САМОУЧИТЕЛЬ

PHP 5

- Подробное описание языка PHP.
- PHP и базы данных (MySQL и SQLite).
- **Практическое использование PHP:**
  - **форум и чат для вашего сайта,**
  - **лента новостей и голосование,**
  - **почтовая рассылка на сайте,**
  - **«движок» сайта,**
  - **Интернет-магазин своими руками и т.д.**
- Профессиональные приемы PHP.
- Специальные технологии PHP 5.
- Впервые! Система PHP-Nuke и форум PHPBB.

ПРОСТО о СЛОЖНОМ

Д.Н. Колисниченко

Лучший выбор для всех, кто хочет освоить и эффективно использовать PHP.

Д.Н. Колисниченко

САМОУЧИТЕЛЬ

PHP 5

Наука и Техника, Санкт-Петербург
2004

Колисниченко Д.Н.

Самоучитель PHP 5. — СПб.: Наука и Техника, 2004. — 576 с.: ил.

Под редакцией М.В. Финкова

ISBN 5-94387-151-9

Серия «Просто о сложном»

Данная книга является превосходным учебным пособием, позволяющим быстро освоить и эффективно использовать PHP. Здесь вы найдете все, что для этого необходимо: от описания настройки PHP, основ CGI и базовых конструкций языка, до работы с сессиями, каналами, сокетами и другими специальными технологиями PHP. Особое место занимает рассмотрение взаимодействия PHP с базами данных. В книге рассмотрены, как MySQL, так и SQLite, поддерживаемая PHP 5.

При всем при этом, книга максимально ориентирована на достижение реальных практических результатов. В деталях описано, как создать на PHP свою гостевую книгу, чат, форум, почтовую рассылку на сайте, новостную ленту и даже Интернет-магазин. Приведены готовые скрипты. Отдельно рассмотрена методика создания системы автоматического управления содержимым сайта («движка»). Такая система позволит вам максимально упростить поддержку и обновление своего сайта. Кроме создания своих скриптов, приведено описание профессиональных продуктов (бесплатных и открытых): форума PHPBB и системы PHP-Nuke. Эти продукты применяются на огромном количестве сайтов и позволяют «дважды не изобретать велосипед», а использовать уже готовые решения.

Книга отличается простотой и ясностью изложения материала. Написана опытным и известным автором. Лучший выбор для начинающих изучать PHP.

9785943871511

ISBN 5-94387-151-9

Контактные телефоны издательства
(812) 567-70-25, 567-70-26
(044) 516-38-66, 559-27-40

Официальный сайт www.nit.com.ru

© Д.Н. Колисниченко
© Наука и Техника (оригинал-макет), 2004

ООО «Наука и Техника».
Лицензия №000350 от 23 декабря 1999 года.
198097, г. Санкт-Петербург, ул. Маршала Говорова, д. 29.
Подписано в печать 25.03.04. Формат 70×100 1/16.
Бумага газетная. Печать офсетная. Объем 36 п. л.
Тираж 5000 экз. Заказ № 623 .

Отпечатано с готовых диапозитивов в ФГУП ордена Трудового Красного Знамени «Техническая книга» Министерства Российской Федерации по делам печати, телерадиовещания и средств массовых коммуникаций.
190005, Санкт-Петербург, Измайловский пр., 29.

Содержание

Введение ... 14

Часть I. Готовимся к работе с PHP ... 17

Глава 1. Что нужно знать об устройстве Интернета, прежде чем приступить к изучению PHP. Основные технологии ... 18

