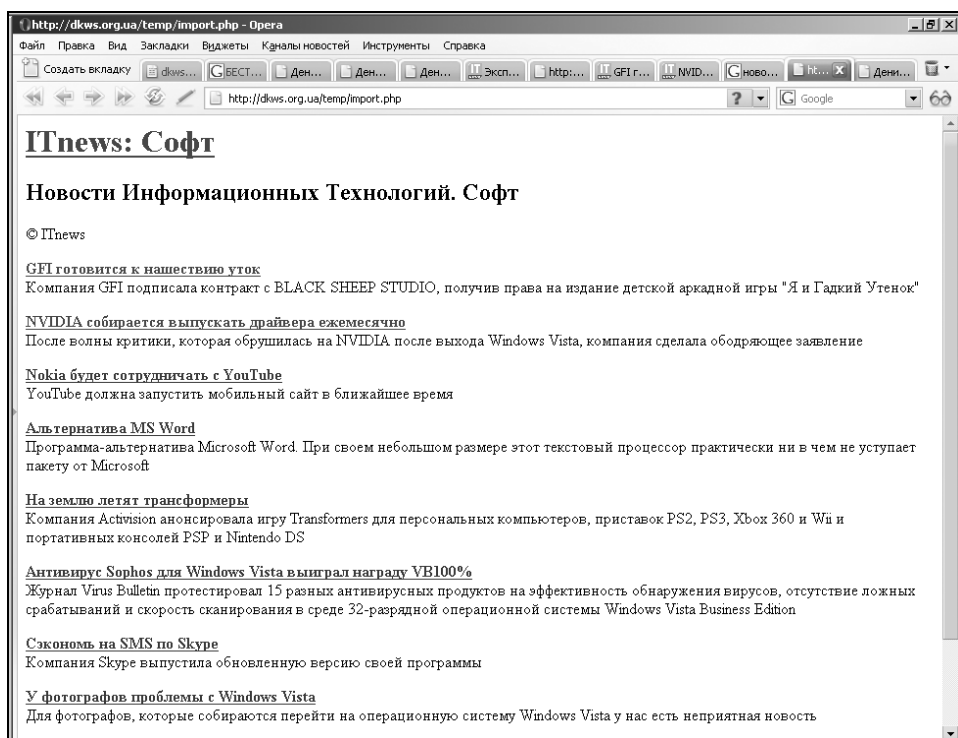


Рис. 32.2. Результат работы сценария import.php

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 выводим 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).

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 данный 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 в нужном месте потребуется просто 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 статьи уведут пользователей с нашего сайта на сайт-

источник новостей. В следующий раз пользователь просто зайдет на сайт-источник, и о нашем сайте забудут. Ведь так? Поэтому желательно написать граббер новостей, который будет подключаться к сайту-источнику, "забирать" новость и выводить ее на нашем сайте в нашем дизайне.

Приступим к написанию подобного сценария. Создание граббера я буду иллюстрировать на примере сайта **itnews.com.ua**. Заметим, что HTML-код у других сайтов будет иным, поэтому с остальными сайтами данный вариант работать не будет. Но по "образу и подобию", изменив пару строк кода, вы сможете адаптировать данный 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 нашего сайта. Как видите, все очень просто. Напишем такой сценарий (пусть он будет называться `news.php`, листинг 32.4).

Листинг 32.4. Сценарий `news.php` — граббер новостей

```
<?

// получаем Web-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;

?>
```

Гrabбер можно вызвать так:

`http://localhost/news.php?u=http://itnews.com.ua/30519.html`

Результат работы сценария представлен на рис. 32.3.

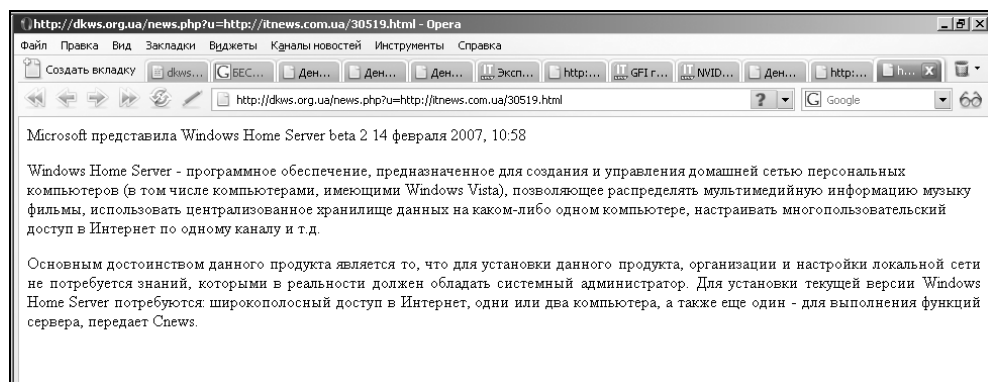


Рис. 32.3. Результат работы grabбера новостей

Но это еще не все. Ведь на главной странице сайта все равно у нас выводятся ссылки на сайт-источник новостей, а не на сценарий news.php. Поэтому сейчас вернемся к тому месту, где вы 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 наш 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.

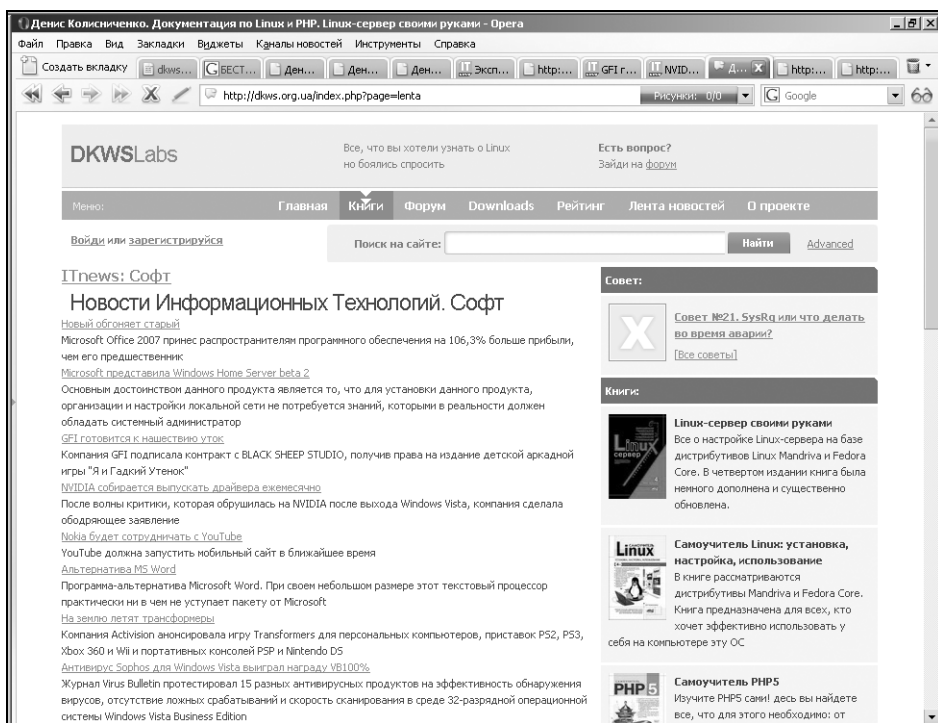


Рис. 32.4. Импорт списка новостей: конечный результат

ПРИМЕЧАНИЕ

Наш граббер новостей никак не нарушает авторские права, поскольку мы не скрываем источник новостей: 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.



Рис. 32.5. Импорт новости



Поиск с использованием регулярных выражений

33.1. Нужно что-то найти...

Иногда нам нужно найти информацию, которая соответствует определенным критериями, но что именно мы ищем, точно мы не знаем. Вот небольшой пример. Предположим, что у нас есть HTML-файл. Нам необходимо найти в нем все 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 все 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 PHP позволяет произвести такой поиск, вызвав всего лишь одну функцию. Этой функции нужно передать образец, то есть "объяснить", что мы ищем. Данным образцом является регулярное выражение. Поначалу все регулярные выражения вам 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 RegEx

PHP поддерживает два языка регулярных выражений: RegExpr (классические регулярные выражения) и PCRE (Perl-Compatible Regular Expression, совместимые с Perl выражения). В этой книге мы будем рассматривать только RegExpr. Во-первых, RegExpr — это классика, а классику необходимо знать. Ведь в других языках программирования вам тоже могут пригодиться регулярные выражения. И не думаю, что какой-то другой язык программирования будет поддерживать PCRE. Наоборот, скорее всего это будут классические выражения. Во-вторых, классические выражения более просты. Чтобы понимать PCRE, вам необходимо знать RegExpr, поэтому, как ни крути, а RegExpr все равно пришлось бы рассмотреть. В-третьих, RegExpr работают быстрее, чем PCRE. Но с другой стороны, по сравнению с классическими выражениями, возможности PCRE шире. Как видите, в любом случае вам сначала потребуется познакомиться с RegExpr, 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, то есть 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 за-

считано совпадение. К простым символам относятся только буквы и цифры, то есть A–Z, a–z, 0–9.

Символы `. ? * + { } [ ] | $ ^` называются управляющими. Если нужно найти строку, содержащую управляющие символы, то перед каждым управляющим символом нужно добавить обратный слэш, например:

```
\[text\]
```

Данное выражение будет соответствовать строке `[text]`. Но, согласно синтаксису РНР, перед символом `\` требуется указывать еще один обратный слэш, поэтому получаем следующее выражение, которое нужно передать РНР-функции:

```
\\[text\\]
```

Помните об этом правиле, поскольку один пропущенный слэш может стать причиной того, что якобы правильно написанные регулярные выражения не работают.

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
```

Все просто: нужно в квадратных скобках перечислить 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 RegEx

Макрос	Соответствие
<code>[:alpha:]</code>	Буква
<code>[:digit:]</code>	Цифра

Таблица 33.1 (окончание)

Макрос	Соответствие
<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]`

Данное выражение означает: все, кроме цифр.

33.3. Управляющие конструкции

Сейчас мы познакомимся с тремя управляющими конструкциями: квантификаторами, альтернативным оператором и скобками.

33.3.1. Квантификаторы

Квантификатор — это символ, использующийся 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. Звездочка `*` означает "ноль или более символов". Читаем определение квантификатора, особенно последнюю часть, — "символов, которые идут перед ним". Перед символом `*` в нашем случае стоит `0`. Следовательно, наше выражение будет соответствовать строке, в которой ноль или более символов нуля и которая заканчивается символом `7`: `7` (ноль символов нуля), `07`, `007`, `0007` и т. д.

Квантификатор `+` задает одно или более повторений (`*` 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.

ПРИМЕЧАНИЕ

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`. Данное 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 RegEx-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, данная функция используется для поиска совпадений. Но намного чаще приходится выполнять замену совпадения на другую строку. Если бы не было других РНР-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]`. 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 PHP-код программы, выделив имена переменных полужирным. Другими словами, нашей программе требуется найти все переменные и перед выводом обернуть 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 будем 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>";  
  
?>
```

Переменная `$php_code` — это наша исходная строка. Первый параметр функции `ereg_replace()` обеспечивает поиск переменных. Замена (второй пара-

метр) будет у нас непростая. Если бы мы знали, на какую именно строку заменять, мы бы просто написали строка, но мы ведь не знаем, какие у нас будут переменные — \$var, \$a или \$BDC. Поэтому мы используем модификатор \\0 для замены на найденное совпадение. Результат работы сценария представлен на рис. 33.1.

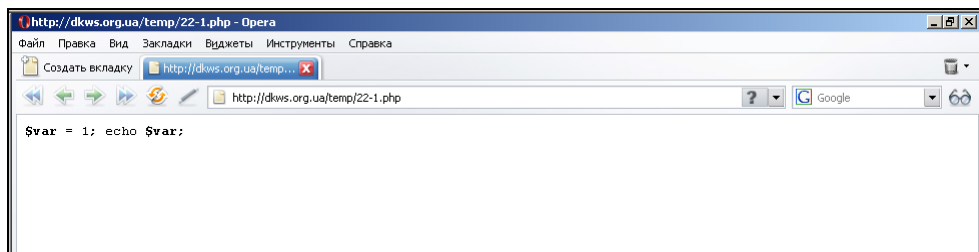


Рис. 33.1. Результат работы сценария из листинга 33.1

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 . 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.\-]*)`

Сценарий, выполняющий нашу задачу, представлен в листинге 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;

?>
```

Результат работы сценария представлен на рис. 33.2.

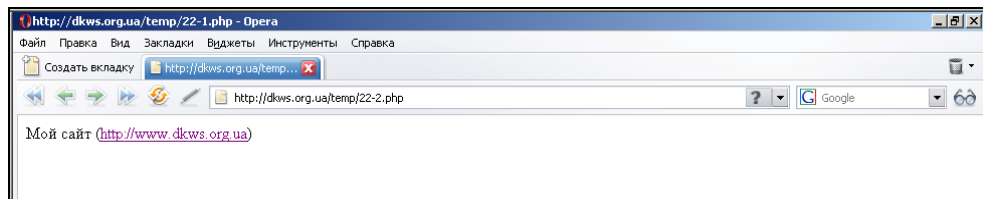


Рис. 33.2. Результат работы сценария из листинга 33.2

Кроме `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 регистр символов.

Рассмотрим несколько полезных примеров регулярных выражений. Регулярные выражения уже представлены в виде строк, которые можно использовать в PHP-функциях (с цитированием необходимых символов).

Следующее выражение является более корректным выражением для поисков URL-адресов (чем приведенное ранее):

```
((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 используется 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);
```



Работаем с MP3

34.1. Формат MP3

На сегодняшний день в Интернете существует очень много сайтов, предлагающих скачать музыку в формате MP3. Для каждой композиции выводится информация о ней: 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 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 встречается версия 1.1, которая отличается от 1.0 тем, что сокращена длина коммента-

рия — до 28 символов, но зато появилось новое поле, содержащее номер дорожки, то есть номер композиции в альбоме. На рис. 34.4 показан фрагмент MP3-файла (в шестнадцатеричном виде): видно служебное слово "TAG", после которого следует название композиции, имя исполнителя, название альбома, а остальная информация для этого MP3-файла не указана.

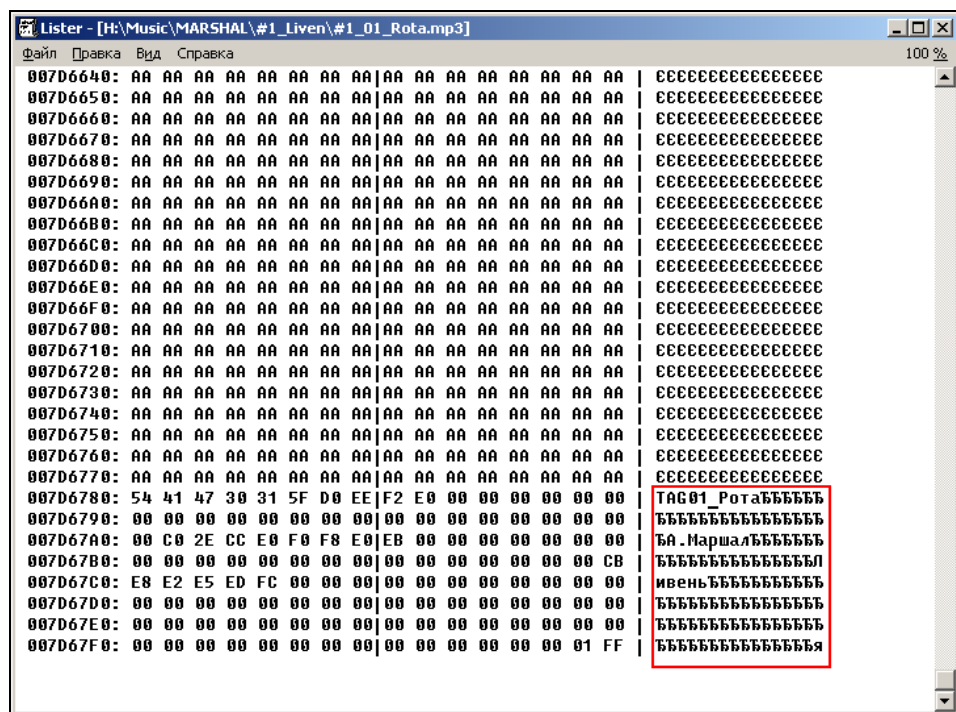


Рис. 34.1. MP3-файл (шестнадцатеричный вид)

34.2. Библиотека PEAR

Если честно, то не хочется "копаться" в MP3-файлах в поисках нужной нам информации. С технической стороны ничего сложного в этом нет, просто нет желания. Причем до такой степени, что можно "угovorить" 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 (на прилагаемом диске) вы найдете:

- ❑ MP3_Id-1.2.0RC3.tgz — последняя версия доступна по адресу http://pear.php.net/package/MP3_Id/download/;
- ❑ PEAR-1.5.0.tgz — последнюю версию можно скачать 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.

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 напишем 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();

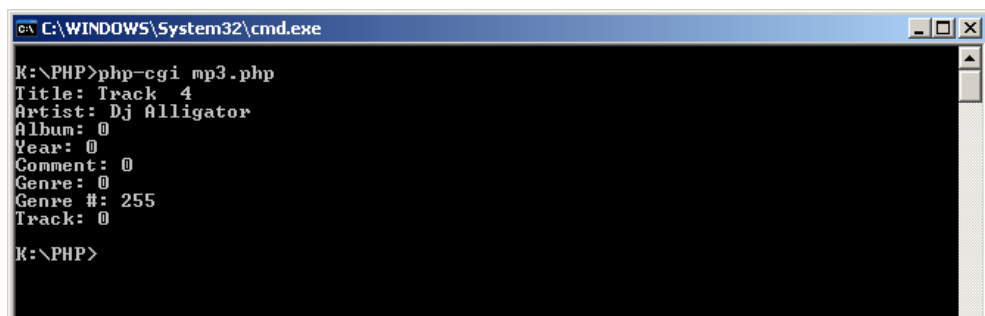
// читаем MP3-файл 1.mp3
$result = $id3->read('1.mp3');

// 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";  
?>
```

Результат работы этого 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 используется 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, требует изменения MP3-файла, поэтому убедитесь, что вы обладаете должными правами для выполнения этой операции.



Технология Ajax

35.1. Что такое Ajax

Аббревиатура Ajax расшифровывается как Asynchronous JavaScript and XML, что означает "асинхронный JavaScript и XML". Говорит это немного, поэтому иногда проще объяснить, что такое Ajax на примере, чем вникать в суть определения.

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>
```

Данная 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 обнов-

лена данными, полученными от сервера, в то время как пользователь все еще с ней работает. Взаимодействию Ajax и PHP посвящена не одна книга, мы же рассмотрим только основы этого взаимодействия. Если вы заинтересовались, вы без проблем найдете 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 `ajax.php` будет сгенерирован XML-код, который будет передан сценарию `ajax.js`. Сценарий `ajax.js`, получив XML-код, на его основании обновит HTML-страницу `index.html`.

Рассмотрим листинг `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 `Ajax.js`, которая будет выполнять асинхронный запрос;
- ☐ `UserName` — имя поля, содержащего ввод пользователя;
- ☐ `WelcomeMessage` — сюда будем выводить ответ от сервера.

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 будет получен ответ сервера, будет запущена функция `ServerResponse()`:

```
xml.onreadystatechange = ServerResponse;
```

Данная функция распаковывает полученный XML-объект, извлекает из него текстовое сообщение и обновляет поле `WelcomeMessage` нашей формы из `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>');
?>
```

Как видите, в Ajax нет ничего сложного. Конечно, наш сценарий ajax.js очень примитивен. Ознакомиться с другими Ajax-инструментами для PHP можно по следующему адресу:

<http://www.dkws.org.ua/phpbb2/viewtopic.php?p=14075#14075>.



РАЗДЕЛ 8

Разработка типичного сайта средней сложности



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: ни один из них не обладает нужными вам функциями. А может, вы боитесь, что стандартные движки стандартно и взламываются. Причина может быть и другая — 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 поддержание сайта стало занимать много времени. Нужно было автоматизировать процесс управления страницами. Понятно, что на HTML организовать автоматизацию невозможно. Нужен был PHP (или любой другой Web-ориентированный язык программирования, например Perl, но свой выбор я остановил на PHP). Исследовав предложения других хостинг-провайдеров, я пришел к выводу, что далеко не всегда бесплатный хостинг поддерживал 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, если честно, мне было лень создавать базу данных со своими страницами — ведь они все уже были готовы и хранились на диске в виде HTML-файлов.

Мне нужна была CMS, ориентированная на работу без базы данных. Были 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 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) можно найти в Интернете, поэтому вам придется постоянно следить за обновлениями движка: вдруг одно из них устраняет огромную "дыру" в системе безопасности вашего сайта. С другой стороны, ваш движок — темная лошадка для неквалифицированных злоумышленников. Рекомендаций по взлому, понятно, в Интернете не будет, а поэтому неквалифицированные злоумышленники, страдающие комплексом неполноценности, не смогут ничего сделать вашему сайту. Квалификация же разработчиков готовой 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 используется. 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 нет. А иногда даже в состав CMS входит сам форум, который часто не используется, поскольку в этом нет никакой необходимости.

Все неиспользуемые функции попросту занимают место на диске, 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?

В следующем разделе мы поговорим о функциях, которые мы собираемся реализовать в нашей CMS.

36.1.2. Необходимые нам функции движка

Успех любого дела зависит от хорошего плана. В данной 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. Хотя бы нужно организовать постраничный вывод из базы. Правда, и это не панацея. Если выводить по 10 ссылок на страницу, то всего будет 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 вашей базе данных будет информация не об одном процессоре, поэтому в категории "Процессоры" будут как минимум три подкатегории: "Процессоры Intel", "Процессоры AMD", "Процессоры прочие".