1.1. Архитектура сети: одноранговая и клиент/сервер ... 18
1.1.1. Одноранговые сети ... 19
1.1.2. Сети клиент/сервер ... 20
1.2. Что такое протокол. Протоколы Интернета ... 23
1.3. Как осуществляется сетевое взаимодействие ... 25
1.3.1. В каком виде информация передается по сети ... 25
1.3.2. Семиуровневая модель взаимодействия открытых систем OSI ... 26
1.3.3. Уровни сетевого взаимодействия ... 28
Физический уровень (Physical Layer) ... 28
Канальный уровень (Data link Layer) ... 28
Сетевой уровень (Network Layer) ... 29
Транспортный уровень (Transport Layer) ... 30
Сеансовый уровень (Session Layer) ... 31
Представительный уровень (Presentation Layer) ... 31
Прикладной уровень (Application Layer) ... 32
1.3.4. Интернет и модель OSI ... 32
1.4. Адресация в сети Интернет ... 33
1.4.1. Вид IP-адреса и способы его назначения ... 33
1.4.2. Классы IP-сетей ... 34
1.4.3. Доменные имена, URL-адреса и их использование ... 36
1.5. Служба доменных имен DNS ... 37
1.6. World Wide Web и HTML-документы ... 40
1.7. Порты и сетевые демоны ... 40
1.8. Структура пакетов IP и TCP ... 41
1.9. Глоссарий ... 43

Глава 2. Выбор хостинга. Регистрация доменного имени ... 44

2.1. Хостинг и хостинг-провайдеры ... 44
2.2. Обзор хостинг-провайдеров ... 45
2.2.1. Платный хостинг ... 45
AvaHost.Ru — http://www.avahost.ru ... 45
AlexHost.Ru — http://www.alexhost.ru ... 46
Agava.Ru — http://www.agava.ru, http://www.holm.ru ... 47
2.2.2. Бесплатный хостинг ... 47
Бесплатный хостинг от Agava ... 47
2.3. Поиск оптимального хостинг-провайдера ... 48
2.4. На что следует обратить внимание ... 49
2.5. Регистрация доменного имени ... 50

Глава 3. Интерфейс CGI ... 51
3.1. Принципы работы CGI ... 51
3.1.1. Общие положения ... 51
3.1.2. Как происходит взаимодействие сервера с браузером ... 52
3.1.3. Зачем нужен CGI-интерфейс и какие цели он выполняет ... 52
3.1.4. Немного подробнее о CGI ... 53
3.1.5. Способы создания CGI-сценариев ... 54
3.2. Методы передачи данных GET и POST ... 55
3.2.1. Зачем и в каком виде происходит передача данных серверу и CGI-сценариям ... 55
3.2.2. Метод GET ... 57
Формирование запроса браузером ... 57
Формирование ответа сервером ... 58
3.2.3. Метод POST ... 60
3.3. Переменные окружения CGI ... 60
3.4. Заголовки HTTP ... 62
3.4.1. Описание основных заголовков HTTP ... 62
Заголовок Accept ... 63
Заголовок Content-type ... 63
Заголовок Content-length ... 63
Заголовок Cookie ... 63
Заголовок GET ... 63
Заголовок Location ... 64
Заголовок POST ... 64
Заголовок Pragma ... 64
Заголовок Server ... 64
Заголовок Referer ... 64
Заголовок User-Agent ... 64
3.4.2. Некоторые особенности ... 64
3.5. Коды ответов HTTP ... 65
3.6. Типы MIME ... 67
3.7. PHP и CGI ... 67

Глава 4. Описание, установка и настройка необходимого программного обеспечения ... 69
4.1. Выбор необходимого программного обеспечения ... 69
4.1.1. Выбор операционной системы ... 69
4.1.2. Выбор программ ... 70
4.2. Web-сервер Apache ... 71
4.2.1. Какие функции выполняет Apache (и зачем он нам сейчас нужен на клиентской машине) ... 71
4.2.2. Конфигурационные файлы сервера Apache ... 72
4.2.3. Конфигурационный файл httpd.conf и его директивы ... 72
Общие директивы ... 73
Директивы протоколирования ... 75
Директивы управления производительностью ... 75
Директивы обеспечения постоянного соединения с клиентом ... 76
Директивы создания виртуальных хостов ... 77
Директивы настройки отображения каталогов ... 77
Директивы обработки MIME-типов ... 78
Директивы для работы с многоязычными документами ... 79
Директивы перенаправления ... 80
Директивы обработки ошибок ... 81
4.2.4. Директивы управления доступом к отдельным каталогам ... 81
Блок директив Directory ... 81
Использование директивы Limit имеет следующий вид: Limit, метод ... 82

Блок директив Limit ... 82
Блок директив Location ... 84
4.2.5. Каталоги пользователей ... 84
4.2.6. Виртуальные HTTP-серверы ... 85
Виртуальные серверы с идентификацией по имени ... 86
Виртуальные серверы с идентификацией по IP-адресу ... 87
4.2.7. Пример конфигурационного файла httpd.conf ... 89
4.3. Установка Apache + PHP + MySQL под Linux (Unix) ... 105
4.3.1. Установка Apache ... 105
Установка пакетов ... 105
Конфигурирование Apache с помощью netconf ... 107
4.3.2. Установка MySQL ... 109
4.3.3. Установка PHP и настройка связки Apache + PHP + MySQL ... 113
4.3.4. Тестирование созданной конфигурации ... 117
4.4. Установка Apache + PHP + MySQL под Windows ... 117
4.4.1. Установка и настройка Apache ... 117
4.4.2. Тестирование Apache под Windows ... 119
4.4.3. Установка PHP и настройка Apache для работы с PHP ... 120
4.4.4. Установка MySQL ... 121

Часть II. Основные понятия, функции и механизмы PHP ... 123

Глава 5. Основы языка PHP ... 124

5.1. Структура программ в PHP. Первая программа ... 124
5.1.1. Стандартный вид PHP-сценария ... 124
Общие понятия и описание PHP ... 124
Пишем первую программу (скрипт) на PHP ... 126
5.1.2. PHP-сценарии в теле HTML-документов ... 130
5.1.3. Комментарии в PHP-сценариях ... 131
5.2. Переменные и типы данных ... 131
Правила задания переменных ... 132
Типы данных ... 132
Функции определения и задания типа переменных ... 133
Логические переменные и их особенности в PHP ... 134
5.3. Константы ... 135
Задание констант. Функция define() ... 135
Проверка существования констант. Функция defined() ... 136
Стандартные константы ... 136
5.4. Операции с переменными ... 137
Присвоение значений. Оператор присваивания ... 137
Проверка существования переменной ... 138
Удаление переменных ... 140
5.5. Выражения и операции ... 140
5.5.1. Понятие «выражение» в PHP ... 140
5.5.2. Операции в PHP ... 141
Арифметические операции ... 141
Битовые операции ... 141
Логические выражения ... 142
5.5.3. Строки и строковые выражения ... 144
Строки в апострофах и строки в кавычках ... 144
Строковые операции ... 144
Многострочные строковые переменные ... 145
Строки в обратных кавычках (вызов внешней программы) ... 146
5.5.4. Оператор эквивалентности (===) ... 146
5.6. Ссылки ... 147
Жесткие ссылки ... 147

Символические ссылки ... 148
5.7. Основные конструкции языка ... 148
5.7.1. Условный оператор ... 148
5.7.2. Циклы ... 150
Цикл с предусловием While ... 150
Цикл с постусловием do while ... 151
Цикл со счетчиком For ... 152
Альтернативный синтаксис ... 153
Вложенные циклы ... 153
Операторы break и continue ... 154
Цикл перебора массивов foreach ... 155
5.7.3. Инструкция switch-case (конструкция выбора) ... 156
5.8. Собираем один скрипт из нескольких отдельных скриптов ... 158
5.8.1. Инструкции require ... 158
5.8.2. Инструкции include ... 158
5.8.3. Особенности использования include и require ... 159
5.8.4. Инструкции однократного включения include_once и require_once ... 160
5.9. Обработка ошибок ... 160