Под управлением страницами подразумевается добавление, редактирование и удаление страниц. Конечно, раз у нас будут подкатегории, то нужно обеспечить перемещение страницы из одной категории в другую. Все эти функции будут реализованы в панели управления сайтом.

Понятно, что у нашего сайта будет дизайн. Но дизайн — это штука не постоянная. Сегодня он один, а завтра вам захочется чего-то новенького. Наша задача — разделить PHP-код CMS и HTML-код дизайна сайта. Другими словами, чтобы при изменении дизайна вам не пришлось редактировать CMS. Лучший способ разделения PHP-кода и HTML-кода — это шаблоны. Мы не будем использовать уже готовые шаблонизаторы вроде Smarty — они сложны, тяжеловесны и не всегда надежно работают. Поэтому мы напишем свой простенький шаблонизатор, он 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** — 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 рассмотрели уже пример RSS-граббера новостей (см. главу 32), позволяющий 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 ваши фотографии, размещенные на LiveJournal, в дизайне сайта, что очень удобно. Тогда 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. В следующем параграфе мы рассмотрим схему нашей CMS.

36.1.3. "Принципиальная схема" движка

Итак, наша CMS состоит из следующих частей:

- ☐ база данных — содержит все страницы нашего сайта и прочую информацию;
- ☐ панель управления — управляет страницами нашего сайта, а именно: позволяет создавать новые страницы, редактировать и удалять уже имеющиеся;
- ☐ движок — управляет показом страниц пользователю, навигацией по сайту, реализует дополнительные функции CMS;
- ☐ шаблонизатор — движок отправляет информацию шаблонизатору, который выводит ее в соответствии с созданными дизайнером HTML-шаблонами.

На рис. 36.1 показана схема работы нашей CMS.

Как показано на схеме, двусторонний обмен данными может быть 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;



Рис. 36.1. Схема CMS

- 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.

У большинства уже готовых CMS модульная структура, то есть можно использовать модули для добавления в CMS новых возможностей. У нашей CMS модулей не будет по следующим причинам:

- поскольку CMS разрабатывается для себя, не думаю, что ею заинтересуются тысячи пользователей. Следовательно, даже если сделать возможность использования модулей, она останется невостребованной;
- модульная структура необходима для быстрого и простого добавления нового кода в CMS, то есть 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.

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 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. Шаблон `page.tpl` представлен в листинге 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 не стоит этого делать.

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';

?>
```

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`, который будет содержать идентификатор выводимой страницы. В итоге URL в строке браузера будет выглядеть так:

`http://сайт/index.php?p=тип&id=число`.

Если переменная `$p` не задана, то будет выведена главная страница сайта. В этом случае URL будет 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 `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 index.php, нужно скопировать файлы index.php, template.php, config.php и page.tpl в какой-нибудь каталог сервера и ввести в строке адреса браузера следующий URL:

<http://сервер/каталог/index.php>.

Наш `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}`.

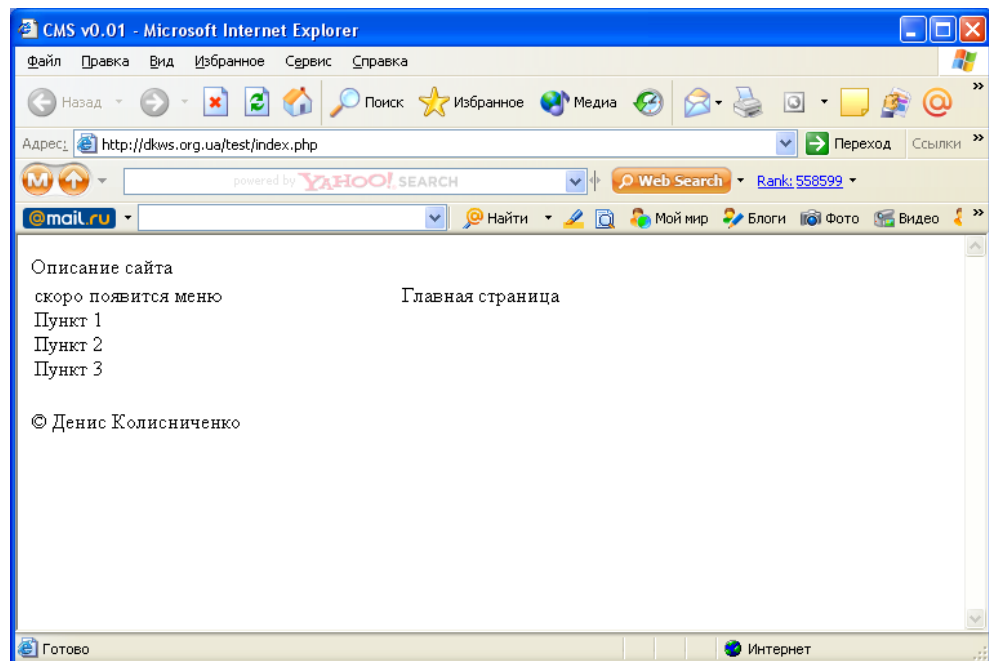


Рис. 36.2. Вывод сценария `index.php`

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. Таблица содержит всего два поля — id (идентификатор страницы) и content (содержимое страницы).

Чтобы создать данную таблицу, нужно выполнить следующий SQL-запрос:

```
CREATE TABLE 'static' (  
    'id' varchar(15) NOT NULL,  
    'content' text,  
    PRIMARY KEY ('id')  
) DEFAULT CHARSET=cp1251;
```

ПРИМЕЧАНИЕ

Данный SQL-запрос можно ввести в программе phpMyAdmin, описание которой вы найдете на прилагаемом диске. Основы SQL изложены в *главе 18*.

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 данной таблице соответствует одной странице сайта. В нашем случае будут три страницы — Главная страница, Прайс-листы и Контакты. Конечно, на практике содержимое данных 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 номер корневого раздела равен -1, то перед нами корневой раздел. Если же поле root содержит положительное значение, то перед нами — подраздел, а поле root указывает но-

мер его родительского раздела. Значение по умолчанию для этого поля `-1`, то есть если поле `root` не задано, мы подразумеваем, что будет создан корневой (родительский) раздел;

- ☐ `txt` — описание раздела, может содержать любую информацию.

Структура таблицы `cats` изображена 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 данной 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, то есть родительскими. Разделы 5–7 являются подразделами 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
☐	id	int(11)			Нет		auto_increment
☐	cat	int(11)			Нет		
☐	header	varchar(250)	cp1251_general_ci		Да	NULL	
☐	content	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,
  'price' varchar(250) default NULL,
  'rd' varchar(250) default NULL,
  PRIMARY KEY ('id')
) DEFAULT CHARSET=cp1251;
```

Для добавления новых страниц используется 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 `$r` 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

Пользователь 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 есть подразделы, то будет выведен список подразделов и информация о разделе. На рис. 36.7 информация о разделе не заполнена.

Обычно, если у раздела есть подразделы, то такой раздел не содержит страниц. Не нужно представлять разделы сайта, как каталоги файловой системы, а страницы — как файлы. Ведь 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 под-

раздела: "Процессоры AMD", "Процессоры Intel" и "Процессоры прочие". Следовательно, все страницы, описывающие процессоры разных производителей, будут помещены в соответствующий подраздел. Иного не дано. Поэтому если раздел содержит подразделы, то не нужно искать в базе данных страницы, которые относятся к данному разделу. Это особенность данной разработки. Конечно, вы можете немного изменить приведенный в книге пример и сделать так, чтобы разделы вашей CMS могли содержать и обычные страницы, и подразделы.



ПРОДУКЦИЯ

Дисковые пилы для стали
Дисковые пилы для алюминия
Ленточно-пилные станки
Линии обработки стальных профилей
Центры обработки листового металла
Штамповочно-рубочные линии



Контакты:

Разделы

Дисковые пилы для алюминия

TL 250
TL 350
TL 450
SKL 450 E
SKL 450 H
SKL 450 NA
RKL 551 AX

Страницы раздела

Название раздела

Полуавтоматические станки	Диапазон резки (мм)		Резка под углом
	круг	прямоугольник	
TL 250			да
TL 350			да
TL 450			да
SKL 450 E	160	145	да
SKL 450 H	160	145	да

Полностью автоматические станки	Диапазон резки (мм)		Резка под углом
	круг	прямоугольник	
SKL 450 NA	160	145	да

Высокопроизводительные станки	Диапазон резки (мм)		Резка под углом
	круг	прямоугольник	
RKL 551 AX	165	160	нет

Описание раздела

Рис. 36.6. Вывод содержимого раздела

Вы можете немного модифицировать код CMS (он будет представлен далее), чтобы разделы и страницы сайта были похожи на файлы и каталоги файловой системы.

Когда же пользователь откроет страницу, меню станет иерархическим (рис. 36.8), то есть будут отображены разделы и подразделы (если это нужно), а также

содержимое последнего активного раздела/подраздела. До этого отображать иерархию меню нет смысла, зачем выводить лишнюю информацию?

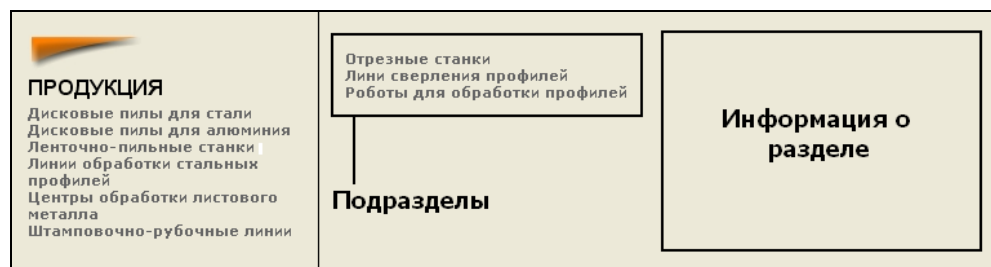


Рис. 36.7. Вывод раздела, содержащего подразделы

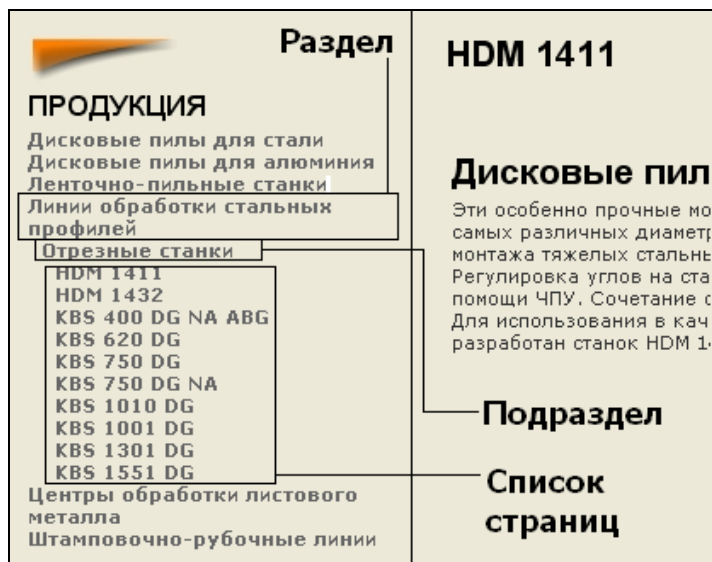


Рис. 36.8. Иерархическое меню сайта

Сценарий menu.php

Теперь, когда нам ясен алгоритм работы меню, напишем 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 будем использовать несколько служебных переменных:

- ❑ `$sub` — если данная переменная принимает значение 1, то выводимый раздел является подразделом;
- ❑ `$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 нужно 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)  
    $sqr = "select * from pages where id=$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 того как мы выполнили подготовительные действия, мы можем с чистой совестью вывести корневые (родительские) разделы.

Это можно сделать 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
```

Данный 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, то есть 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 = $menu .  
                "<br>&nbsp;&nbsp;&nbsp;<a  
href=index.php?p=show&id=$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);  
  
// выводим список страниц  
while ($row = mysql_fetch_array($res))  
    $menu = $menu .  
        "<br>&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&  
        <a  
        href=index.php?p=showpage&id=$row[id]>$row[header] </a>";  
  
}  
} // if ($is_ext_menu == 1)  
  
}
```

В листинге 36.4 представлен полный текст файла `menu.php`. Чтобы листинг был менее громоздким, я удалил все комментарии, которые были приведены ранее.

Листинг 36.4. Файл menu.php

```
<?

$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 = $menu . "<br>&nbsp;&nbsp;&nbsp;<a
href=index.php?p=show&id=$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 .
```


Замените его оператором `include`:

```
include "menu.php";
```

Запустите `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}`.

Вывод содержимого раздела и страницы

Наш `index.php` должен реагировать на значения `'show'` и `'showpage'` переменной `$p`. Сначала реализуем вывод содержимого раздела. Откройте `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, то есть 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 = $page .  
                "<br><b>  
                <a href=index.php?p=showpage&page=$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[id]>
                    $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 документа. Нужный нам код представлен в листинге 36.6. Данный код нужно добавить сразу после последней закрывающей фигурной скобки сценария index.php.

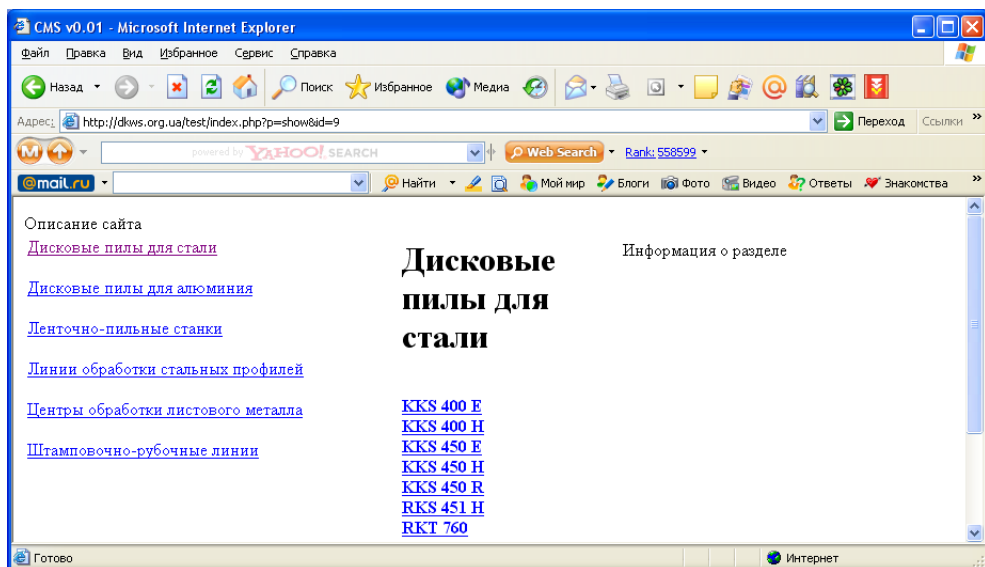


Рис. 36.10. Вывод раздела сайта

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><br><a href=#down>Вниз</a></center><p>";

        // выводим содержимое страницы
        $page = $page . "<p><br><br>$row[content]";

        // выводим ссылку на версию для печати
        $page = $page . "<p><p><a target=_blank href=print.php?page=$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);
```

Последняя строка 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.

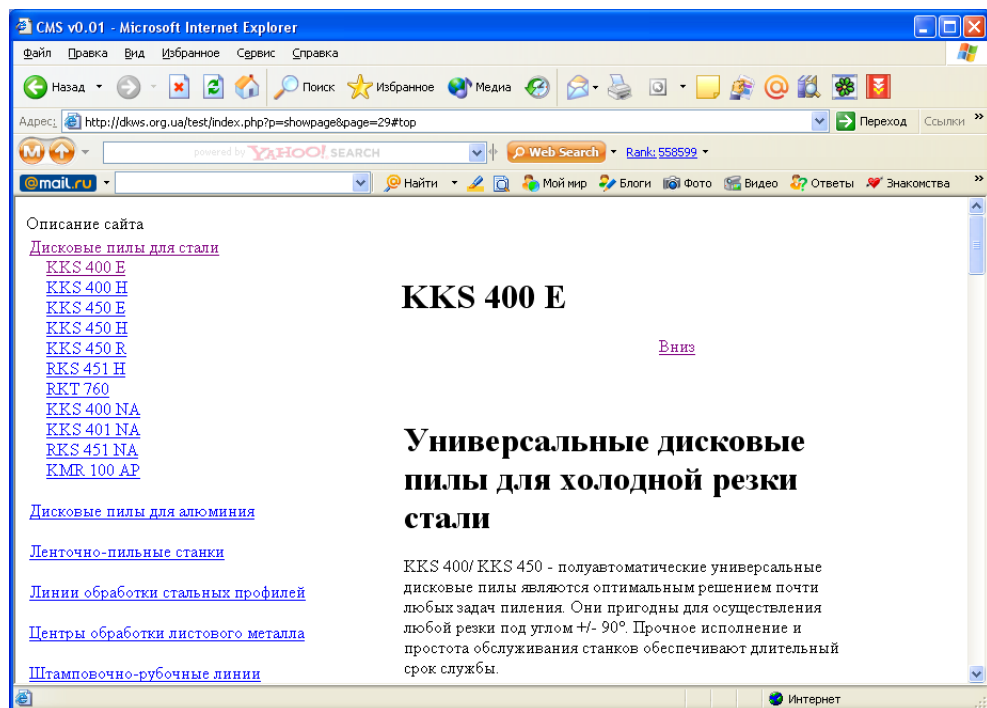


Рис. 36.11. Меню готово

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}`. В каталоге `Glava_36` вы найдете измененную версию файла `page.tpl`.

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'`, 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` (см. разд. 36.4). Так как на протяжении этой главы мы будем изменять данный файл, наращивая возможности CMS, то версия с поддержкой таблицы `static` будет называться `index-static.php`.

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 данной функции открыть произвольный файл в системе.

Как злоумышленник может открыть любой файл? Сейчас поясню. Ваш сценарий `index.php` находится на жестком диске Web-сервера. Пусть данный файл находится в каталоге `/var/www/client01/html`. При неправильной настройке Web-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 Web-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` не будем, а просто напомним файл `print.php`. Данному 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

Понимаю, что создавать все сценарии этой главы вам лень, поэтому я решил облегчить вам жизнь. Все листинги этой главы доступны на прилагаемом диске в каталоге Glava_36.

В каталоге Glava_36 вы найдете следующие файлы:

- ☐ 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.



Создание фотогалереи

37.1. Постановка задачи

В этой главе мы разработаем совсем простую галерею изображений. Мы не будем использовать аутентификацию пользователей, поэтому добавить изображение в нашу галерею сможет любой пользователь. Понятно, мы установим ограничение на размер файла и размеры фотографии. При загрузке файла в галерею производится создание его миниатюры.

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

Первым делом напишем сценарий, управляющий загрузкой изображения на сервер. Ведь перед просмотром галереи нужно загрузить 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>
```

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. Мы будем использовать приведенный в ней код. Наша форма предусматривает загрузку только одного файла, но вы можете модернизировать 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 мы будем с помощью функции `imageresize()`, которую мы разработали в *главе 15*. Если нужно, вы сможете получить дополнительную информацию о загруженных изображениях с помощью функции `exif_read_data()`. Пример использования этой функции был представлен 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 160x100, качество JPEG 75
imageresize($f_thum, "gallery/$myfile_name", 160, 100, 75);

}

}

else
{
    // выводим форму
    echo file_get_contents('gallery_form.html');
}

?>
```

Проанализируем код данного сценария. Сначала мы проверяем, был ли передан файл. Если нет, то мы выводим форму (файла 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, которую можно получить с помощью функции `exif_read_data()`.

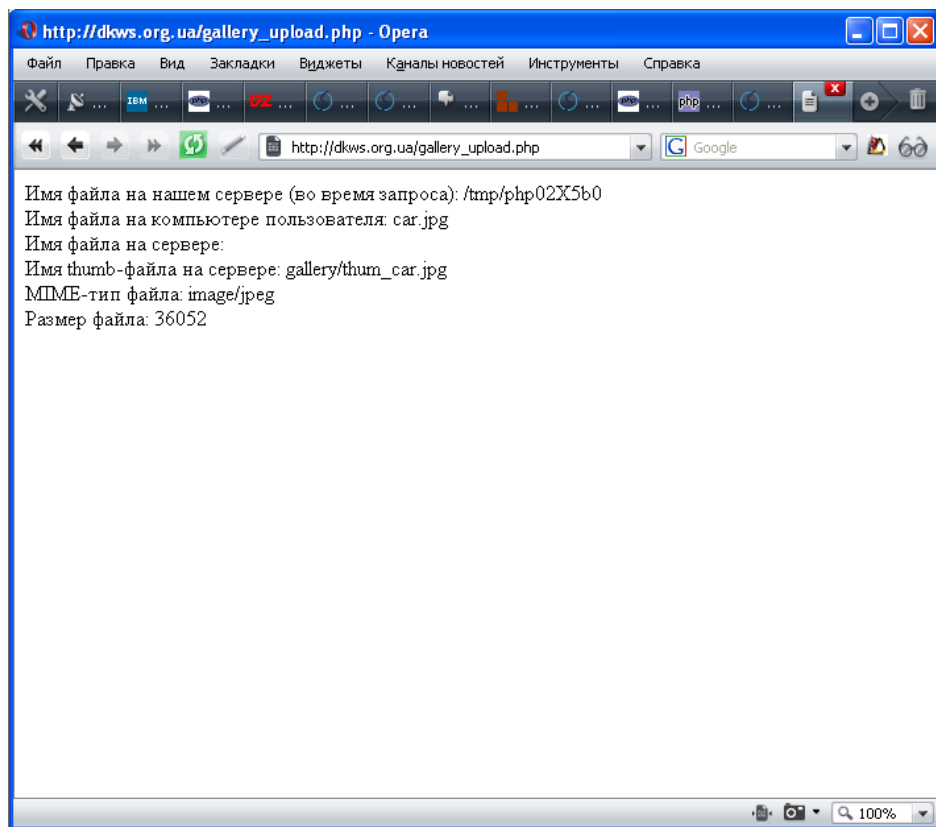


Рис. 37.1. Характеристики загруженного файла

Потом, если размер файла превышает 512 Кбайт, то мы ничего копировать в галерею не будем, 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 лимит, тогда мы копируем файл из временного каталога в каталог `gallery`:

```
copy($myfile, "gallery/$myfile_name");
```

Понятно, что нужно создать этот каталог и установить права доступа 777:

```
mkdir gallery
chmod 777 gallery
```

Можно попытаться создать этот каталог в RНР-сценарии, но я не совсем уверен, что у вас хватит прав для выполнения этого оператора (напомню, что интерпретатор RНР выполняется от бесправного пользователя):

```
@mkdir("gallery", 0777);
```

Вернемся к копированию загруженного файла. Мы используем переменную `$myfile_name`, содержащую имя файла на компьютере пользователя. Если пользователь загружает файл `car.jpg`, то и на сервере он будет называться `car.jpg`. У этого метода есть 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 так же будет называться на сервере, а это тоже нежелательно, поскольку сервер обычно работает под управлением UNIX и, учитывая, что Windows и UNIX используют разные кодировки, могут быть проблемы с именем файла.

Поэтому вам нужно придумать систему создания уникальных имен файлов, которая будет проверять, содержит ли имя файла русские буквы, а также будет проверять существование файла на сервере. 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 имя, только нужно удалить 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 вот так:

```
$f_thum="gallery/thum_" . $myfile_name;
```

то есть префикс `thum_` дописывается к имени файла. 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 качество. Чтобы миниатюры создавались более качественно, вам нужно с помощью функции `exif_read_data()` узнать высоту и ширину картинки. Обычно у картинки горизонтальная (альбомная) ориентация, то есть ширина больше, чем высота. Но если пользователь будет загружать картинку вертикальной (портретной) ориентации, когда высота больше, чем ширина, то вам нужно 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 нужно реализовать систему одобрения загружаемых картинок. Для этого создается MySQL-таблица, в которую вносятся все имена файлов из каталога `gallery`. Для каждого файла в таблице будет отметка (поле `APR`): одобрен он или нет. 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: вывести 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.

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');

?>
```

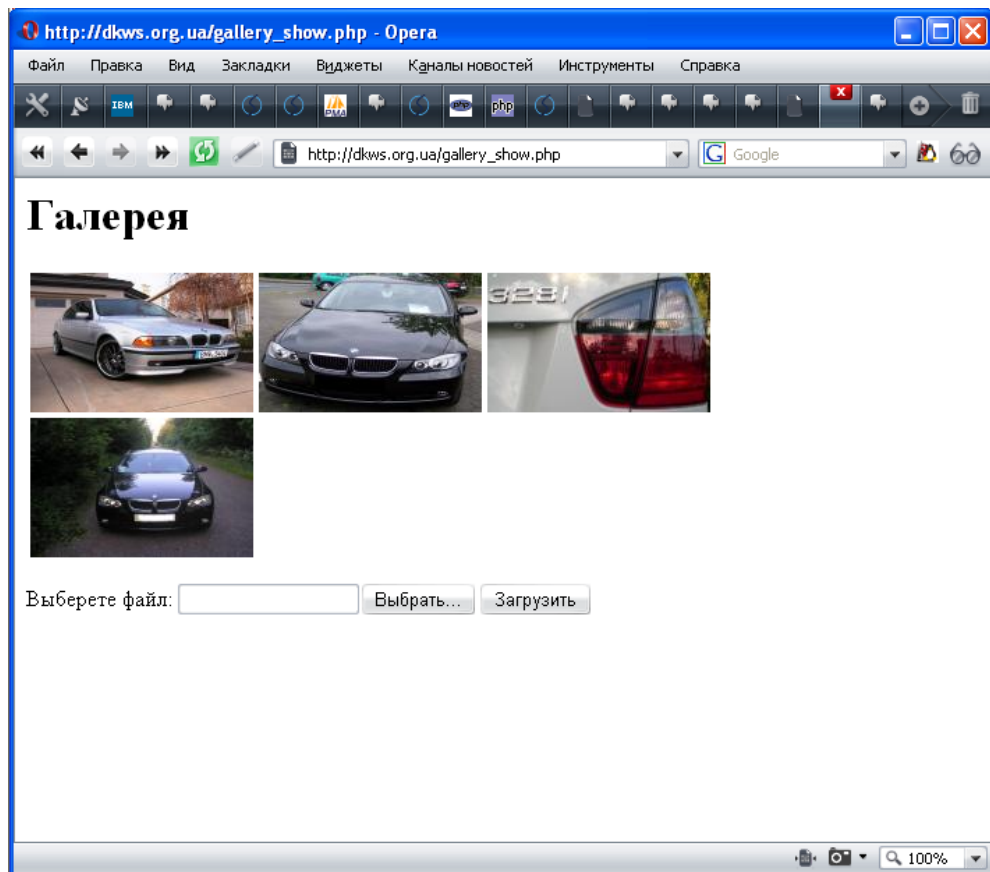


Рис. 37.2. Созданная нами галерея

Результат работы данного сценария приведен на рис. 37.2. Конечно, нет предела совершенству. В наш сценарий можно добавить, как минимум, постраничный вывод галереи, но в книге я не стал этого делать, чтобы не усложнять код сценария вывода галереи. Принцип постраничного вывода содержимого приведен в *главе 21*.

Полный исходный код галереи вы можете найти в каталоге Glava_37 прилагаемого диска.

Для подключения галереи к нашему движку нужно в определенном (по вашему усмотрению) месте кода 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 совсем не 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 будем выводить не все сообщения, а только проверенные администратором. Это вполне логично: ведь как только спаммеры узнают о вашей гостевой книге, она сразу же будет завалена спамом. При получении нового сообщения администратору будет отправлено сообщение с просьбой подтвердить сообщение. Если сообщение корректное, администратор может подтвердить его размещение в базе данных. В противном случае администратор может удалить это сообщение (в e-mail, отправленном администратору, будет соответствующая ссылка). Как видите, наша гостевая книга даже не будет нуждаться в панели управления.

Во-вторых, перед помещением в базу данных из сообщения будет удален весь вредоносный код, который, возможно, поместили в сообщение спаммеры.

В-третьих, мы установим переменную сессии сроком в одну неделю, что не позволит одному и тому же пользователю отправлять повторное сообщение (это оградит нас от спаммеров).

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.

38.3. Вывод гостевой книги

Наш сценарий вывода гостевой книги будет очень похож на сценарий страничного вывода таблицы, созданный нами в *главе 21*, но у него будут два отличия: он будет выводить только подтвержденные сообщения, для которых поле `APR` = 1. Подробно этот сценарий рассматривать мы не будем. Все, что изменилось, — это названия полей и самого сценария, также добавились 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 ob, то есть 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, то есть 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 tyle=text name=uname>  
<p><textarea cols=50 rows=10 name=t></textarea>  
<p><input type=submit value=Добавить>';  
  
?>
```

Сценарий gb_show.php в работе изображен на рис. 38.1.

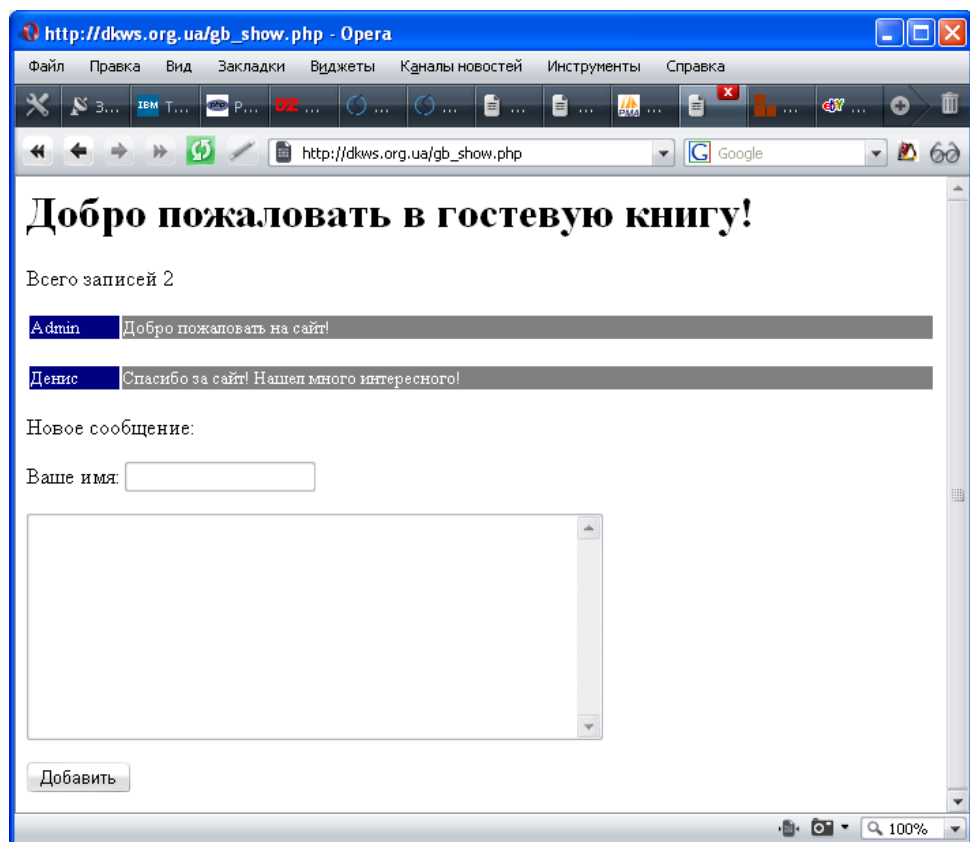


Рис. 38.1. Наша гостевая книга

38.4. Добавление записей в гостевую книгу

Теперь приступим к реализации сценария `gb_add.php`. Первым делом 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 его указать, будет использовано имя Гость. А вот сообщение указывать обязательно. Если сообщение 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 мы формируем запрос на добавление новой записи в нашу таблицу. После запроса мы получаем идентификатор последней вставленной записи с полем типа AUTO_INCREMENT. Его мы будем использовать 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();
```

Затем мы выводим сообщение пользователю о том, что его сообщение добавлено (рис. 38.2). Администратору мы отправляем сообщение, добавленное пользователем, 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);
```

```
?>
```

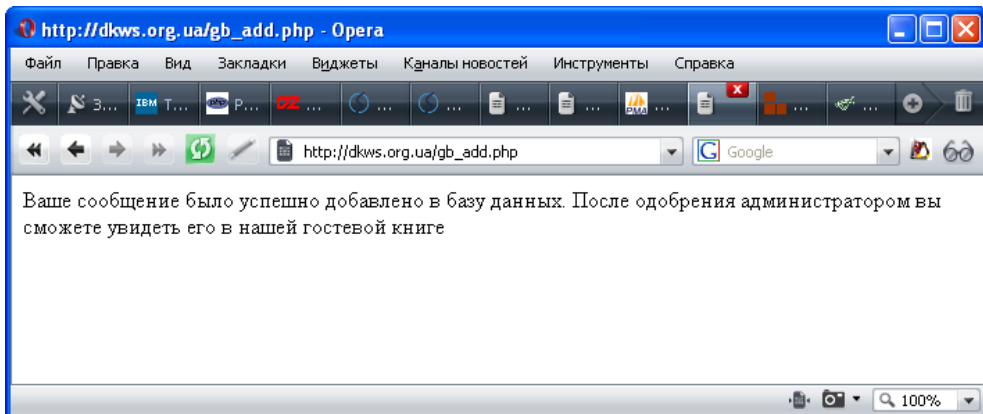


Рис. 38.2. Сообщение, которое видит пользователь после добавления новой записи

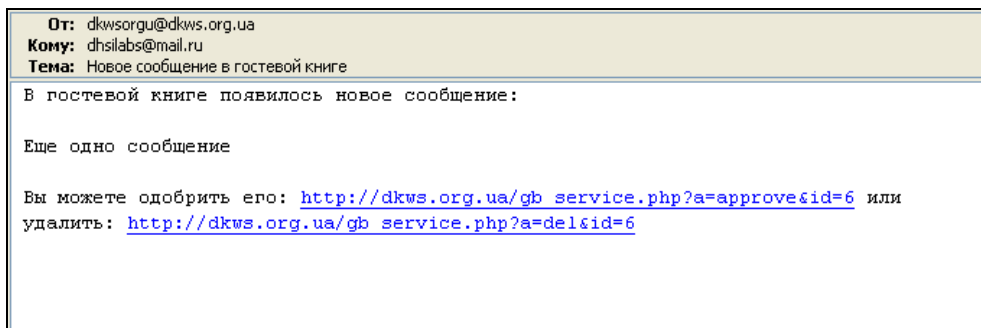


Рис. 38.3. Сервисное сообщение администратору

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 одо-
бления 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 (`approve` — одобрить, `del` — удалить) над сообщением с номером `id`. Затем, в зависимости от значения параметра `a`, 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";  
  
}  
?>
```

Данный 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 будет лень каждый раз вводить пароль администратора, можно для аутентификации пользователя использовать сессии. Вы можете установить длительность сессии, скажем, в 15 минут, а 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, вы без проблем реализуете защиту гостевой книги от злоумышленника. Все файлы гостевой книги вы можете найти в каталоге Glava_38 прилагаемого диска.



Интеграция галереи LiveJournal и вашего сайта

39.1. Что такое Живой журнал

В *главе 37* мы рассмотрели создание галереи вручную.

"Живой" журнал (LiveJournal) — простое и бесплатное средство для общения в Интернете. Он предоставляет вам следующие сервисы: частный дневник (его никто не увидит, кроме вас), блог (открытый сетевой дневник, который может прочитать кто угодно) и фотогалерею.

Для регистрации в LiveJournal нужно зайти на сайт <http://www.livejournal.com/> и создать бесплатную учетную запись (рис. 39.1).

На все про все (дневники и фотографии) вам отводится 1 Гбайт (!) дискового пространства. И это все бесплатно.

Да, согласен, что если вы хотите создать серьезный проект, тогда возможностей Живого журнала будет маловато. Но, с другой стороны, он позволяет вам хранить свои фотоснимки. Идея предельно проста: все данные, кроме фотографий, вы размещаете на обычном хостинге, а фотографии, которые занимают много места, вы помещаете на LiveJournal.

Узнать практически все о Живом журнале можно, прочитав список вопросов и ответов по адресу:

<http://www.livejournal.com/support/faq.bml>.

В частности, в 68-м вопросе рассматривается, как можно встроить LiveJournal в сайт:

<http://www.livejournal.com/support/faqbrowse.bml?faqid=68>.

Более подробно несколько способов такой интеграции рассмотрены здесь:

http://community.livejournal.com/howto_in_ru/1507.html.

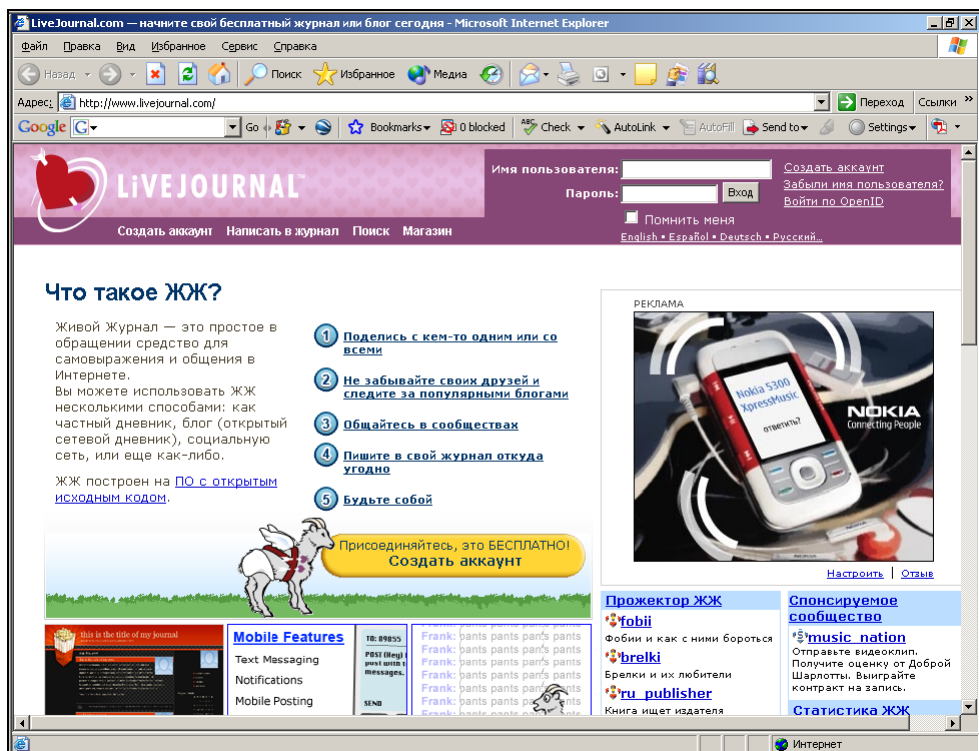


Рис. 39.1. Создание учетной записи на <http://www.livejournal.com/>

Но у всех этих способов два недостатка:

1. Интеграция осуществляется с помощью фреймов, а их использовать, мягко говоря, нежелательно. Я рекомендую отказаться от фреймов, поскольку они являются пережитком прошлого, да и в дизайн современного сайта они не вписываются.
2. Ничего не сказано об интеграции фотогалереи, а она интересует нас в первую очередь. Если бы нам хотелось сделать блог, то мы могли бы создать его собственными силами, не прибегая к помощи LiveJournal (или хотя бы просто установить на свой сайт уже готовый сценарий блога, каких в Сети очень много).

39.2. Интеграция фотогалереи LiveJournal и сайта

После того как вы создадите свой аккаунт, редактировать фотогалереи можно с помощью команды меню LiveJournal **Фотографии** | **Редактировать галерею** (рис. 39.2).

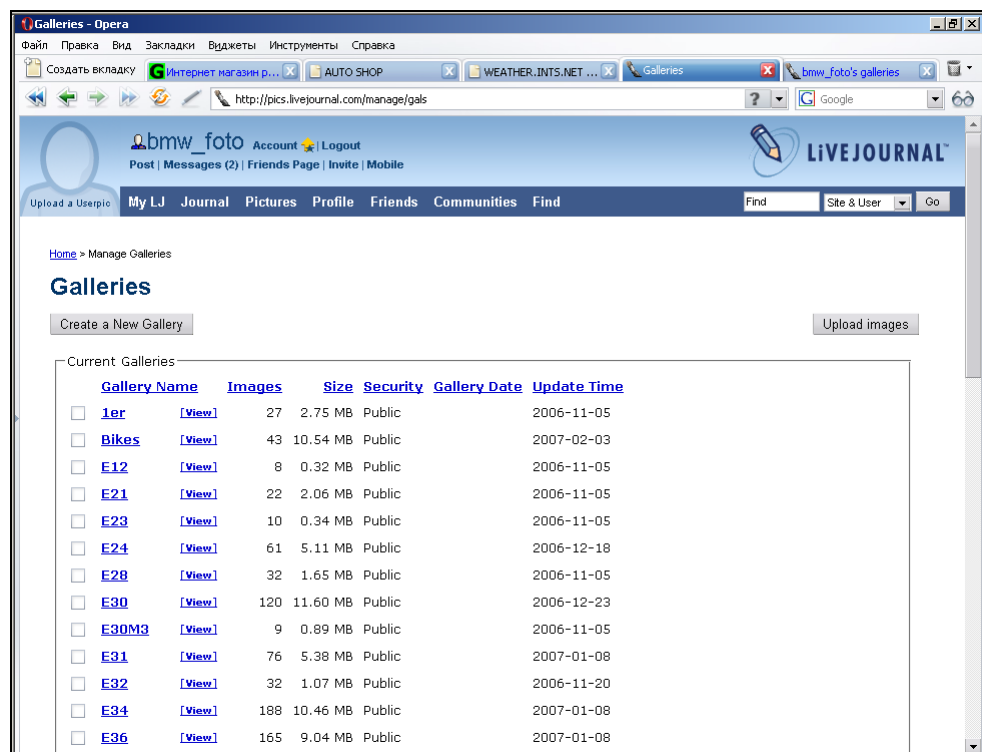


Рис. 39.2. Редактирование галерей

Просмотреть загруженные фотографии вы и другие пользователи могут по адресу:

http://pics.livejournal.com/<ваш_логин>/.

На рис. 39.3 представлена моя галерея (как она выглядела до интеграции в сайт).

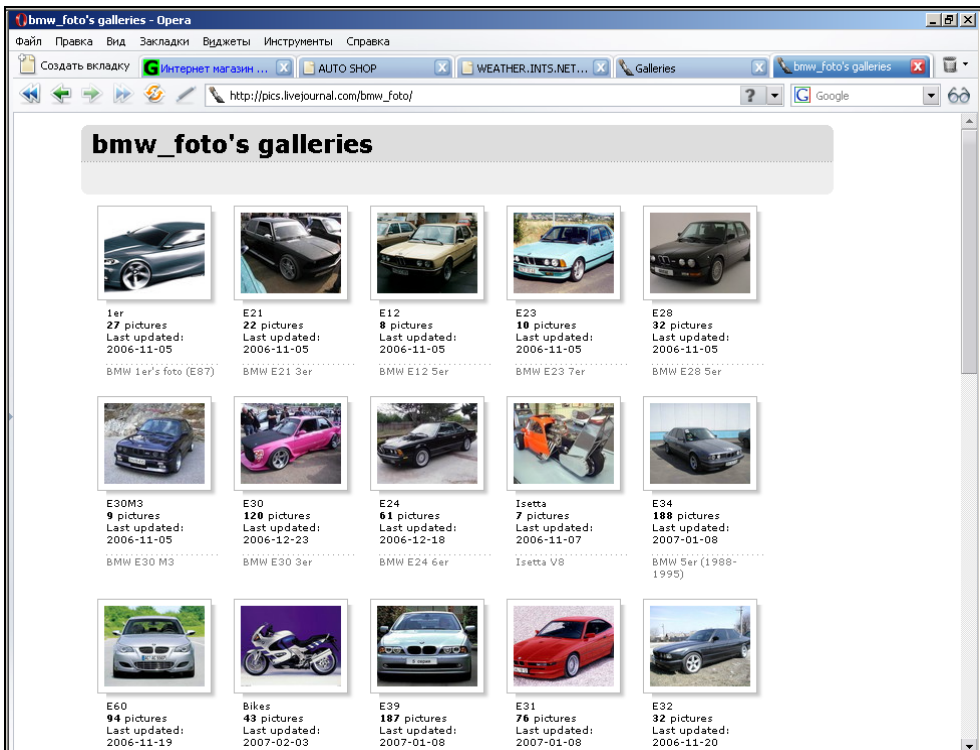


Рис. 39.3. Фотогалерея до интеграции в сайт

Как интегрировать фотогалерею в сайт? Первое, что приходит в голову, — это запросить по HTTP адрес http://pics.livejournal.com/<ваш_логин>/ и вывести его в дизайне сайта. Как говорят психологи, первая мысль — самая удачная. Так и сделаем. Предположим, что у нас есть основной сценарий сайта `index.php`, который выводит нужные страницы в зависимости от значения переменной `$page`. Договоримся, что если `$page` содержит значение 'gallery', то будет выведена наша галерея (листинг 39.1).

Листинг 39.1. Пример вывода фотогалереи

```
if ($page=="gallery")
{
// переменная $text теперь содержит HTML-код галереи
// нам остается вывести переменную $text в нужном месте,
// чтобы не нарушить дизайн сайта
```

```
$text = join('',file('http://pics.livejournal.com/<ваш_логин>/'));  
}  
  
...  
echo $text;
```

Увидеть галерею в дизайне нашего сайта можно по адресу:

`http://<имя_сервера>/index.php?page=galery.`

Все бы хорошо, но при первом же просмотре фотографии происходит переход с нашего сайта на сайт LiveJournal. Конечно, нам хочется, чтобы фотографии открывались в окне нашего сайта. Поэтому нам необходимо заменить все ссылки, содержащиеся в коде галереи, на ссылки вида:

`http://<наш_сервер>/index.php?page=view_url&u=<исходный URL фото>.`

Тогда при открытии фотографии ее адрес будет передан нашему сценарию `index.php`, который отобразит фотографию как нужно. В листинге 39.2 представлен фрагмент файла `index.php`, который обеспечивает вывод галереи и отдельных фотографий.

Листинг 39.2. Вывод фотогалереи в дизайне нашего сайта

```
if ($page=="galery")  
{  
    // переменная $text теперь содержит HTML-код галереи  
    // нам остается вывести переменную $text в нужном месте,  
    // чтобы не нарушить дизайн сайта  
    $text = join('',file('http://pics.livejournal.com/<ваш_логин>/'));  
  
    $text = str_replace('href="http://',  
        'href="http://<сервер>/index.php?page=view_url&u=http://', $text);  
}  
  
if ($page=="view_url")  
{
```

```
// получаем переданный URL
$text = join(' ',file($u));

// опять производим замену URL
$text = str_replace('href="http://',
    'href="http://<сервер>/index.php?page=view_url&u=http://', $text);
}

...

// выводим переменную $text
echo $text;
```

Окончательный результат представлен на рис. 39.4 и 39.5.

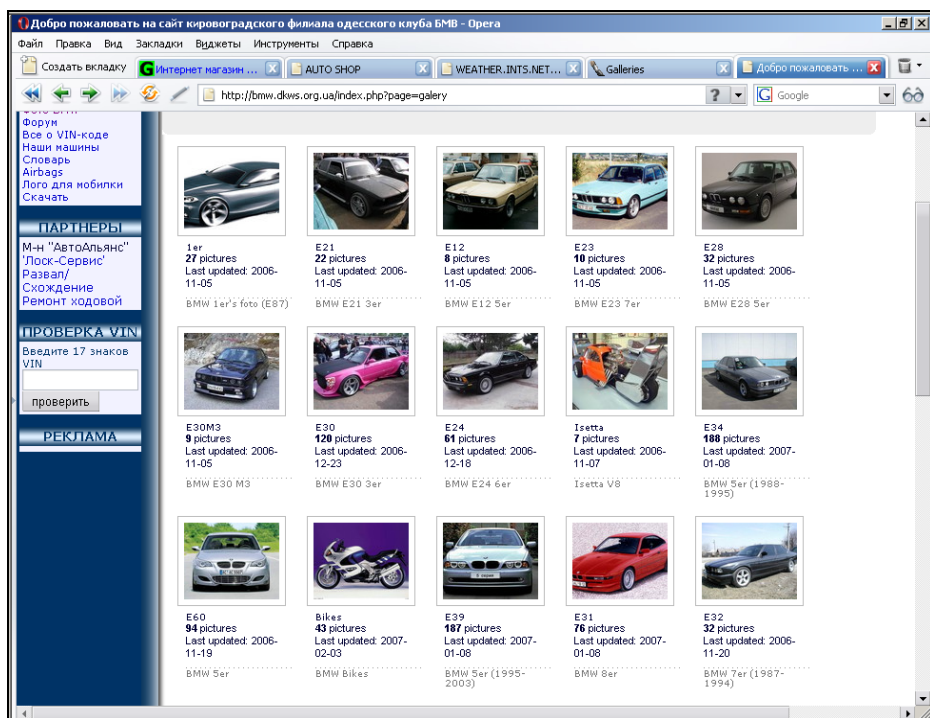


Рис. 39.4. Вывод галереи в дизайне сайта

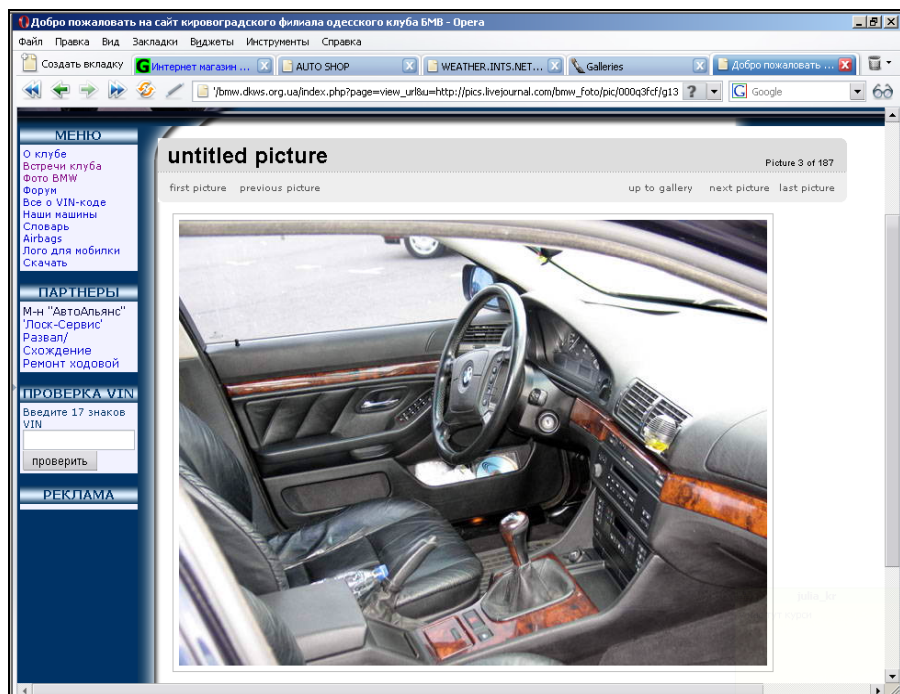


Рис. 39.5. Вывод отдельной фотографии

39.3. Настройка внешнего вида галереи

Путем изменения полученного HTML-кода можно придать фотогалерее вид, который максимально соответствует дизайну нашего сайта, но зачем усложнять наш сценарий, если можно настроить сам Живой журнал, чтобы он выводил галерею так, как нам хочется? Вот этим сейчас и займемся. В меню LiveJournal выберите команду **Pictures | Edit Style** (Фотографии | Настройка стиля). После этого в группе **Current Style** следует перейти по ссылке **Edit**. В результате вы увидите страничку, изображенную на рис. 39.6.

При базовом редактировании стиля вы можете выбрать только расположение галереи и тему оформления. Понятно, что хотелось бы большего (рис. 39.7), поэтому нажимаем расположенную чуть ниже кнопку **Edit**, после чего вы сможете указать число фотографий на одной странице, максимальное число фотоснимков в одной галерее, размер эскизов (уменьшенных изображений), а также задать альтернативный файл стиля (CSS).

После этого с отображением фотогалереи все будет в порядке.

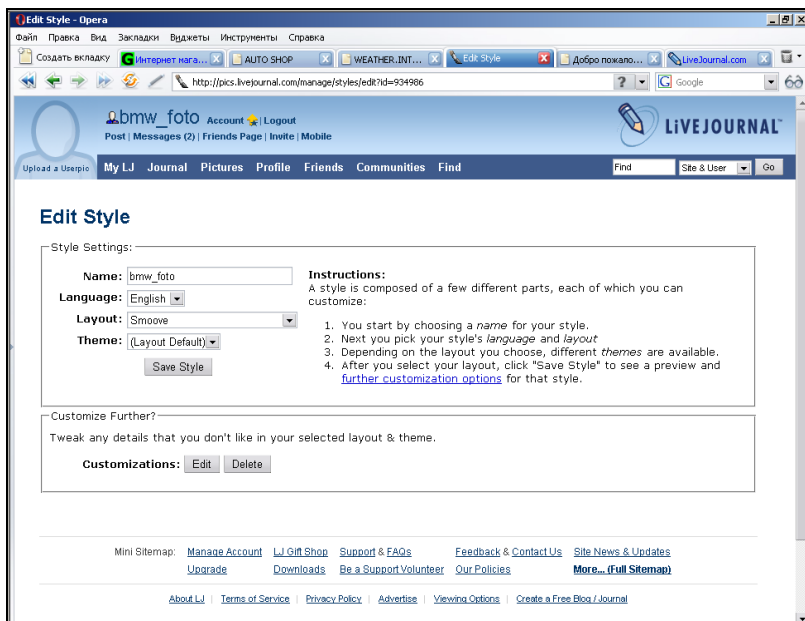


Рис. 39.6. Базовое редактирование стиля галереи

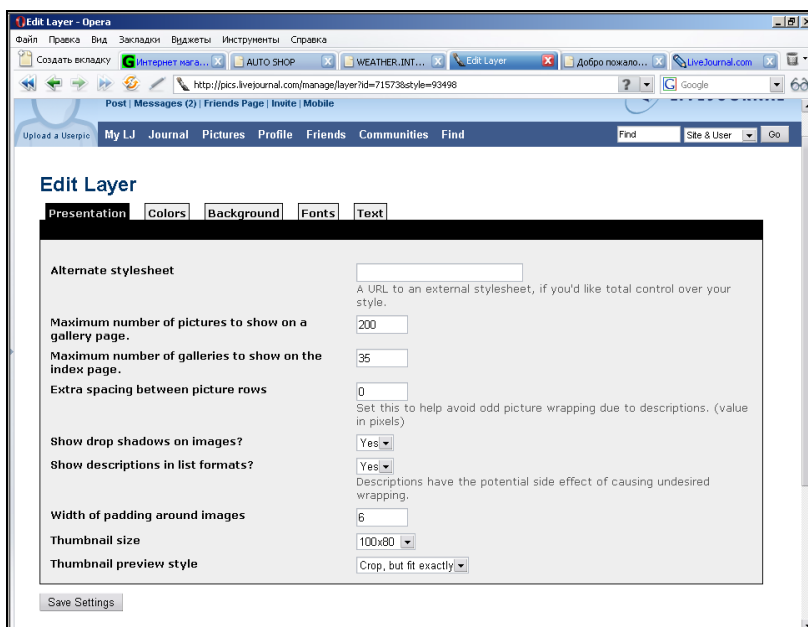


Рис. 39.7. Расширенное редактирование стиля галереи



Создание счетчика сайта

40.1. Постановка задачи

В этой главе мы разработаем простейший классический счетчик посещений, выводящий два числа — количество посещений за сутки и общее количество посещений. Данный счетчик будет только подсчитывать посещения, но не будет 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.

Работать наш счетчик будет очень просто. Значения счетчика будут храниться 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 (то есть на протяжении последних суток он еще не заходил на сайт), то мы увеличиваем оба значения и выводим их в браузер в виде картинки.

Как счетчик будет знать, что посетитель уникален? У нас будет еще таблица IP-адресов, в которую будут вноситься IP-адреса всех пользователей, посетивших наш сайт сегодня. Если IP-адрес есть в данной 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 наш счетчик.

40.2. Файл конфигурации

Наш счетчик будет использовать файл конфигурации `config.php`, разработанный в *главе 36*. Чтобы вы лишней раз не листали книгу, напомним вам его (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`. Данные таблицы 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)  
);
```

Данные таблицы довольно примитивны, и по большому счету для нашей цели хватило бы два файла, но использование MySQL значительно упрощает сценарий, поскольку нам не нужно беспокоиться о правах доступа к файлу.

Если вам лень 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 для начала нужно продумать его логику. Первым делом сценарий должен получить текущие значения счетчика и список IP-адресов. Затем сценарий должен 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 нужно поместить в таблицу counter. Затем нам останется лишь вывести эти значения как картинку.

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 опреде-

ляем 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` как картинку:

```
// 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);
```

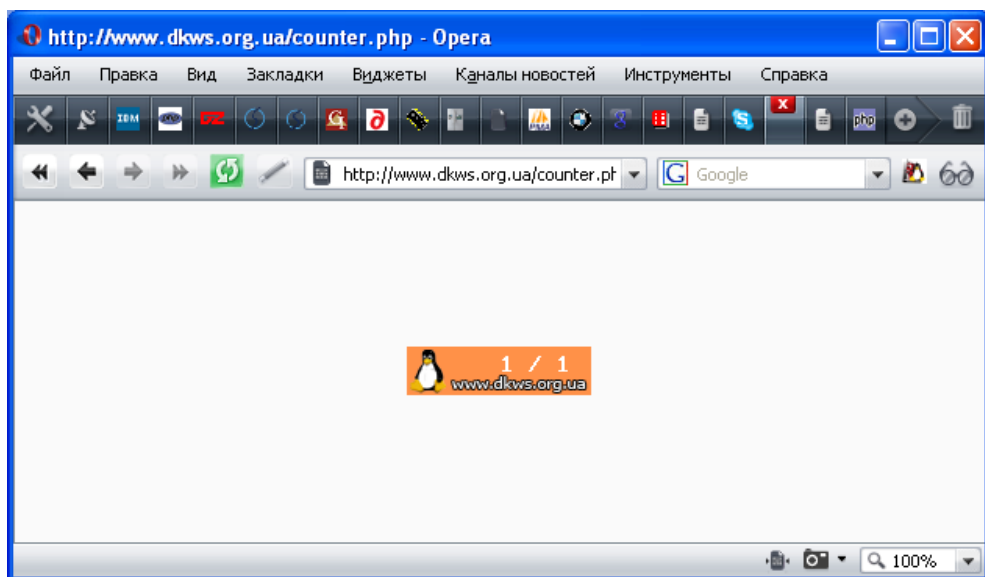


Рис. 40.1. Счетчик в работе

Мы не будем 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. Когда будете работать 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.

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. После этого запустите наш счетчик. Его показания будут — 1/2, то есть 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 вашего провайдера.



Статистика сайта

41.1. Методы сбора статистики

Понятно, что вам, как любому Web-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 вашего сайта и предлагать более интересную аудиторию 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 Web-сервер записывает каждое посещение в свой журнал. В журнал записываются дата и время посещения, IP-адрес пользователя, его агент (браузер), а также запрошенный пользователем файл. Вот пример одной записи из файла журнала Web-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)"
```

Данная 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 пользовате-

ля — 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 у нас есть компьютерный магазин. Не интернет-магазин, а самый обычный магазин. У вас есть список постоянных клиентов, которые несколько раз в месяц обязательно делают крупные покупки в вашем магазине. 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 10 бумаги, а Сидорову — пара сотен DVD-дисков. Также у вас есть 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, потеряете свой бизнес.

Размещая счетчик посторонней системы статистики на вашем сайте, вы бесплатно отдаете ей всю информацию о пользователях вашего сайта, об их интересах и т. д. Если у вас сугубо информационный сайт, например, как у меня, то это еще полбеды. А вот если у вас коммерческий сайт (например, интернет-магазин), то это равноценно тому, что вы дарите конкурентам своих покупателей.

41.2. Программы-анализаторы журналов Web-сервера

Лучшей программой-анализатором журналов является Webalizer (рис. 41.1). Довольно часто данная программа устанавливается по умолчанию на хостинг, 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, использующих 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. Программа 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/**.

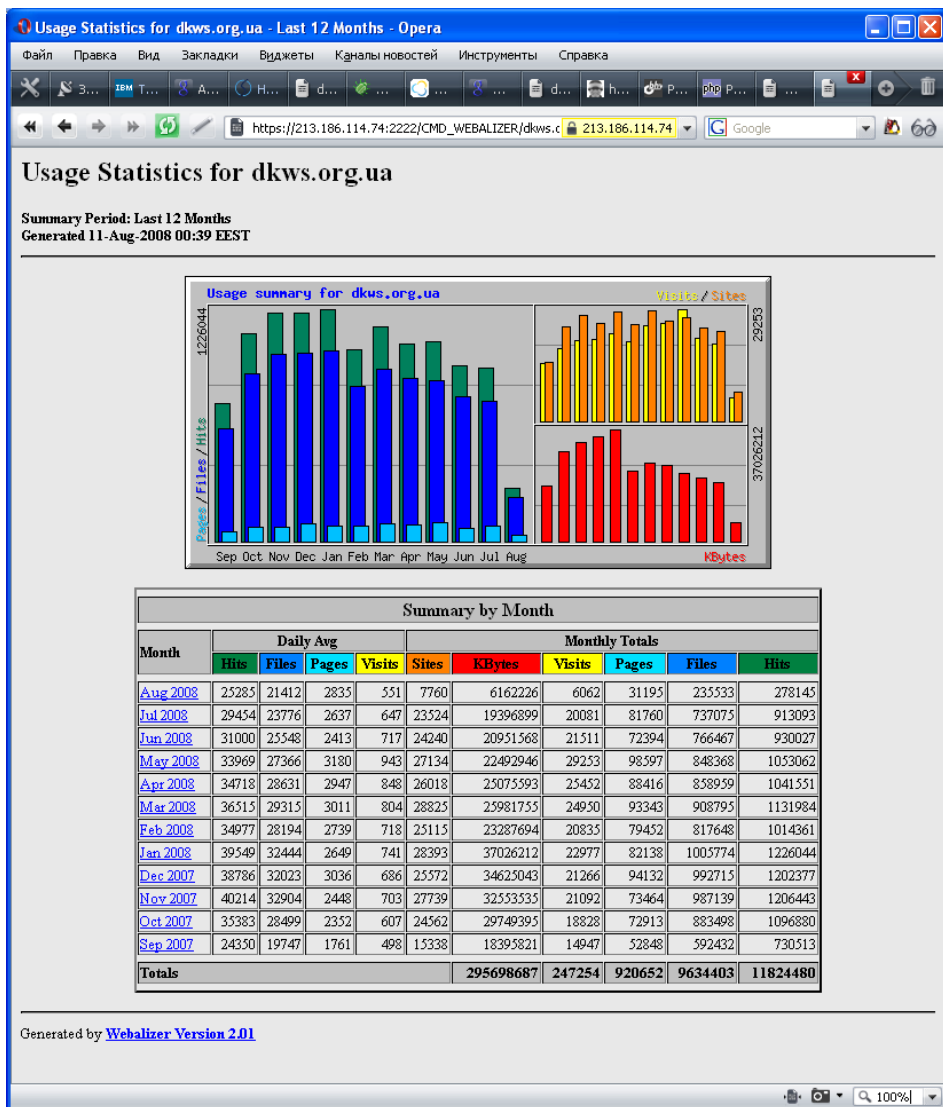


Рис. 41.1. Программа Webalizer

41.3. Системы статистики

Изо всех систем статистики, которыми мне приходилось пользоваться, я выбрал две: Google Analytics и bigmir.net. Первая система предоставляет огром-

ное количество различных отчетов, которые, может, и не всегда нужны. Но для расширенного анализа посещаемости сайта — самое оно (рис. 41.2)!

Для регистрации в Google Analytics вам нужно посетить сайт этой системы:

<https://www.google.com/analytics/>.