Глава 6. Передача параметров сценариям ... 162
6.1. Формы в HTML-документах ... 162
6.1.1. Общие положения ... 162
6.1.2. Элементы форм ... 163
6.1.3. Тег FORM – контейнер форм ... 164
6.1.4. Тег INPUT и способы его использования ... 165
Однострочные поля ввода ... 165
Поле ввода пароля ... 165
Скрытое текстовое поле ... 166
Независимые переключатели ... 166
Зависимые переключатели ... 167
Кнопка отправки формы ... 168
Кнопка сброса ... 169
Кнопка отправки с индивидуальным рисунком ... 169
6.1.5. Ввод многострочного текста. Тег TEXTAREA ... 169
6.1.6. Списки выбора. Тег SELECT ... 170
Списки с единственным выбором ... 170
Списки множественного выбора ... 172
6.1.7. Загрузка файлов ... 173
6.2. Передача параметров с помощью формы ... 173
6.3. Механизм Cookies ... 178
6.3.1. Принцип работы механизма Cookies ... 178
6.3.2. Программирование Cookies ... 182
Установка Cookies ... 182
Чтение значений Cookies ... 183

Глава 7. Массивы и списки ... 185
7.1. Массивы и списки ... 185
Понятие массива ... 185
Понятие списка ... 186
7.2. Ассоциативные массивы ... 187
7.3. Инструкции list() ... 188
7.4. Инструкция array() и создание многомерных массивов ... 188
7.5. Операции над массивами ... 189
7.5.1. Удаление массивов ... 189

7.5.2. Слияние массивов ... 190
Стандартное слияние ... 190
Слияние с помощью функции array_merge() ... 191
7.5.3. Перебор массива ... 191
Косвенный перебор ... 191
Прямой перебор ... 192
Недостатки косвенного перебора ... 193
7.6. Сортировка массивов ... 193
Сортировка массива по значению. Функции asort() и arsort() ... 193
Сортировка массивов по ключам. Функции ksort() и krsort() ... 193
Сортировка списка ... 194
Переворачивание массива. Функция array_reverse() ... 194
Перемешивание списков. Функция shuffle() ... 195
Пользовательская сортировка массивов.
Функции uksort(), uasort() и usort() ... 195
7.7. Получение части массива ... 196
7.8. Добавление и удаление элементов массива ... 198
Добавление в конец массива ... 198
Удаление последнего элемента списка ... 198
Добавление в начало списка. Удаление первого элемента списка .. 199
Функция array_unique() ... 199
7.9. Переменные и массивы. Взаимные переходы ... 200
Упаковка переменных в массив. Функция compact() ... 200
Разупаковка (извлечение) переменных из массива.
Функция extract() ... 200
7.10. Создание диапазона чисел ... 201
7.11. Небольшой практический пример ... 202

Глава 8. Функции пользователя.
Создание и использование собственных функций ... 204
8.1. Пользовательские функции в PHP ... 204
Что такое «пользовательские функции» ... 204
Особенности пользовательских функций в PHP ... 205
8.2. Создание пользовательских функций ... 207
Синтаксис задания функций в PHP ... 207
Первая функция ... 207
8.3. Передача параметров функции ... 208
Передача значений ... 208
Передача переменных по ссылке ... 208
8.4. Параметры по умолчанию ... 210
8.5. Переменное число параметров в функции ... 212
8.6. Локальные и глобальные переменные ... 213
8.7. Статические переменные ... 216
8.8. Рекурсия ... 216
8.9. Полезные примеры ... 217
Инициализация переменных ... 217
Параметры-массивы ... 218
Возврат функцией ссылки ... 219
8.10. Условно определяемые функции ... 220

Глава 9. Функции для работы со строками ... 221
9.1. Операции над строками ... 221
Строки в PHP ... 221
Конкатенация строк ... 221
Сравнение строк ... 222

9.2. Базовые функции работы со строками ... 223
9.3. Замена в строке ... 224
9.4. Функции для работы с отдельными символами ... 226
9.5. Изменение регистра символов. Установка и настройка локали ... 226
9.6. Преобразование символов ... 227
Функции удаления пробелов ... 227
URL-кодирование и декодирование ... 228
Использование слэшей ... 229
Замена специальных символов на их HTML-эквиваленты ... 230
9.7. Преобразование кодировок ... 230
9.8. Специальные функции ... 231
Хэш-функции ... 231
Функция Serialize() ... 232

Глава 10. Полезные стандартные функции ... 234

10.1. Работа с датой и временем ... 234
10.2. Генератор случайных чисел. Вывод баннеров случайным образом ... 235
10.3. Математические функции ... 236
10.4. Преобразования систем счисления ... 237
10.5. Запуск внешних программ ... 238