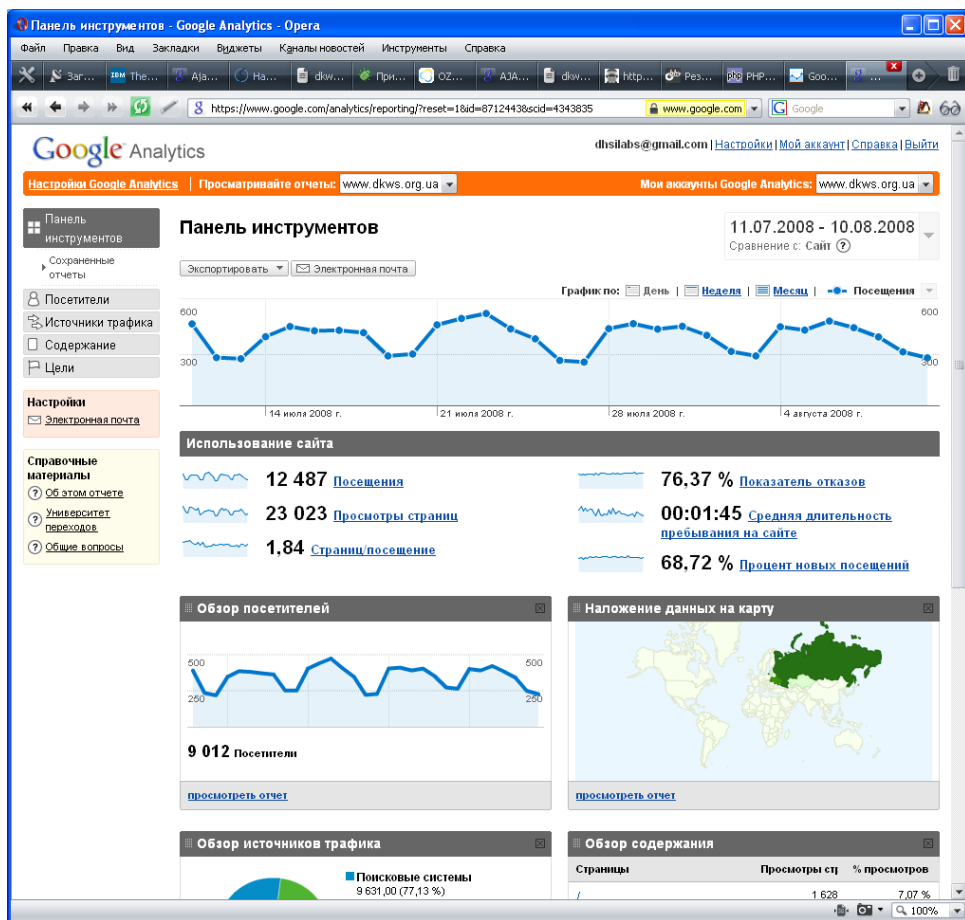


Рис. 41.2. Статистика сайта dkws.org.ua

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.

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

bigmir.net Рейтинг сайтов

Поиск Украина мир файлы товары

Найти

Начало ТОП-25 Статистика Добавить Изменить Удалить Счетчики Настройки Правила Помощь

Статистика сайта DKWS: настройка Linux

Общая статистика сайта

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

Отчеты

Для более полных данных - перейдите на новые отчеты

- общая статистика сайта
- посещаемость
- посетители
- страницы
- откуда пришли
- о посетителях
- география
- сессии
- аудитория
- сравнение аудиторий
- сравнение посещаемости

Смотрите также

Код счетчика

Реклама в поиске

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 выходы счетчика

Показатель	Сегодня	Вчера	За 7 дней	За 30 дней	Всего
Всего	18	19	133	785	37 803
В том числе:					
Через публичные проxy	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

Рис. 41.3. Система bigmir.net

Данный код нужно установить на каждой странице сайта или один раз в TPL-шаблоне сайта.

Что же касается bigmir)net (<http://top.bigmir.net/>), то эта система статистики 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;
- ☐ **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.



Голосования (опросы)

42.1. Разработка собственной системы голосования

Системы опросов давно стали популярными, и сейчас голосование можно встретить практически на любом сайте. В Интернете можно найти много уже готовых систем голосования, например Advanced Poll. С использованием этих систем, как правило, не возникает проблем, и вы всегда сможете разобраться самостоятельно. В этой главе 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 базе данных будут три таблицы:

- ☐ polls — данная таблица будет содержать вопросы голосований;
- ☐ answers — таблица вариантов ответов на вопросы голосований;
- ☐ votes — ответы пользователей на вопросы голосований.

Следующие SQL-операторы создают наши таблицы:

```
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 будет такой же, как и ранее, 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';
```

```
?>
```

42.2. Разработка сценария poll_form.php

Разработку нашей системы опросов начнем со сценария poll_form.php. Данный сценарий довольно прост: ему нужно передать идентификатор голосования, и он отобразит указанный опрос.

Первым делом 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>";  
?>
```

Данный сценарий только выводит голосование (рис. 42.1). Обработкой голосования занимается наш следующий сценарий — `poll_process.php`.

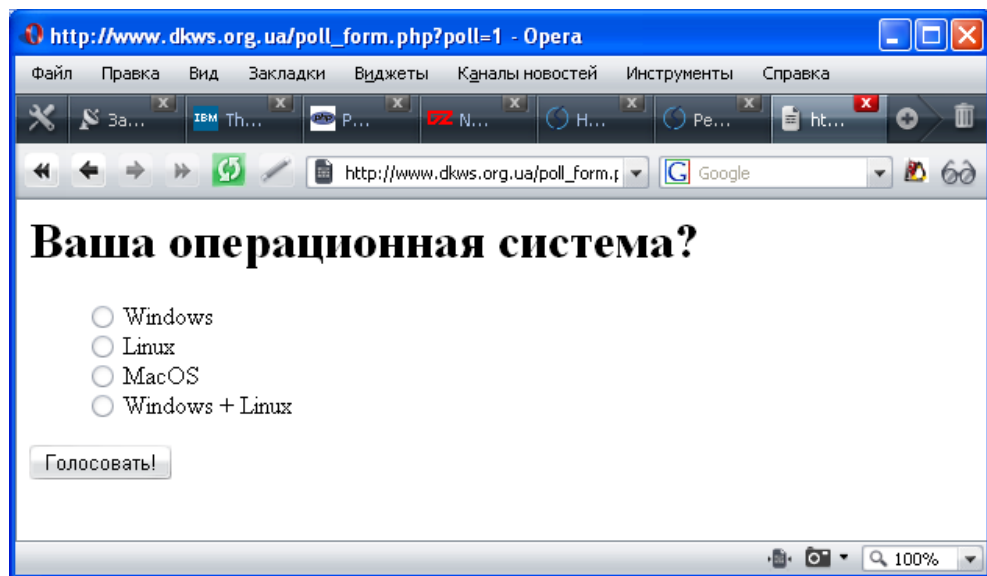


Рис. 42.1. Форма голосования

42.3. Сценарий `poll_process.php`

Задача данного 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 объекты с указанными идентификаторами в нашей БД:

```
$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");

?>
```

42.4. Сценарий poll_results.php

Теперь осталось вывести результаты голосования. Ради упрощения сценария для построения графиков мы не будем использовать возможности GD, а воспользуемся возможностями 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"];
```

Теперь 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"];
```

Теперь получим счетчики каждого голоса. Для этого будем использовать огромный запрос. Если вы не понимаете, что он делает, то прочитайте внимательно *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;">';
```

На этом 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 phpMyAdmin. Для вывода результатов голосования (рис. 42.2) можно использовать библиотеку GD.

Все файлы разработанной системы голосования доступны в каталоге Glava_42 прилагаемого диска.

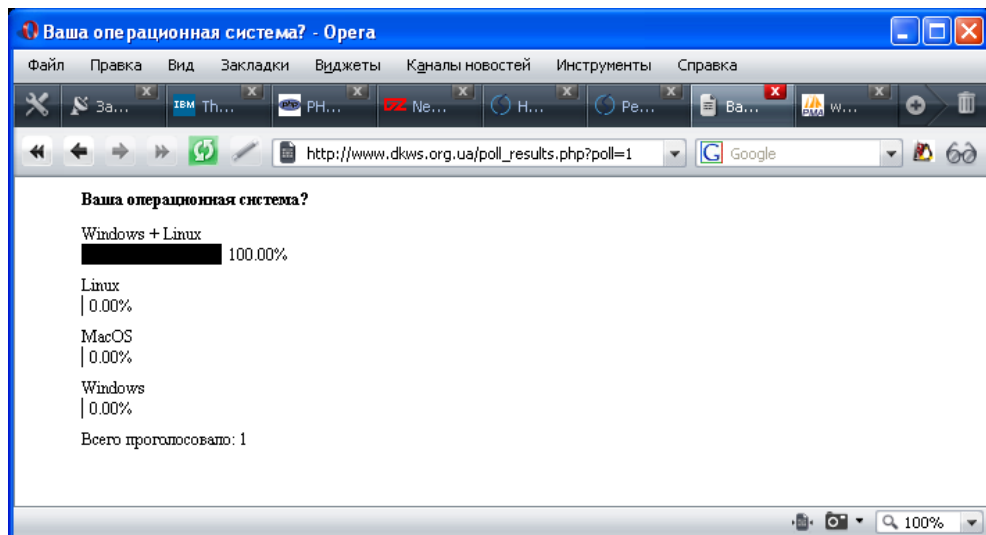


Рис. 42.2. Результаты голосования



РАЗДЕЛ 9

Сложный проект: сайт по продаже недвижимости



Постановка задачи

43.1. Функции будущего проекта

В данном разделе будет рассмотрено создание довольно сложного проекта. Сразу хочется отметить, что поскольку проект сложный, PHP-кода будет много, поэтому весь он приводиться в книге не будет, но вы его найдете на прилагаемом диске. На диске находится полный код системы (сценарии отображения и поиска объявлений, рабочий кабинет пользователя и панель управления). Система полностью готова к использованию и уже более полугода (на момент написания этих строк) используется в "боевых условиях" (а это говорит о том, что все листинги полностью рабочие и за полгода в них исправлены всевозможные небольшие "баги"). Адрес сайта сообщать не буду, чтобы не создавать лишней рекламы.

Рассмотрим основные функции нашей системы (функции панели управления будем рассматривать в *главе 45*).

- ❑ Возможность поиска объявлений по типу недвижимости (квартира, дом и т. д.), по области и по районному центру.
- ❑ Дополнительные критерии поиска (цена, наличие фотографии и т. д.).
- ❑ Поиск по статьям (материалам), размещенным на сайте.
- ❑ Регистрация пользователей. Зарегистрированные пользователи могут использовать недоступные гостям части сайта — Мой кабинет (здесь пользователь может управлять своими объявлениями — удалять, изменять текст и фотографии) и Мой блокнот (в блокнот можно добавить интересные объявления).
- ❑ Добавление объявлений (объявления могут добавить, как зарегистрированные пользователи, так и гости).

- ❑ Вывод статей и новостей сайта.
- ❑ Дополнительные функции для модератора (например, удаление объявления без регистрации в панели управления).

Как видите, изюминка сайта — функции Мой кабинет и Мой блокнот. Эти функции заставляют пользователей регистрироваться, причем регистрация будет полезной как тем, кто продает недвижимость (им будет интересна функция Мой кабинет), так и тем, кто хочет купить (в Мой блокнот можно добавить заинтересовавшие пользователя объявления).

Зарегистрированный пользователь может быть модератором. Модератор может работать с системой, как любой другой зарегистрированный пользователь, то есть производить поиск объявлений, добавлять свои объявления. Помимо этого он может удалять объявления других пользователей, если сочтет это нужным.

43.2. Разработка базы данных

Проектирование системы нужно начать с базы данных. Начнем с таблицы `n_type`, которая представляет собой небольшой справочник типа объявлений. Эта таблица состоит (листинг 43.1) всего из двух полей — `id` (уникальный идентификатор записи) и `t` — описание типа недвижимости. В листинге 43.1 производится создание таблицы `n_type` и заполнение ее исходными данными.

Листинг 43.1. Таблица `n_type`

```
CREATE TABLE `n_type` (  
  `id` int(11) NOT NULL auto_increment,  
  `t` varchar(250) default NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;  
  
INSERT INTO `n_type` VALUES (1, 'Квартира');  
INSERT INTO `n_type` VALUES (2, 'Дом/дача');  
INSERT INTO `n_type` VALUES (3, 'Участок');  
INSERT INTO `n_type` VALUES (4, 'Офис');  
INSERT INTO `n_type` VALUES (5, 'Коммерческая');  
INSERT INTO `n_type` VALUES (6, 'Ремонт/услуги');
```

Предусмотрено шесть типов объявлений: первые пять типов — это типы недвижимости, а шестой тип — это тип объявлений о предоставлении услуг по ремонту и строительным работам.

Далее создадим таблицы областей (регионов) и городов (районных центров), см. листинг 43.2.

Листинг 43.2. Таблицы `n_regions` и `n_towns`

```
CREATE TABLE `n_regions` (  
  `id` int(11) NOT NULL auto_increment,  
  `rname` varchar(120) default NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;  
  
CREATE TABLE `n_towns` (  
  `id` int(11) NOT NULL auto_increment,  
  `regid` int(11) default NULL,  
  `tname` varchar(100) default NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Первая таблица (`n_regions`) выполнена по принципу таблицы `n_type` — поле `id` и текстовое поле `rname`, содержащее название области. Таблица `n_towns` чуть сложнее: кроме полей `id` (идентификатор) и `tname` (название города) она содержит поле `regid` — идентификатор области, к которой относится районный центр. Вообще в таблицу `n_towns` можно добавлять не только районные центры, но населенные пункты поменьше. Тут вопрос в ином — нужно ли вам это и хватит ли у вас терпения внести все населенные пункты в области?

Для Украины список областей выглядит так:

```
INSERT INTO `n_regions` VALUES(1, 'Винницкая');  
INSERT INTO `n_regions` VALUES(2, 'Волынская');  
INSERT INTO `n_regions` VALUES(3, 'Днепропетровская');  
INSERT INTO `n_regions` VALUES(4, 'Донецкая');  
INSERT INTO `n_regions` VALUES(5, 'Житомирская');  
INSERT INTO `n_regions` VALUES(6, 'Закарпатская');  
INSERT INTO `n_regions` VALUES(7, 'Запорожская');
```

```
INSERT INTO `n_regions` VALUES (8, 'Ивано-Франковская');
INSERT INTO `n_regions` VALUES (9, 'Киевская');
INSERT INTO `n_regions` VALUES (10, 'Кировоградская');
INSERT INTO `n_regions` VALUES (11, 'Луганская');
INSERT INTO `n_regions` VALUES (12, 'Львовская');
INSERT INTO `n_regions` VALUES (13, 'Николаевская');
INSERT INTO `n_regions` VALUES (14, 'Одесская');
INSERT INTO `n_regions` VALUES (15, 'Полтавская');
INSERT INTO `n_regions` VALUES (16, 'Ровенская');
INSERT INTO `n_regions` VALUES (17, 'Сумская');
INSERT INTO `n_regions` VALUES (18, 'Тернопольская');
INSERT INTO `n_regions` VALUES (19, 'Харьковская');
INSERT INTO `n_regions` VALUES (20, 'Херсонская');
INSERT INTO `n_regions` VALUES (21, 'Хмельницкая');
INSERT INTO `n_regions` VALUES (22, 'Черкасская');
INSERT INTO `n_regions` VALUES (23, 'Черниговская');
INSERT INTO `n_regions` VALUES (24, 'Черновицкая');
INSERT INTO `n_regions` VALUES (25, 'АР Крым');
```

Для РФ список областей можно найти в Интернете, например здесь:

<http://www.gks.ru/scripts/free/1c.exe?XXXX32F.5.4.1/010210R>

Вот пример заполнения таблицы `n_towns`:

```
...
INSERT INTO `n_towns` VALUES (82, 4, 'Горловка');
INSERT INTO `n_towns` VALUES (83, 4, 'Дебальцево');
INSERT INTO `n_towns` VALUES (84, 4, 'Дзержинск');
INSERT INTO `n_towns` VALUES (85, 4, 'Доброполье');
INSERT INTO `n_towns` VALUES (86, 4, 'Донецк');
INSERT INTO `n_towns` VALUES (87, 4, 'Дружковка');
INSERT INTO `n_towns` VALUES (88, 4, 'Енакиево');
...
```

Первое поле — идентификатор записи (города), второе — идентификатор области (в данном случае — 4), далее — название населенного пункта.

Настало время разработать главную таблицу, содержащую все объявления, кроме объявлений о предоставлении услуг (листинг 43.3).

Листинг 43.3. Основная таблица — n_main

```
CREATE TABLE `n_main` (  
  `id` int(11) NOT NULL auto_increment,  
  `dt` bigint(20) NOT NULL,  
  `vip` int(11) NOT NULL default '0',  
  `uid` int(11) default '1',  
  `sale` int(11) default '1',  
  `lease` int(11) default '0',  
  `obmen` int(11) default '0',  
  `region_id` int(11) default NULL,  
  `town_id` int(11) default NULL,  
  `town` varchar(100) default NULL,  
  `ntype` int(11) default NULL,  
  `rooms` int(11) default NULL,  
  `floor` varchar(10) default NULL,  
  `sq_all` int(11) default NULL,  
  `sq_live` int(11) default NULL,  
  `sq_k` int(11) default NULL,  
  `wall` varchar(20) default NULL,  
  `street` varchar(40) default NULL,  
  `house` varchar(5) default NULL,  
  `year` int(11) default NULL,  
  `price` int(11) default NULL,  
  `price_currency` varchar(3) default NULL,  
  `price_comment` int(11) default NULL,  
  `photo` varchar(255) default NULL,  
  `comment` text,  
  `publication` int(11) default NULL,  
  `zem_uch` int(11) default NULL,  
  `target` int(11) default NULL,  
  `name_object` varchar(100) default NULL,  
  `phone` varchar(20) default NULL,
```

```

`v` int(11) NOT NULL default '0' COMMENT 'к-во просмотров',
`scode` int(10) NOT NULL COMMENT 'Секретный код для удаления объявления',
PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;

```

Описание полей таблицы `p_main` представлено в табл. 43.1 (полей-то много, а таблица улучшает восприятие информации).

Таблица 43.1. Поля таблицы `p_main`

Поле	Описание
<code>id</code>	Уникальный идентификатор объявления
<code>dt</code>	Дата добавления объявления
<code>vip</code>	Флаг VIP-объявления. При выводе результатов поиска сначала выводятся VIP-объявления, а потом — все остальные. Значение по умолчанию — 0, что означает, что объявление не является VIP-объявлением
<code>uid</code>	Идентификатор пользователя, добавившего объявление. Значение по умолчанию — 1, это идентификатор гостя (незарегистрированного пользователя)
<code>sale</code>	Если это поле содержит 1, то это объявление о продаже недвижимости
<code>lease</code>	Если это поле содержит 1, то перед нами объявление об аренде недвижимости
<code>obmen</code>	Объявление об обмене. Вообще-то можно было создать одно поле, например <code>type</code> , и объявить, что значение 1 — это продажа, 2 — аренда, а 3 — обмен. Но не все так просто — ведь объявление может быть и о продаже и об обмене. Например, кто-то или продает квартиру, или желает обменять ее на другую. В конечном итоге три разных поля позволяют сделать более понятными SQL-запросы. Вы же можете использовать одно поле (правда, тогда придется немного модифицировать код системы), если вам так больше нравится. И еще: если вас смущает название поля, тоже можно его изменить — когда разрабатывалась система, никто не планировал публиковать ее код в книге, поэтому о названии полей я задумывался меньше всего

Таблица 43.1 (продолжение)

Поле	Описание
region_id	Идентификатор региона
town_id	Идентификатор города
town	Текстовое поле, которое может заполнить пользователь, если его города нет в списке
ntype	Тип объявления (см. таблицу n_type)
rooms	Количество комнат
floor	Информация об этаже и этажности, например 3/9 — третий этаж девятиэтажного дома
sq_all	Общая площадь
sq_live	Жилая площадь
sq_k	Площадь кухни
wall	Тип стен дома (кирпич, дерево и т. д.)
street	Название улицы, если есть желание ее указать
house	Номер дома
year	Год постройки дома
price	Цена
price_currency	Валюта, в которой указана цена
price_comment	Комментарий к цене, актуален для объявлений об аренде квартиры, например за месяц, за год, или для объявлений и продажа новых квартир — как правило, цена указывается за квадратный метр
photo	Имя файла с основной фотографией квартиры
comment	Комментарий — произвольный текст, введенный пользователем
publication	Период публикации (в днях) объявления
zem_uch	Актуально для продажи дома с земельным участком, в этом случае это поле содержит площадь земельного участка

Таблица 43.1 (окончание)

Поле	Описание
target	Целевое назначение недвижимости, актуально для участков под застройку
name_object	Название объекта, указывается только для коммерческой недвижимости
phone	Телефон продавца. Данное поле заполняется, только если объявление добавляется незарегистрированным пользователем. Контактная информация зарегистрированных пользователей хранится в таблице n_users
v	Количество просмотров объявления, поле увеличивается движком при каждом просмотре объявления
scode	Секретный код. Когда объявление добавляется гостем, ему сообщается секретный код (сгенерированный случайным образом). Если пользователь захочет удалить объявление, ему нужно ввести секретный код объявления. Зарегистрированные пользователи смогут удалить объявления без ввода секретного кода, используя функцию Мой кабинет

Поле uid таблицы n_main связано с таблицей n_users, SQL-оператор создания которой представлен в листинге 43.4.

Листинг 43.4. Таблица n_users

```
CREATE TABLE `n_users` (
  `id` int(11) NOT NULL auto_increment,
  `email` varchar(100) NOT NULL,
  `uname` varchar(100) NOT NULL,
  `phone1` varchar(100) NOT NULL,
  `phone2` varchar(20) default NULL,
  `region_id` int(11) default NULL,
  `town` varchar(50) default NULL,
  `pas` varchar(254) default NULL,
  `sendm` int(11) default NULL,
  `fio` varchar(128) default NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Первое поле — это уникальный идентификатор пользователя. Идентификатор с номером 1 — это гость:

```
INSERT INTO `n_users` VALUES(1, '-', 'Гость', '-', NULL, NULL, NULL, NULL, NULL, '');
```

Второе поле — электронный адрес пользователя, `uname` — имя пользователя, потом следуют два поля, в которых хранятся два телефона пользователя, например домашний и мобильный. После этого представлены поля `region_id` и `town`. Что должно храниться в этих полях, вы уже догадались, — область и город. Поле `pass` — это пароль пользователя. Его можно хранить в зашифрованном виде (MD5) или в открытом виде. Какой вариант выбрать — решать вам. С точки зрения безопасности лучше хранить пароль в зашифрованном виде (зашифровать пароль можно функцией `md5()`). Впрочем, о паролях мы поговорим в следующей главе, а пока нам нужно знать, что поле `pas` используется для хранения пароля, а в каком виде на данный момент не важно.

Если поле `sendm` содержит 1, пользователь разрешил получение сообщений от администратора сайта (в панели управления будет реализована функция массовой рассылки). Последнее поле — это Ф. И. О. пользователя или название агентства недвижимости, опубликовавшего объявление.

Объявления об услугах хранятся в таблице `n_uslugi` (листинг 43.5).

Листинг 43.5. Таблица `n_uslugi`

```
CREATE TABLE `n_uslugi` (  
  `id` int(11) NOT NULL auto_increment,  
  `dt` bigint(20) NOT NULL,  
  `vip` int(11) NOT NULL default '0',  
  `uid` int(11) default '1',  
  `region_id` int(11) default NULL,  
  `town_id` int(11) default NULL,  
  `town` varchar(100) default NULL,  
  `vid_uslug` int(11) NOT NULL,  
  `comment` text,  
  `phone` varchar(20) default NULL,  
  `photo` varchar(250) default NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Назначение полей, думаю, вам понятно по таблице `n_main`. Есть только одно поле, отсутствующее в таблице `n_main` — `viduslug`. Для него не будем создавать отдельную таблицу-справочник, а просто определим константы видов услуг в самом движке. Так будет проще.

При публикации объявлений можно указывать не только основную фотографию, но и несколько дополнительных фотографий. Информация о дополнительных фотографиях будет храниться в таблице `n_fotos` (листинг 43.6).

Листинг 43.6. Таблица `n_fotos`

```
CREATE TABLE `n_fotos` (  
  `i` int(11) NOT NULL auto_increment,  
  `id` int(11) NOT NULL,  
  `fname` varchar(250) default NULL,  
  PRIMARY KEY (`i`)  
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Поле `i` — уникальный идентификатор записи в таблице `n_fotos`, используется для изменения записи. Второе поле (`id`) — это идентификатор объявления, к которому "привязана" фотография. Третье поле — имя файла фотографии. Рассмотрим, как будет заполняться эта блица:

```
INSERT INTO `n_fotos` VALUES(1, 27, 'fotos/phpahCaAe.jpg');  
INSERT INTO `n_fotos` VALUES(2, 27, './fotos/phpahCaAe.jpg');  
INSERT INTO `n_fotos` VALUES(5, 36, './fotos/phpV2djOo.jpg');
```

Файлы `phpahCaAe.jpg` и `phpV2djOo.jpg` относятся к объявлению № 27, а файл `phpV2djOo.jpg` — к объявлению № 36. Следовательно, при выводе объявления № 27 будут показаны первые две фотографии, а при выводе объявления № 36 будет показана последняя фотография — `phpV2djOo.jpg`.

Для реализации функции Мой блокнот нам понадобится дополнительная таблица `n_notepad` (листинг 43.7). Поле `id` — идентификатор записи, поле `uid` — идентификатор пользователя, `oid` — идентификатор объявления.

Листинг 43.7. Таблица `n_notepad`

```
CREATE TABLE `n_notepad` (  
  `id` int(11) NOT NULL auto_increment,
```

```
`uid` int(11) NOT NULL,  
`oid` int(11) NOT NULL,  
PRIMARY KEY (`id`)  
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Предположим, что пользователь с идентификатором 4 добавил в блокнот объявление № 242. Тогда система выполнит следующий SQL-запрос:

```
INSERT INTO `n_notepad` VALUES(25, 4, 242);
```

Аналогично, при добавлении пользователем № 2 в свой блокнот объявлений № 365 и № 352 будут выполнены следующие запросы:

```
INSERT INTO `n_notepad` VALUES(26, 2, 365);
```

```
INSERT INTO `n_notepad` VALUES(27, 2, 352);
```

Примеры запросов я привожу, чтобы вы понимали, как будет работать система — это очень важно.

Чуть выше было отмечено, что наша система будет выводить статьи и новости. Для хранения статей и новостей нам понадобятся две таблицы — `n_art` (статьи) и `n_news` (новости), см. листинг 43.8.

Листинг 43.8. Таблицы `n_art` и `n_news`

```
CREATE TABLE `n_art` (  
  `id` int(11) NOT NULL auto_increment,  
  `title` varchar(250) default NULL,  
  `html` text,  
  PRIMARY KEY (`id`)  
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

```
CREATE TABLE `n_news` (  
  `id` int(11) NOT NULL auto_increment,  
  `dt` int(11) default NULL,  
  `title` varchar(250) default NULL,  
  `html` text,  
  PRIMARY KEY (`id`)  
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Таблица `n_art` более простая: поле `id` — идентификатор статьи, `title` — заголовков статьи, а `html` — текст статьи в формате HTML. Таблица `n_news` чуть сложнее — она содержит дополнительное поле `dt`, в котором хранится дата публикации новости — для новостей это актуально. Редактирование таблиц `n_art` и `n_news` осуществляется только через панель управления сайтом. Также через панель управления сайтом будут редактироваться еще две таблицы — `n_adv` и `n_static` (листинг 43.9). Первая таблица содержит код баннеров, которые будут демонстрироваться при выводе объявлений (ведь мы же создаем систему не просто так — нам нужно на ней заработать), а во второй таблице будет HTML-код "статических" (редко изменяющихся) страниц, например контактная информация. Подробно эти таблицы мы рассмотрим в *главе 45*.

Листинг 43.9. Таблицы `n_adv` и `n_static`

```
CREATE TABLE `n_adv` (  
  `id` int(11) NOT NULL auto_increment,  
  `comment` varchar(100) default NULL,  
  `html` text,  
  PRIMARY KEY (`id`)  
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;  
  
CREATE TABLE `n_static` (  
  `id` int(11) NOT NULL auto_increment,  
  `html` text,  
  PRIMARY KEY (`id`)  
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

На этом разработка базы данных завершена. Создать все таблицы, рассмотренные в этой главе, можно с помощью `phpMyAdmin`. В следующей главе мы приступим к разработке основной части системы, а в *главе 45* — к разработке панели управления.



Разработка основной части сайта

44.1. С чего начать

Когда начинаешь большой проект, то справедливо выражение "глаза боятся, а руки делают". Я предпочитаю начинать с конфигурационного файла, в котором будут храниться глобальные переменные, например параметры доступа к СУБД. Конфигурационный файл нашего проекта представлен в листинге 44.1.

Листинг 44.1. Конфигурационный файл config.php

```
<?
// параметры доступа к MySQL
$server = 'localhost';
$user = 'dom';
$pswd = 'password';
$db = 'dom';

// логин и пароль для доступа к админке
$admin = "admin";
$passw = "123";

// ID модератора
$moder = 4;
```

```
// каталог с картинками
$image_folder = "foto/";

$admin_mail = "root@localhost";

?>
```

Разберемся, для чего используются переменные. Первая группа переменных — это параметры доступа к СУБД. Тут все ясно. Переменные `$admin` и `$passwd` содержат имя пользователя и пароль администратора. Имена пользователей и их пароли хранятся в базе данных, а вот имя администратора и его пароль (для доступа к панели управления) хранятся в конфигурационном файле. При желании можно хранить не пароль в открытом виде, а его MD5-хэш, что безопаснее.

Переменная `$image_folder` — каталог для хранения картинок. Не забудьте создать этот каталог и установить права доступа к нему 777:

```
mkdir foto
chmod 777 foto
```

Переменная `$admin_email` содержит электронный адрес администратора. А вот переменную `$moder` я специально пропустил. Данная переменная содержит ID пользователя, являющегося модератором. Модератор регистрируется в системе как обычный пользователь, но при просмотре объявлений он может их редактировать и удалять. Это позволяет избавиться от функций модерирования в панели управления сайтом. Зачем дублировать код вывода объявлений, если достаточно добавить проверку, является ли пользователь модератором, и вывести две дополнительных ссылки — **Редактировать как модератор** и **Удалить как модератор**?

Идея довольно удачная, но не совсем хорошо реализована, поскольку для назначения модератора требуется редактировать `config.php`. Но моему заказчику больше и не нужно было. А вы, если понадобится, можете создать таблицу `moderators`, куда будете заносить ID модераторов. Правда, в панели управления придется предусмотреть редактирование этой таблицы.

44.2. Реализация основных функций системы

После создания конфигурационного файла можно приступить к реализации основных функций нашей системы. Поскольку полный код системы будет доступен в каталоге Glava_44 (в том числе и код панели управления) прилагаемого диска, то полный код файла `index.php` выкладывать не буду — это только увеличит размер книги, а эффекта должного все равно не будет, поскольку вы будете использовать уже готовый вариант. К тому же размещение части информации на диске можно расценивать как попытку борьбы с пиратством. Как правило, пираты выкладывают только отсканированный текст книги, но не прилагаемый диск (накладно выкладывать по 600–700 Мбайт информации для каждой книги!). Поэтому если вы читаете пиратский "электронный вариант" книги, вам не повезло — полного кода рассматриваемой системы вам не видать.

44.2.1. Аутентификация пользователей

Поскольку наш проект поддерживает регистрацию пользователей в системе, а также содержит функции, доступные только зарегистрированным пользователям, разработку системы нужно начать с аутентификации пользователей. В листинге 44.2 приведен фрагмент файла `index.php`, отвечающий за аутентификацию пользователей.

Листинг 44.2. Аутентификация пользователей

```
<?
session_name('iseekhouse'); // имя сессии
session_start();           // запуск сессии

include "config.php";      // подключаем конфигурационный файл

// подключаемся к серверу БД и выбираем БД
mysql_connect($server, $user, $pswd) or die(mysql_error());
mysql_select_db($db) or die(mysql_error());
```

```
// проверка имени пользователя и пароля
if ($p=="login") {

// запрос
$q = "select * from n_users where (uname=\"\$username\") and
(pas=\"\$password\")";

// выполняем запрос
$r = mysql_query($q);

// если получен результат
if (mysql_num_rows($r) > 0) {
    $row=mysql_fetch_array($r);

    // то устанавливаем переменные сессии

    session_register("session_username");        // логин
    session_register("session_uid");              // ID пользователя
    session_register("session_fio");              // Ф. И. О.
    // флаг успешной аутен-ции
    session_register("auth");
    $session_username = $username;
    $session_fio = $row[fio];
    $session_uid=$row[id];
    $auth=1;
    $first=1;
}
else {
    // сообщение
    echo "<script>alert('Неправильные имя пользователя
        или пароль')</script>";

    // уничтожаем переменную p, чтобы вывести главную страницу
    unset($p);
}
}
```

```
// если пользователь не аутентифицирован, выводим форму ввода имени
// пользователя и пароля
if (!isset($auth)) {
    $au_info = file_get_contents('auth.html');
}
else
    {

if (strlen($session_fio)==0)
$nm = $session_username;
else $nm = $session_fio;

// формируем приветствие, позже мы его выведем
$au_info = "<br>Добро пожаловать, $nm<br>
<a href=index.php?p=cabinet>Мой кабинет</a>
<br><a href=index.php?p=notepad>Мой блокнот</a>
<br><a href=index.php?p=profile>Мой профиль</a>
<br><a href=index.php?p=logout>Выйти</a>";

        //unset($p);
    }

// подключаем файл со служебными функциями
include "functions.php";

// получаем форму поиска объявления
// замените localhost на имя узла или используйте переменные окружения,
// чтобы получить имя узла автоматически и сделать сценарий более
// универсальным
$form = file_get_contents('http://localhost/form.php');
```

Теперь рассмотрим этот код более подробно. Переменная `$p` принимает значение `login`, если пользователь ввел имя пользователя и пароль в форме аутентификации, которая содержится в файле `auth.html`. Приводить код этого файла не стану (вы всегда можете посмотреть его на диске), зато на рис. 44.1 показано, как выглядит эта форма в жизни.

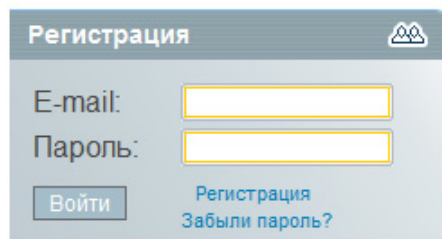
A registration form titled "Регистрация" (Registration) with a user icon in the top right corner. It contains two input fields: "E-mail:" and "Пароль:" (Password:). Below the password field is a "Войти" (Login) button. To the right of the login button are two links: "Регистрация" (Registration) and "Забыли пароль?" (Forgot password?).

Рис. 44.1. Форма аутентификации

Форма аутентификации выводится, только если пользователь не аутентифицирован. Если пользователь прошел аутентификацию, то есть имя пользователя и пароль правильные, то отображается приветствие (переменная `$au_info`). Приветствие изображено на рис. 44.2.

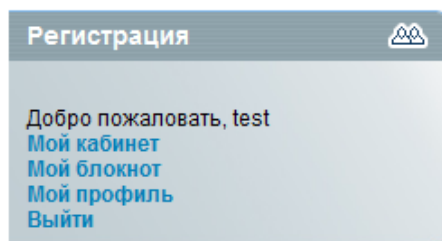
A user greeting box titled "Регистрация" (Registration) with a user icon in the top right corner. It displays the text "Добро пожаловать, test" (Welcome, test). Below this text are four links: "Мой кабинет" (My cabinet), "Мой блокнот" (My notebook), "Мой профиль" (My profile), and "Выйти" (Logout).

Рис. 44.2. Приветствие для зарегистрированного пользователя

В файле `functions` содержатся служебные функции, необходимые для работы нашей системы. Назначение функций понятно из их названия, например функция `show_news()` показывает новости, а функция `ls()` отображает объявления, принадлежащие конкретному пользователю. Код некоторых функций (наиболее сложных) будет рассмотрен далее. Код простых функций вообще рассматриваться не будет — он понятен и без моих комментариев (если, конечно, вы читали эту книгу по порядку, а не начали чтение с этой главы).

ПРИМЕЧАНИЕ

Поскольку код, приведенный в книге, используется в реальном проекте, а не разрабатывался специально для книги, то из всех файлов, прилагаемых на диске, исключены элементы дизайна. Нет ни картинок, ни таблиц стилей, ни JavaScript-сценариев.

Файл `form.php` выводит форму поиска (рис. 44.3). Файл является второстепенным, поэтому его листинг рассматривать мы не будем. Главное, чтобы вы знали, для чего используется этот файл.

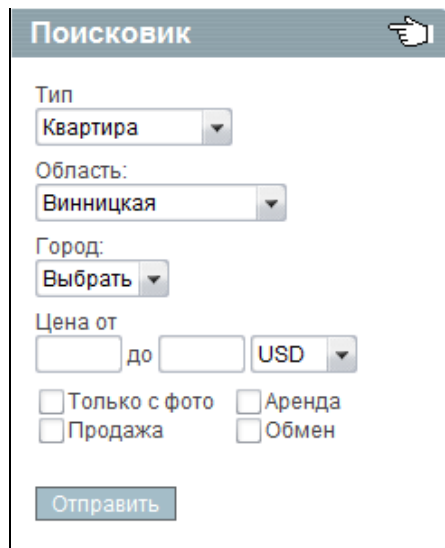


Рис. 44.3. Форма поиска

44.2.2. Вывод VIP-объявлений

Наша система предусматривает возможность задания особого статуса для объявлений — VIP. Пока система работает в тестовом режиме, сделать свое объявление привилегированным может любой пользователь и при этом абсолютно бесплатно. В дальнейшем вы можете доработать приведенный движок и добавить возможность оплаты за получение VIP-статуса. Способ оплаты — выбирайте сами, от этого также будет зависеть РНР-код для обработки финансовых вопросов. А пока разберемся с выводом VIP-объявлений. VIP-объявления выводятся только на главной странице блоком в девять объявлений (3×3), после них выводятся объявления о строительных услугах, а после — объявления о продаже недвижимости. На рис. 44.4 показан блок VIP-объявлений.

Код, обеспечивающий вывод VIP-объявлений, представлен в листинге 44.3.

Последние VIP-объявления

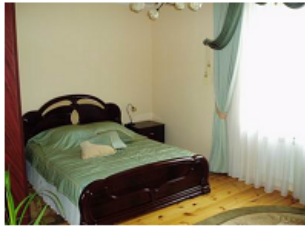
		
Ирпень (Ворзель) 349000	КИЕВ 450000	Ирпень 300000
		
Перечанка 230000	Буча 160000	Ирпень 220000
		
Ирпень 400000	Бородянка 175000	Запорожье 150000

Рис. 44.4. Блок VIP-объявлений

Листинг 44.3. Вывод VIP-объявлений

```
$q = 'SELECT *
FROM n_main
WHERE vip>0
ORDER BY RAND()
LIMIT 9';
```

```
$r = mysql_query($q) or die(mysql_error());
```

```
echo "<table>
<tr>";
$i=0;

while ($row=mysql_fetch_array($r)) {

// функция get_town() описана в function.php
$town = get_town($row[town_id]);

// если счетчик делится на 3 без остатка, мы выводим <TR>, то есть
// начинаем новый ряд объявлений
if ($i %3 == 0) echo "<tr>";

// выводим ссылку на объявление, фото, город и цену
echo "<td width=\"33%\">

<a href=index.php?p=show&id=$row[id]>
<img src=$row[photo] width=218 height=161px border=\"0\"></a>

<table>
<tr>
<td width=60% valign=\"top\">$town</td>
<td align=\"right\"><b>$row[price]</b></td>
</tr>
</table>
<br></td>";

$i++;          // увеличиваем счетчик
}
...
```

Сначала мы получаем девять случайных (используется SQL-функция `RAND()`) объявлений, где поле `VIP > 0`. Затем выводим их по три объявления в ряду.

44.2.3. Вывод объявлений об услугах

Вывод объявлений об услугах производится аналогично выводу VIP-объявлений, только выборка производится из таблицы `n_uslugi`:

```
$q = "select * from n_uslugi order by id desc limit 9";  
$r = mysql_query($q);
```

При выводе объявлений мы используем функции, определенные в `function.php`:

```
$vid = get_vid_uslug($row[vid_uslug]); // определить тип услуг  
$town = get_town($row[town_id]); // определить город по номеру  
$region = get_region($row[region_id]); // определить регион по номеру
```

Проверяем, была ли указана фотография при публикации объявления. Если нет, тогда выводим "заглушку" — файл `no_photo.jpg` в качестве фотографии:

```
if (strlen($row[photo])>0)  
$photo = "<a href=index.php?p=u&id=$row[id]><img src=$row[photo]></a>";  
else $photo = "<a href=index.php?p=u&id=$row[id]><img  
src=no_photo.jpg></a>";
```

А теперь самое интересное. Нам нужно проверить, является ли текущий пользователь модератором. Для этого мы сравниваем переменную `$session_uid`, содержащую идентификатор текущего пользователя, с переменной `$moder`, определенной в `config.php`. Если пользователь является модератором, под каждым объявлением мы выводим ссылку **Удалить как модератор**, позволяющую удалить объявление, не являясь его владельцем:

```
$moder_d = "";  
if ($session_uid==$moder)  
    $moder_d = "<br><a href=index.php?p=moderdelete_u&id=$row[id]>  
Удалить (как модератор)</a>";
```

После вывода девяти объявлений о строительных услугах мы выводим ссылку, позволяющую просмотреть все остальные объявления подобного типа:

```
echo "<p><a href=index.php?p=allu>Все объявления о строительных  
услугах</a>";
```

44.2.4. Вывод рекламных баннеров

В предыдущей главе была описана таблица `p_adv`, содержащая код баннеров. Для вывода баннера используется функция `get_adv_block()`, которой нужно передать номер выводимого баннера:

```
$adv = get_adv_block(5);      // получаем баннер
echo "<br>$adv<br>";           // и выводим его
```

44.2.5. Постраничный вывод объявлений

Вот мы и добрались до основной функции нашего движка — вывода объявлений о продаже недвижимости. Мы будем выводить по десять объявлений на странице. Внешний вид выводимых объявлений показан на рис. 44.5.

Всего записей 381

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 [Далее](#)

#470 КИЕВ
135000 USD



Киевская обл.
К-во комнат: 2
Этаж:
ул.Эрнста, 12 2х комнатная раздельная Соломенский р-н ул.Эрнста, 12 5/16кб АППС — люкс 67кв.м. общая, жилая 35кв.м, кухня 11.5 1 лоджия- застеклена и оборудована, евроремонт, паркет, мебель, встроенная кухня с бытовой техникой. Чистая продажа, без опекунского.
Цена: 135 000 у.е.
[Далее](#) [В блокнот](#)

#469 КИЕВ
75000 USD



Киевская обл.
К-во комнат: 2
Этаж: 9/9
Борщаговка Борщаговка Р. Ролана, № 5\8 2х комн. раздельная, 9/9п 45/25/7 балкон на 2 комнаты, сделан хороший ремонт в с/у, и в целом хорошее состояние. Крыша не течет и стены сухие! 75000у.е.
[Далее](#) [В блокнот](#)

Рис. 44.5. Вывод объявлений

Как видите, мы выводим следующую информацию:

- ☐ город;
- ☐ цену;

- ☐ регион;
- ☐ количество комнат;
- ☐ этаж и этажность
- ☐ краткое описание;
- ☐ ссылку **Далее**, позволяющую получить дополнительную информацию (данные продавца) и просмотреть дополнительные фотографии, если они были загружены;
- ☐ ссылку **В блокнот**, позволяющую добавить объявление в блокнот;
- ☐ ссылку **Удалить как модератор**, позволяющую удалить объявление модератору.