Глава 11. Работа с файлами и каталогами ... 243

11.1. Общие замечания ... 243
11.2. Основные операции с файлами ... 245
11.2.1. Открытие файла ... 245
Функция fopen() и открытие файла ... 245
Конструкция or die() ... 246
11.2.2. Временные файлы и их использование ... 247
11.2.3. Чтение и запись в бинарном режиме ... 248
11.2.4. Чтение и запись текстовых файлов (построчные чтение и запись) ... 254
11.2.5. Закрытие файлов ... 254
11.2.6. Функция file(). Чтение файла в список ... 255
11.2.7. Чтение файлов в строку ... 255
11.2.8. Чтение CSV-файлов ... 257
11.3. Функции для работы с файлами ... 258
11.3.1. Функции манипулирования файлами ... 258
11.3.2. Функции для работы с именами файлов ... 259
11.3.3. Функции определения типа и параметров файла ... 260
11.3.4. Специальные функции ... 262
11.4. Права доступа ... 263
11.4.1. Права доступа в ОС Linux ... 263
11.4.2. Права доступа в Windows ... 266
11.4.3. Функции определения прав доступа к файлу ... 267
11.5. Блокирование доступа ... 267
Общие положения ... 267
Виды блокировки ... 268
Реализация блокировки в PHP ... 268
11.6. Запись данных в начало файла ... 271
11.7. Функции для работы с каталогами ... 272
11.7.1. Манипуляции с каталогами ... 272
11.7.2. Открытие, чтение и закрытие каталогов ... 273
11.8. Каналы ... 275
11.8.1. Общие положения ... 275
11.8.2. Управление каналами с помощью языка C ... 275
11.8.3. Управление каналами в PHP ... 278

Часть III. Расширенные функции PHP ... 280

Глава 12. Графические функции PHP ... 281

12.1. Основные понятия ... 281
12.1.1. Библиотека GD ... 281
12.1.2. Методика создания графического оформления сайтов в PHP ... 281
12.2. Вывод готовых изображений ... 282
12.3. Создание изображений в PHP ... 284
12.4. Графические форматы и библиотека GD ... 284
Форматы и их указание ... 284
12.5. Определение размера картинки ... 285
12.6. Графические примитивы ... 285
12.6.1. Создание (определение) цвета ... 286
Задание нового цвета в палитре рисунка ... 286
Подбор ближайшего цвета из палитры ... 286
Эффект прозрачности ... 287
12.6.2. Рисование графических примитивов в PHP ... 287
Рисование точки ... 287
Рисование линий ... 288
Рисование прямоугольников ... 288
Рисование многоугольников ... 288
Рисование эллипсов и дуг ... 288
Заполнение области ... 289
12.7. Вывод текста в изображении ... 290
Вывод строки текста ... 290
Задание собственного шрифта ... 291
Вывод строки текста в вертикальном направлении ... 291
12.8. Работа с TTF-шрифтами ... 292
12.9. Копирование участков изображения ... 295

Глава 13. PHP и базы данных MySQL ... 297

13.1. Общие положения ... 297
База данных, таблицы записи ... 297
Клиентская и серверные части MySQL ... 297
Язык SQL ... 299
13.2. Основы SQL ... 299
13.2.1. Базовая терминология ... 299
13.2.2. Поля и их типы ... 300
13.2.3. Создание таблиц ... 303
13.2.4. Добавление данных в таблицу ... 304
13.2.5. Обновление записей ... 305
13.2.6. Удаление записей ... 306
13.2.7. Выбор записей ... 306
13.2.8. Дополнительные функции MIN, MAX, AVG, SUM ... 308
13.2.9. Группировка записей ... 308
13.2.10. Сортировка записей ... 309
13.2.11. Ключи ... 309
13.2.12. Отключение от базы данных ... 310
13.2.13. Внешние ключи ... 310
13.2.14. Удаление полей и таблиц ... 312
13.3. Функции PHP для работы с MySQL ... 312
13.3.1. Соединение PHP-сценариев с таблицами MySQL ... 312
13.3.2. Выбор базы данных ... 314
13.3.3. Обработка ошибок ... 314
13.3.4. Выполнение запросов к базе данных ... 314
13.3.5. Обработка результата запроса ... 315
13.3.6. Получение информации о результате ... 317