Код, реализующий постраничный вывод объявлений, представлен в листинге 44.4. Переменная `$lineyka` — это не что иное, как постраничная навигация.

Листинг 44.4. Постраничный вывод объявлений

```
$N = 10;                      // к-во объявлений на странице

$rl=mysql_query("select count(*) as rec from n_main");
$f = mysql_fetch_row($rl);
$rec = $f[0];                 // всего записей в таблице

// если страница не указана, выводим первую
if (!is_numeric($p)) $pg=0;

// записи, которые нужно вывести
$records = $pg * $N;

echo "<p>Всего записей $rec<br>";

$lineyka="";                  // навигация

// вычисляем к-во страниц
$pages = floor($rec/$N);
if ($rec%$N==0) $pages++;
```

```
for($i=0;$i<=$pages;$i++) { $m=$i+1; $lineyka .= "<a
href=index.php?p=$i>$m</a> "; }

// запрос
$q="select * from n_main order by id desc limit ".$records.", $N";

$r=mysql_query($q); // выполняем запрос
$n = mysql_num_rows($r); // к-во записей

// если страница не первая, выводим ссылку "Назад"
if ($pg > 0) {
    $p = $pg - 1;
    $lineyka .= "<a href=index.php?p=$p>Назад</a>&nbsp;";
}

$pg++; // увеличиваем страницу

// выводим ссылку на следующие 10 записей (на след. страницу),
// если она есть, то есть количество записей, которые нужно вывести, и
// смещение не превышает общее количество записей
if ($records+$N < $rec)
    $lineyka .= "<a href=index.php?p=$pg>Далее</a>";

echo $lineyka; // выводим навигацию

// выводим объявления
for ($i=0; $i<$n; $i++)
{
    $row=mysql_fetch_array($r);

    $moder_link = "";
    if ($session_uid==$moder)
        $moder_link .= "&nbsp;<a
href=index.php?p=moderdelete&id=$row[id]>Удалить (как модератор)</a>";

    $add_to_notepad = "&nbsp;<a href=index.php?p=add_2notepad&id==$row[id]>В
блокнот</a>";
```

```
$town = get_town($row[town_id]); // определяем город
$region = get_region($row[region_id]); // определяем регион
$poselok = ""; // поселок

// поселок выводится, только если заполнено поле town таблицы n_main
// данное поле заполняется, если нужного города нет в списке и невозможно
// определить его номер. Как правило, выводятся только районные центры,
// а небольшие населенные пункты нужно вводить вручную
if (strlen($row[town])>0) $poselok = "(" . $row[town] . ")";

// выводим фото
if (strlen($row[photo])>0)
    $photo = "<a href=index.php?p=show&id=$row[id]>
<img width=160px border=0 src=$row[photo]></a>";
else
    $photo = "<index.php?p=show&id=$row[id]><img border=0
src=no_photo.jpg></a>";

// шапка объявления
echo "<table border=0 width=95% id=objavl>
<tr bgcolor=#A3D9EC>
<td width=160px>#<$row[id] <b>$town $poselok</b></td>
<td align=right class=price>$row[price] $row[price_currency]</td>
<tr>
<td valign=top width=120px>$photo<br><br></td><td valign=top
class=desc>$region обл.";

// определяем тип недвижимости, в соответствии с ним выводим
// дополнительную информацию, например этаж для квартиры и площадь для
// участка и дома
switch ($row[ntype]) {
case 1 :
case 2 :
case 4 :
case 5 : echo "<br>К-во комнат: $row[rooms]
<br>Этаж: $row[floor] ";
break;
```

```

case 3 : echo "<br>$row[sq_all] соток";
        break;
case 6 : echo "<br>$row[sq_all] кв. м. ";
        break;
}

// выводим комментарий
echo "<br>$row[comment]

<br><a href=index.php?p=show&id=$row[id]>Далее</a> $add_to_notepad
$moder_link
</td></tr></table>";

}

echo "<p>$lineyka";           // выводим навигацию

```

Вторую часть кода, заставляющую реагировать движок на ссылки вида **index.php?p=число**, вы найдете на прилагаемом диске в файле Glava_44\index.php (код начинается с 311-й строчки).

44.3. На что обратить внимание

Как уже отмечалось, полный код системы приводиться в книге не будет. Ведь только в index.php более 1000 строк кода, не говоря уже о вспомогательных сценариях. Поэтому мы разберемся с основными моментами, на которые стоит обратить внимание при исследовании кода системы. А проанализировать код настоятельно советую, если вы планируете модифицировать систему в дальнейшем, а не использовать как есть.

Показом объявления занимается страница show. Код, выводящий эту страницу, начинается с 437-й строки файла index.php. Код показа объявлений о строительных услугах начинается с 592-й строки.

Реализация функции Мой блокнот начинается 619-й строки. Рассмотрим следующий фрагмент кода:

```

// добавление в блокнот
if ($p=="add_2notepad") {

```

```
// если пользователь аутентифицирован,
if (isset($auth) && isset($session_uid))
{
    if (strlen($id)>7) die('Hacking attempt');
    // то добавляем номер выбранного объявления в его блокнот,
    // в таблицу n_notepad заносятся номер пользователя и номер объявления
    $q = "insert into n_notepad values(0,$session_uid, $id)";
    mysql_query($q);
    echo "<br><br>Объявление # $id было добавлено в ваш блокнот";
}

else echo "<br><br>Для доступа к этой функции вам нужно
        зарегистрироваться на сайте!";

}

// в файле notepad.php находятся функции вывода блокнота и управления
// объявлениями, которые добавлены в блокнот
if ($p=="notepad") {
    include "notepad.php";
}

// удаление из блокнота
if ($p=="note_delete") {
    if (isset($auth)) {

        if (strlen($id)>9) die('Hack attempt');

        // перед удалением проверяем, принадлежит ли удаляемое из общей
        // таблицы объявление пользователю — чтобы случайно не удалить
        // объявление, которое было добавлено в n_notepad другим пользователем
        $r = mysql_query("select * from n_notepad where id=$id");
        $row = mysql_fetch_array($r);
        if ($row[uid]==$session_uid) {
            mysql_query("delete from n_notepad where id=$id limit 1");
```

```
        echo "<br><br>Объявление # $row[oid] удалено из вашего  
        блокнота";  
    }  
else echo "<br><br>Отказано в доступе";  
  
}  
else echo "<br><br>Для доступа к блокноту нужно зарегистрироваться";  
}
```

Функции управления кабинетом пользователя описаны в файле `cabinet.php`, а пользовательские операции, такие как правка добавленных объявлений, добавление дополнительных фотографий, описаны в файле `user_operations.php`:

```
if ($p=="cabinet") {  
  
include "cabinet.php";  
}  
  
// удаление и правка пользовательских объявлений  
include "user_operations.php";
```

Страница `all` показывает все объявления пользователя. Функция `get_user_info()` выводит информацию о пользователе с указанным `UID` (идентификатором), а функция `ls()` выводит все объявления пользователя:

```
if ($p=="all") {  
  
echo "<br><h1>Все объявления пользователя: " . get_user_info($uid) .  
"</h1>";  
  
echo ls($uid);  
  
}
```

Страница `register` показывает форму регистрации нового пользователя, а непосредственно самой регистрацией (то есть добавлением информации о пользователе в БД) занимается страница `new_user`. Код вывода формы и ре-

гистрации пользователя находится в файлах соответственно register.php и new_user.php:

```
if ($p=="register") {

include "register.php";

}

if ($p=="new_user") {

include "new_user.php";

}
```

Если пользователь хочет добавить всего одно объявление и больше не планирует пользоваться системой, то ему не обязательно регистрироваться на сайте. Он может добавить объявление анонимно. При добавлении анонимного объявления пользователю сообщается секретный код удаления объявления — на случай, если пользователь уже продал недвижимость и желает удалить объявление из БД, чтобы не было лишних звонков. Для этого нужно выбрать функцию анонимного удаления и ввести секретный код. PHP-код, обрабатывающий удаление анонимных сообщений, начинается с 691-й строки:

```
if ($p=="delete") {

echo "<h1>Удаление анонимного объявления</h1>";
<form method=post action=index.php>
<input type=hidden name=p value=adelete>
<br>Введите номер объявления <input type=text name=id>
<br>Введите секретный код <input type=text name=scode>
<br><input type=submit value=Удалить></form>";

}

if ($p=="adelete") {

if (strlen($id)>7) die('<br><br>Попытка взлома');
if (strlen($scode)>9) die('<br><br>Попытка взлома');
```

```
$q = "select * from n_main where id=$id";
$r = mysql_query($q);
$row = mysql_fetch_array($r);

// если анонимный пользователь: проверяем, принадлежит ли
// удаляемое объявление анонимному пользователю
if ($row[uid]==1) {

// если введенный код равен секретному коду, удаляем
if ($row[scode]==$scode) {

mysql_query("delete from n_main where id=$id limit 1");
echo "<br><br>Ваше объявление удалено";
}

else echo ("<br><br>Проверьте секретный код удаления");

}

else echo "<br><br>Отказано в доступе";

}
```

Страницы, занимающиеся добавлением объявления, описаны в файлах add*.php:

```
if ($p=="add") {

include "add.php";

}

if ($p=="add2") {

include "add2.php";

}
```

```
if ($p=="add3") {  
  
include "add3.php";  
  
}  
  
if ($p=="add4") {  
  
include "add4.php";  
  
}
```

Загрузка дополнительных фотографий начинается с 877-й строчки файла `index.php`. Подробно рассматривать код загрузки мы не будем — он полностью аналогичен коду, который был уже ранее рассмотрен в этой книге (см. главу о загрузке файлов). Единственное отличие: после загрузки файла его имя заносится в БД, чтобы можно было отобразить загруженную картинку при выводе объявления.

На 817-й строчке вы найдете код, обрабатывающий изменение профиля пользователя. Мы храним пароль пользователя в открытом виде. Если вы хотите шифровать пароль, тогда нужно использовать PHP-функцию `md5()`. Если пользователь забыл пароль, то страница `lost` выведет форму напоминания пароля, а страница `sendpass` отправит пароль пользователю на e-mail, указанный при регистрации.

```
if ($p=="lost") {  
// выводит форму напоминания пароля  
include "lost.php";  
}  
  
if ($p=="sendpass") {  
// отправляет пароль пользователю  
include "sendpass.php";  
}
```

Код в сценарии `sendpass.php` написан с учетом, что мы храним незашифрованные пароли — этот код просто отправляет пароль из БД по указанному

e-mail. Если вы будете шифровать пароли функцией `md5()`, то расшифровать их невозможно. Следовательно, нужно выполнить следующие действия:

- ☐ сгенерировать новый пароль;
- ☐ зашифровать его функцией `md5()`;
- ☐ записать зашифрованный пароль в БД (в таблицу `p_users`);
- ☐ отправить пользователю сгенерированный пароль в незашифрованном виде.

В табл. 44.1 приведено описание всех файлов системы.

Таблица 44.1. Описание файлов системы

Файл(ы)	Описание
adm.php, action.php, adm-header.html	Файлы, относящиеся к панели управления системой
add.php, add2.php, add3.php, add4.php	Добавление объявления
allu.php	Выводит все объявления о строительных услугах
auth.html	Форма аутентификации
cabinet.php	Функция Мой кабинет
config.php	Конфигурационный файл
form.php	Вывод формы поиска
fotos.php	Редактирование фотографий (часть функции Мой кабинет)
functions.php	Содержит функции, используемые системой
index.php	Основной файл
lost.php	Форма восстановления пароля
new_user.php	Регистрация нового пользователя
notepad.php	Функция Мой блокнот
no_photo.jpg	Картинка, выводющаяся если у объявления нет фото

Таблица 44.1 (окончание)

Файл(ы)	Описание
preview.php	Предварительный просмотр объявления (перед публикацией)
print.php	Версия для печати
register.php	Форма регистрации нового пользователя
seek.php	Поиск объявлений
sendpass.php	Восстановление забытого пароля
show_uslugi.php	Показ объявлений об услугах
user_operations.php	Пользовательские операции над объявлением

В следующей главе мы поговорим о разработке панели управления системой.



Разработка панели администрирования

45.1. Функции панели управления

Первым делом нам нужно определить функции панели управления. Когда я разрабатывал систему, заказчик определил следующие функции:

- ☐ управление VIP-объявлениями, возможность установки статуса VIP любому объявлению вручную, ровно как и снятия этого статуса;
- ☐ массовая рассылка электронной почты всем зарегистрированным пользователям;
- ☐ некоторые общие функции управления базой данных с объявлениями;
- ☐ редактирование новостей и статей;
- ☐ редактирование рекламных баннеров;
- ☐ изменение контактной информации.

Как видите, удаление объявлений из панели управления не предусматривается: для этого есть функция модерирования, которая была реализована в *главе 44*. От себя я бы добавил возможность управления пользователями, например бан пользователей, удаление учетных записей. Но поскольку эти функции не были указаны в техническом задании, то и реализовывать я их не стал. Однако реализовать данные функции не составит особого труда — это обычная работа с таблицами, и, если вы уже знаете основы SQL, вы без проблем сможете доработать панель управления. Исходная панель управления изображена на рис. 45.1.

Войти	VIP	Рассылка	БД	Реклама	Новости	Статьи	Контакты	
Управление VIP-объявлениями								
Сделать объявление # VIP-объявлением Отправить								
Снять VIP-статус у объявления # Отправить								
Список VIP-объявлений								
#	Дата подачи	Дата VIP	Тип объявления	Регион	Город	Пользователь	Просмотр	Снять VIP
452	11.28.09	11.28.09	дом/дача	Киевская	Ирпень	aandreyko@gmail.com	Просмотреть	UnVIP
392	10.30.09	10.30.09	квартира	Одесская	Одесса	shuniastar@mail.ru	Просмотреть	UnVIP
384	10.22.09	10.30.09	квартира	Одесская	Одесса	shuniastar@mail.ru	Просмотреть	UnVIP
364	09.03.09	09.03.09	участок	Киевская	Ирпень	aandreyko@gmail.com	Просмотреть	UnVIP
351	08.21.09	08.21.09	квартира	Киевская	Буча	aandreyko@gmail.com	Просмотреть	UnVIP
350	08.21.09	08.21.09	дом/дача	Киевская	Ирпень	aandreyko@gmail.com	Просмотреть	UnVIP
180	07.11.09	08.21.09	дом/дача	Ивано-Франковская	Ивано-Франковск	Гость	Просмотреть	UnVIP
177	07.11.09	08.21.09	дом/дача	Запорожская	Запорожье	Гость	Просмотреть	UnVIP
80	06.10.09	06.10.09	участок	Киевская	Ирпень	aandreyko@gmail.com	Просмотреть	UnVIP
69	06.05.09	06.19.09	участок	Киевская	Бородянка	Андрей	Просмотреть	UnVIP
66	06.04.09	06.04.09	дом/дача	Киевская	КИЕВ	aandreyko@gmail.com	Просмотреть	UnVIP
65	06.04.09	06.10.09	дом/дача	Киевская	Буча	aandreyko@gmail.com	Просмотреть	UnVIP
64	06.04.09	06.12.09	дом/дача	Киевская	Ирпень	aandreyko@gmail.com	Просмотреть	UnVIP
62	06.02.09	06.10.09	дом/дача	Киевская	Ирпень	aandreyko@gmail.com	Просмотреть	UnVIP
61	06.02.09	06.06.09	дом/дача	Киевская	Бородянка	Гость	Просмотреть	UnVIP
41	05.30.09	06.10.09	участок	Киевская	Ирпень	aandreyko@gmail.com	Просмотреть	UnVIP

Рис. 45.1. Панель управления

Для доступа к панели управления нужно ввести следующий URL:

http://сервер/adm.php

По сути сценарий adm.php только выводит форму регистрации, затем передает управление сценарию action.php, который и реализует основные функции панели управления.

45.2. Управление VIP-объявлениями

Для вывода VIP-объявлений в виде таблицы, изображенной на рис. 45.1, используется код, приведенный в листинге 45.1.

Листинг 45.1. Вывод VIP-объявлений

```
// запрос
$r = mysql_query("select * from n_main where vip>0 order by id desc");

// цикл
while($row=mysql_fetch_array($r)) {
```

```
$town = get_town($row[town_id]);
$region = get_region($row[region_id]);

$dt = date("m.d.y", $row[dt]);
$vip = date("m.d.y", $row[vip]);

$tp = get_ntype($row[ntype]);
$user = get_user_name($row[uid]);

echo "<tr>
<td>$row[id]</td><td>$dt</td><td>$vip</td>
<td>$tp</td><td>$region</td><td>$town</td>
<td>$user</td>
<td><a target=_blank
href=index.php?p=show&id=$row[id]>Просмотреть</td>
<td><a href=action.php?p=unvip&id=$row[id]>UnVIP</td></tr>";

}
```

Функция UnVIP, то есть сброс VIP-статуса реализуется очень просто:

```
$r = mysql_query("update n_main set vip=0 where id=$id limit 1");
show_message("Статус VIP для объявления $id был успешно СБРОШЕН");
```

Соответственно, для установки VIP-статуса используется следующий код:

```
$r = mysql_query("update n_main set vip=$vip where id=$id limit 1");
show_message("Статус VIP для объявления $id был успешно установлен");
```

Функция `show_message()` определена в сценарии `function.php`. По сути это немного модифицированная версия функции `echo()`.

45.3. Массовая отправка электронного сообщения

Код, реализующий отpravку электронного сообщения всем пользователям системы, представлен в листинге 45.2.

Листинг 45.2. Массовая рассылка

```
$r = mysql_query("select * from n_users");
while ($row=mysql_fetch_array($r)) {

    if (ereg("^[_.0-9a-z-]+@[0-9a-z-]+[.]+[a-z]{2,3}$",
trim($row[email]))){

        if (mail(trim($row[email]), $subj, $msg))
            echo "Отправляем письмо по адресу $row[email]... [OK]";
        else echo "Отправляем письмо по адресу $row[email]...
[FAULT] ";
    }
}
```

Здесь тема сообщения содержится в переменной `$subj`, текст сообщения — в переменной `$msg`. Функция `ereg()` перед отправкой сообщения проверяет корректность адреса электронной почты.

45.4. Общие операции с базой данных

К общим операциям над базой данных объявлений относят следующие операции:

- ☐ удалить старые объявления (>3 месяцев с момента подачи);
- ☐ сбросить VIP, где с момента получения VIP-статуса прошло больше недели;
- ☐ сбросить VIP у всех объявлений.

По сути реализация этих операций сводится к выполнению соответствующих SQL-запросов. Вот код, удаляющий старые объявления:

```
// вычисляем разницу между текущим временем и текущим временем — 3 месяца
$current_time = time();
$raznica = 3600 * 24 * 30 * 3;           // 3 месяца
```

```
$S = $current_time - $raznica;  
$q = "delete from n_main where dt < $S";  
$r = mysql_query($q);  
show_message ("Всего удалено: " . mysql_affected_rows());
```

Сброс VIP-статуса у старых объявлений осуществляется аналогично, только иначе вычисляется разница, да и SQL-запрос будет иной:

```
$current_time = time();  
$raznica = 3600 * 24 * 7;    // 7 дней  
  
$S = $current_time - $raznica;  
  
$q = "update n_main set vip=0 where vip < $S";  
$r = mysql_query($q);  
show_message ("Всего обработано: " . mysql_affected_rows());
```

Проще всего реализовать последнюю операцию:

```
mysql_query("update n_main set vip=0");  
show_message ("VIP сброшен у всех");
```

45.5. Редактирование статей, новостей и контактов

Редактирование статей, новостей и контактов представляет собой создание новых записей, содержащих в себе поля типа text и обновление этих самых записей. Ничего сложного — с таким заданием справится даже студент с весьма удовлетворительными знаниями SQL и PHP.

ПРИМЕЧАНИЕ

Поскольку редактирование статей, новостей и контактов тесно связано с обработкой текста, настоятельно рекомендую заглянуть в каталог Edition_1 прилагаемого диска. В этом каталоге вы найдете некоторые главы в формате PDF, удаленные из 2-го издания (для уменьшения объема). В частности, в файле ch-44.pdf (это *глава 44* 1-го издания книги) вы найдете информацию о создании визуального редактора текста, а также по использованию уже готового редактора текста SPAW2.



РАЗДЕЛ 10

Обеспечение безопасности сайта



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 за сохранность данных. Да, с вашего сервера украсть информацию невозможно. Но от клиента до сервера информация передается в абсолютно открытом виде, следовательно, по пути к серверу информация 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 передаваемые данные ваших клиентов 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 не сможет воспользоваться. Конечно, и в этом случае можно обойти защиту, но это 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>.

В этой главе мы не будем 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!

Перейдите по ссылке **Управление доменами** и выберите ваш домен (у вас их может быть несколько). Если опция **Secure SSL** не включена, ее нужно включить (рис. 46.2). Если же этой опции нет или она неактивна, ваш хостинг не поддерживает SSL. Обратитесь к службе поддержки. Возможно, нужно сменить тарифный план для поддержки SSL.

Обратите внимание на параметр **Настройки private_html**. По умолчанию все файлы, которые доступны Web-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), нужно поместить в каталог `private_html`. После этого они будут доступны по протоколу HTTPS. Например, имя моего узла **dkws.org.ua** я поместил в каталог `private_html` сценарий `credit_card_info.php`. Получить доступ к нему теперь можно так:

https://dkws.org.ua/credit_card_info.php.



Рис. 46.1. Главная страница DirectAdmin

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

Таблица 46.1 (продолжение)

Переменная	Описание
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 Email — 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";
```



Защита PHP с помощью конфигурационного файла

47.1. Конфигурационный файл php.ini

Конфигурационный файл `php.ini` содержит все настройки интерпретатора PHP. В этой главе мы рассмотрим параметры PHP, относящиеся к безопасности. Но прежде разберемся, как найти `php.ini`.

В Linux/UNIX файл `php.ini` обычно находится в каталоге `/etc`. Иногда этот файл находится в каталоге `/usr/lib` или `/usr/local/lib`. В Windows этот файл 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
```

В Linux/UNIX для редактирования этого файла вам нужны права пользователя `root`. Другими словами, если у вас хостинг, а не выделенный сервер, вы не сможете редактировать этот файл.

Все директивы этого конфигурационного файла мы рассматривать не будем. В этом нет смысла, если в большинстве случаев вы не сможете их изменить.

Если вас интересует какая-то директива, ее описание на русском языке вы можете найти на следующей странице:

<http://ua2.php.net/manual/ru/ini.core.php>.

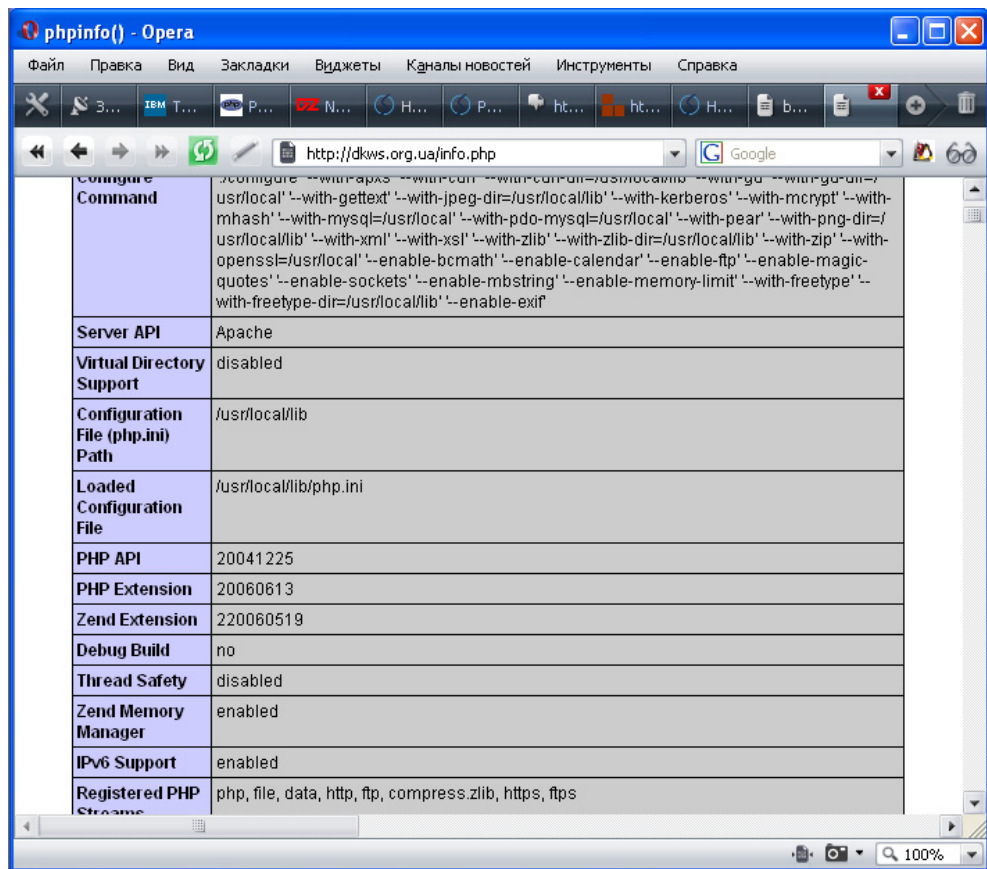


Рис. 47.1. Вывод `phpinfo()`

Однако вы можете включить ту или иную директиву для своего сценария с помощью функции `ini_set()`:

```
string ini_set ( string $директива, string $значение )
```

Например:

```
ini_set("max_execution_time", "30");
```

Для установки значения директивы конфигурации используется 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 используется функция:

```
error_reporting(E_ALL);
```

Вызов функции должен быть самым первым вызовом в сценарии. Когда сценарий отлажен и работает без ошибок, тогда нужно выключить вывод всех ошибок:

```
error_reporting(0);
```

Дело в том, что при выводе ошибки PHP выводит 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 используются 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 Web-сервера, то с помощью директивы `disable_functions` отключить 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 значения (с точки зрения безопасности) 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 Web-сервера и каталог /tmp. 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



Защита сайта от атак

48.1. Сайт в опасности

Взломать могут любой сайт. Абсолютно любой. Надеюсь, вы понимаете, что даже самую защищенную систему можно взломать. 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 — это еще полбеды, ее легко восстановить, а вот если вы потеряете свою базу данных, то, конечно, будет не очень хорошо. 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.

В этой главе 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 "кречер" ассоциируется, скорее, с пищей. Почему я не употребляю здесь слово "хакер"? Так вот, 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. А вот СМИ ошибочно навязали общественности другое понятие, заключающееся в том, что хакеры связаны со взломом компьютерных систем. Нет, наоборот, хакеры выполняют созидательную функцию в компьютерном мире, а не разрушительную. А вот взломом занимаются специалисты несколько другого профиля — кречеры (от англ. *cracker*). Чтобы это слово не резало слух, "кречеров" 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, то есть 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 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 Web-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 Web-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 будет, 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 данную 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";

?>
```

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.

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"
```

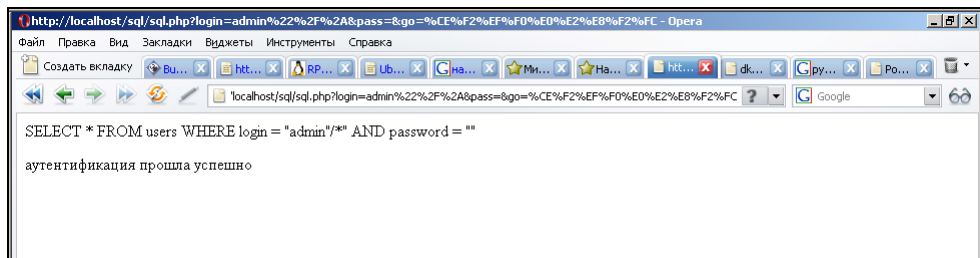


Рис. 48.1. Результат SQL-инъекции

Символы `/*` считаются началом комментария, поэтому все, что находится после них, просто не учитывается. На самом деле получается, что 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. Нужно сначала 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";

?>

```

Вывод данного 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 не "впустил" нас.

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);

```

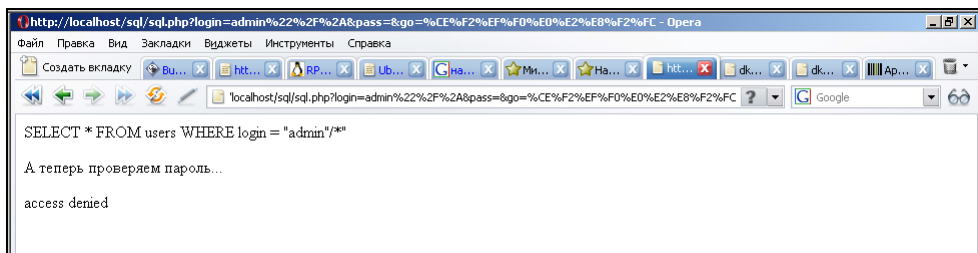


Рис. 48.2. Результат выполнения листинга 48.2

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, то есть 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 строка `'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-инъекции.

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 данной 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>
```

Как видите, мы добавили в форму скрытый параметр `ssid`, то есть 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 постороннего 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 RNPBB2 от спаммеров

Форум RNPBB версии 2 уже устарел, но он все еще установлен на многих сайтах. Бич второй версии — спаммеры. Спам появляется на форуме, даже если разрешить писать сообщения только зарегистрированным пользователям, а при регистрации включить код подтверждения. Боты (программы-роботы) спаммеров 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. В случае с `subSilver` данный 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 спаммеры, 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" => ap-
pend_sid("profile.phpEx?mode=register&agreed=true&"),
"U_AGREE_UNDER13" => ap-
pend_sid("profile.phpEx?mode=register&agreed=true&coppa=true")
```

Данные строки нужно заменить следующими строками:

```
"U_AGREE_OVER13" => ap-
pend_sid("profile.phpEx?mode=register&agreed=true&spam=9"),
"U_AGREE_UNDER13" => ap-
pend_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 спаммер 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.

ПРИЛОЖЕНИЕ

Описание CD

На прилагаемом диске вы найдете все программное обеспечение, рассмотренное в книге, документацию, а также листинги PHP-сценариев, разработанных в книге.

Содержимое диска представлено в табл. П1.

Таблица П1. Содержимое диска

Каталог	Описание	Глава
Edition_1	Главы (в формате PDF) из 1-го издания, которые были исключены из состава 2-го издания, а также описание программы phpMyAdmin	—
software	Все необходимое программное обеспечение (PHP, MySQL, Apache, phpMyAdmin, XAMPP, редакторы и пр.)	Все
php6	Бета-версии PHP 6 и MySQL 6 для Windows	Все
Glava_2	Листинги из <i>главы 2</i>	2
Glava_5	Листинги из <i>главы 5</i>	5
Glava_6	Листинги из <i>главы 6</i>	6
Glava_7	Листинги из <i>главы 7</i>	7
Glava_12	Листинги из <i>главы 12</i>	12
Glava_15	Листинги из <i>главы 15</i> , класс watermark	15
Glava_16	Листинги из <i>главы 16</i>	16
Glava_18	SQL-код из <i>главы 18</i>	18
Glava_20	Листинги из <i>главы 20</i>	20
Glava_21	Листинги из <i>главы 21</i>	21
Glava_22	Листинги и шаблоны (<i>глава 22</i>)	22

Glava_23	Листинги, шаблонизатор Smarty, документация для Smarty	23
Glava_24	Листинги из <i>главы 24</i>	24
Glava_25	Листинги из <i>главы 25</i>	25
Glava_26	Листинги из <i>главы 26</i>	26
Glava_28	Листинги из <i>главы 28</i>	28
Glava_29	Сценарий загрузки файлов	29
Glava_30	Листинги из <i>главы 30</i>	30
Glava_31	Листинги из <i>главы 31</i> , классы отправки и приема почты	31
Glava_32	Листинги из <i>главы 32</i>	32
Glava_33	Листинги из <i>главы 33</i>	33
Glava_34	Сценарий работы с MP3-файлами	34
Glava_35	Примеры использования Ajax	35
Glava_36	Простейший движок сайта, разработанный в <i>главе 36</i> , SQL-код, вспомогательные файлы	36
Glava_37	Простейшая галерея	37
Glava_38	Простейшая гостевая книга	38
Glava_40	Счетчик для сайта	40
Glava_42	Простейшая система голосования	42
Glava_44	Система продажи недвижимости	43–45
Glava_48	Листинги из <i>главы 48</i>	48

Предметный указатель

\$

\$_FILES 314

A

addAttachment() 333
addHtmlImage() 333
Advanced Poll 455
Ajax 370
Apache 16
APOP 338
AUTO_INCREMENT 426
AWFFull 450
AWStats 450

B

Bigmir)net 451
Breakpoint 300

C

CGI 47, 51
CGI-приложение 48
Content-Transfer-Encoding 328
Content-type 328
convert_cyr_string() 210
Cookies 71, 76, 277, 458
cracker 520

create_watermark() 155
crond 439
CVS 292

D

DeleteMessage() 339
DirectAdmin 510
disable_functions() 517
display_errors() 517

E

Eclipse 300
error_reporting() 517
exif_imagetype() 143
exif_read_data() 143

F

fgets() 158
file_uploads 313
FileZilla 15
fputs() 158
fread() 158
fsockopen() 158
ftp_chdir() 322
ftp_connect() 321
ftp_exec() 324

ftp_fget() 323
ftp_fput() 323
ftp_login() 321
ftp_mkdir() 322
ftp_nlist() 322
ftp_pasv() 324
ftp_pwd() 322
ftp_rmdir() 322
fwrite() 158

G

getFile() 332
getTag() 366
Google Analytics 451

H

Header() 147
HtmlMimeMail() 330
HtmlSpecialChars() 522
HTML-код 25, 382, 384, 396
HTML-файл 408
HTML-форма 55
 элементы формы 58

I

ID3-теги 364
imagecolorallocate() 150
imagecolortransparent() 152
imagecopyresampled() 154
ImageCreateFromPng() 147
ImageGif() 147
imageresize() 152
ImageString() 444
imagestring() 149
imagestringup() 150
imagettftext() 151
ini_set() 516
is_numeric() 527

J

jpeg2wbmp() 146

L

lastRSS 346
LIMIT 213
ListMessages() 339
LiveJournal 431
log_errors 517
LogMiner 450

M

mail() 328
MD5 480
MD5-хэш 430
Mercurial 292
Microsoft Internet Information Server 47
MIME 336
MP3_Id 365
mSQL 202
Multipart-форма 312
MySQL 380, 381, 387
mysql_close() 193
mysql_connect() 192
mysql_create_db() 200
mysql_db_query() 200
mysql_drop_db() 200
mysql_errno() 194
mysql_error() 194
mysql_fetch_array() 196
mysql_fetch_assoc() 196
mysql_fetch_field() 196
mysql_fetch_object() 196
mysql_fetch_row() 196
mysql_num_rows() 195
mysql_pconnect() 193
mysql_query() 195
mysql_select_db() 192

N

Netscape Communications 509

O

Open() 338

Oracle 202, 286

ORDER BY 211

P

page tagging 448

PCRE 355

PEAR 285, 365

PECL 287

Photoshop 142

PHP 380

PHP Accelerator 233

PHP Expert Editor 13

php.ini 515

PHP_OS 35

PHP_VERSION 35

phpMyAdmin 390, 411, 440, 463, 478

PHP-код 382

png2wbmp() 146

PostgreSQL 202

primary index 179

primary key 179

Q

Quoted-Printable 329

R

RegExp 355

register_globals() 79

Reply-To 328

Request For Comments, RFC 159

RetrieveMessage() 339

RSS 342

S

secondary index 179

session identifier 76, 278

session_id() 279

session_name() 77, 279

session_register() 278

session_start() 278

session_unregister() 77, 279

SET NAMES 210

setCookie() 76, 278

setFrom() 333

setHtml() 332

setReturnPath() 333

setSubject() 333

setText() 332

SID 76, 278

Smarty 232, 233

 setup.php 237

 конфигурация шаблонов 240

 модификаторы переменных 243

 переменные 239

 подключение к проекту 236

 специальные переменные 258

 стандартные функции 246

 тестирование 235

 шаблоны 238

smtp_port 329

socket_set_blocking() 163

SQL 175

SQL injection 520

sqlite_busy_timeout 207

sqlite_changes 207

sqlite_close() 204

sqlite_column() 205

sqlite_error_string() 207

sqlite_escape_string() 208

sqlite_exec() 208

sqlite_fetch_all() 208

sqlite_fetch_array() 205

sqlite_fetch_object() 208
sqlite_fetch_single() 205
sqlite_field_name() 208
sqlite_has_more() 208
sqlite_has_prev() 208
sqlite_key() 208
sqlite_last_error() 209
sqlite_last_insert_rowid() 209
sqlite_libversion() 209
sqlite_next() 206
sqlite_open() 204
sqlite_query() 205, 207
sqlite_rewind() 206
sqlite_seek() 206
sqlite_unbuffered_query() 209
SQL-запрос 390, 391, 392
SQL-модификатор:
 AUTO_INCREMENT 183
 DEFAULT 182
 NOT NULL 183
 PRIMARY KEY 182
SQL-оператор:
 CREATE 180
 DELETE 187
 GROUP BY 190
 HAVING 191
 INSERT 183
 SELECT 185
 UPDATE 184
SQL-функция:
 AVG 187
 COUNT 187
 MAX 187
 MIN 187
 SUM 187
SSL 509
strip_tags() 522
Subversion 292
Sybase 202

T

timestamp 110
TLS 509
TortoiseHG 292
TPL-шаблоны 230, 385, 410

U

upload_max_filesize 313
upload_tmp_dir 313

W

Watch-список 300
Watermark 154
Webalizer 450
Wephalizer 450
Web-сервер 408

X

XAMPP 20
XML 344, 370
XSS 520
XSS-атака 521

Z

Zend Accelerator 233
Zend Studio 13

А

Агрегатор 342
Ассоциативный массив 90
Аутентификация 74, 276

Б

Базы данных 177, 381, 389
Блок:
 try-catch 272

В

Версия для печати 409
Владелец 122
Вторичный индекс 179
Выражение 35

Г

Голосование 455
Габбер 350, 382

Д

Дескриптор 157
Деструктор 267
Директива:
 error_reporting 22, 305
 register_globals 79
Документ:
 RFC 159

Ж

"Живой" журнал 431

З

Заголовки HTTP 52
Запись 178

И

Имя:
 переменной в PHP 299
Инкапсуляция 263
Инструкция:
 echo 24
 include 225
 include_once 225, 228
 require 68, 225
 require_once 225, 228
Исключительные ситуации 272
Итераторы 275

К

Квантификатор 357
Класс 263
 абстрактные методы 269
 защищенные переменные 268
 модификаторы 269
 частные переменные 268
Клонирование 271
Команда:
 chmod 123, 480
 mkdir 480
 mysqldadmin 21
 service 21
 services.msc 18
 yum install 21
 получение вывода команды в
 переменную 40
Комментарии 24
Константы 34
Конструктор 266

М

Макросы 356
Маркировка страниц 448
Массив 89

\$_COOKIE 73, 79

\$_FILES 79

\$_GET 50

\$_POST 50

\$_REQUEST 79

\$_SESSION 78

Метод:

GET 49

POST 50

Механизм сессий 74

Н

Наследование 263, 267

О

Объект 263, 270

ООП 262

Оператор:

-> 265

break 43

continue 43

foreach 90

if 40

instanceof 274

switch-case 43

throw 272

равенства 38

Операции:

арифметические 36

логические 36

получение вывода системной

команды 40

приоритет 37

строковые 39

П

Панель управления сайтом 501

Первичный индекс 179

Переменная 28

инициализация 28

логическая (булева) 31

проверка существования 32

типы 30

уничтожение 32

функции проверки типа 30

Переменные окружения 51

Поле 179

Полиморфизм 263

Права доступа 122

Пространства имен 275

Псевдосимволы 355, 359

Р

Разработка CMS 379

Регулярные выражения 355

Рекурсия:

непрямая 169

прямая 169

С

Свойство 264

Сессии 74, 278

Система:

Eclipse 301

Система автоматического тестирования

SimpleTest 300

Системы контроля версий 291

Сниффер 509

Создание меню 393

Сокет 157

Спам 420

Список 90

областей 470

Ссылки 33

 символические 34

СУБД 177

Суперглобальные массивы 84

Т

Таблица 178

Теги 24

Точка останова 300

У

Указатель:

 \$this 266

Управляющая конструкция 355

Ф

Функции для работы с файлами 158

Функция:

 array() 92

 array_pop() 103

 array_push() 103

 array_reverse() 100

 array_shift() 104

 array_slice() 106

 array_unshift() 104

 asort() 98

 basename() 134

 compact() 105

 copy() 136

 date() 112

 define() 35

 die() 22, 24, 300

 ereg() 504

 extract() 105

 feof() 128, 129

 fflush() 131

 fgetcsv() 132

 fgetss() 129

 file() 58, 125

 file_exists() 134

 file_get_contents() 58, 125, 130

 file_put_contents() 130

 fileatime() 137

 filemtime() 138

 filesize() 138

 filetype() 138

 flock() 139

 fopen() 125

 fopen() 125, 126

 fputcsv() 133

 fread() 125

 fsockopen() 158

 func_get_args() 170

 fwrite() 130

 get_adv_block() 489

 is_array() 67

 is_cc_num() 68

 is_email() 67, 165

 is_numeric() 67

 is_string() 67

 isset() 32

 join() 58

 ksort() 99

 list() 91

 md5() 475, 498

 mkdir() 140

 move_uploaded_file() 137

 mt_rand() 109

 opendir() 141

 phpinfo() 12

 preg_match() 165

 print_r() 95

 readdir() 141

 realpath() 134

 rename() 137

 scandir() 141

 serialize() 78

 session_id() 77

 session_is_registered() 77

 session_register() 77

 session_start() 76

 session_unset() 77

 setcookie() 72

 show_message() 503

 show_news() 484

 shuffle() 101

 socket_set_blocking() 163

sort() 97
strptime() 111
time() 110
tmpfile() 131
touch() 138
uasort() 101
uksort() 101
unlink() 137
unserialize() 78
unset() 32, 92
usort() 101
вложенность 168
область видимости 167
проверка доступа к файлу или
 каталогу 135
проверка типа переменной 30

Ц

Циклы 41

Ш

Шаблонизатор 230, 233, 383, 396, 410

Шаблоны 382

Э

Экземпляр класса 263