Глава 14. Сетевые функции ... 320
14.1. Сокеты ... 320
14.1.1. Понятие сокета. Функции для работы с сокетами ... 320
Открытие и использование сокетов ... 320
Блокирующий/неблокирующий режимы. Функция socket_set_blocking ... 321
14.1.2. Отправка электронных писем с помощью сокетов ... 322
14.2. Работа с DNS ... 325
14.2.1. DNS-функции ... 325
14.2.2. Корректное разрешение IP-адреса в доменное имя ... 326

Часть IV. Создание своих проектов на PHP ... 328

Глава 15. Отправление и прием писем. Делаем почтовую рассылку на сайте ... 329
15.1. Использование функции mail() для отправки сообщений ... 329
15.2. Если сообщения не отправляются ... 331
15.3. Отправление писем в формате HTML и вложений ... 331
15.3.1. Общие положения ... 331
15.3.2. Использование класса HtmlMimeMail ... 332
15.3.3. Сводный алгоритм отправки сообщений в различных MIME-форматах ... 335
15.3.4. Автоматическая отправка копий сообщений ... 336
15.4. Прием писем по протоколу POP3 ... 336
15.4.1. Общие сведения о механизме приема писем и протоколе POP3 ... 336
15.4.2. Использование класса POP3 ... 337
15.4.3. Пишем сценарий получения сообщений ... 338

Глава 16. Делаем чат для своего сайта ... 345
16.1. Принцип создания чата ... 345
16.2. Простой чат – SimpleChat ... 346
16.2.1. Установка, настройка и работа с чатом ... 347
16.2.2. Проблемы при установке ... 350
Некорректная работа ... 350
Проблемы с отображением даты и времени ... 350
Я не могу очистить чат! ... 350
16.2.3. Полезные советы ... 351
16.3. Профессиональный чат CHAT.PHP.SPB.RU ... 352
Установка и настройка чата под Windows ... 353
Установка и настройка чата под Linux ... 355

Глава 17. Голосование на сайте ... 358
17.1. Делаем голосование сами ... 358
17.2. Используем профессиональный бесплатный скрипт Advanced Poll ... 361
Общее описание и установка Advanced Poll ... 361
Настройка Advanced Poll. Центр управления ... 362
Создание голосования ... 365
Редактирование голосования ... 366
Использование шаблонов ... 368
Несколько голосований на одной Web-странице ... 369

Глава 18. Лента новостей ... 371
18.1. Принцип организации новостной ленты ... 371
18.2. Используем профессиональный скрипт (FusionPHP) ... 371
18.2.1. Устанавливаем скрипт ... 371
18.2.2. Работа со скриптом ... 377

Глава 19. Гостевая книга и Форум ... 380
19.1. Создаем сами гостевую книгу ... 380
19.2. Используем готовый скрипт гостевой книги ... 385
19.2.1. Установка гостевой книги ... 385
19.2.2. Работа с гостевой книгой ... 387
19.3. Форум PHPBB ... 390
19.3.1. Установка форума PHPBB ... 390
19.3.2. Изменение внешнего вида форума ... 395
19.3.3. Администрирование форума ... 395

Глава 20. Счетчик и анализ серверной статистики ... 399
20.1. Счетчик и протоколирование посещений на сайте ... 399
20.1.1. Простой счетчик ... 399
Разрабатываем механизм работы счетчика ... 399
Пишем сценарий счетчика ... 400
«Отладка» счетчика ... 401
20.1.2. Создание графического счетчика ... 402
20.1.3. Протоколирование посещений ... 403
Методика протоколирования ... 403
Реализация протоколирования ... 404
20.1.4. Разработка системы статистики ... 407
20.2. Анализ серверной статистики (кто откуда приходил и что делал). Программа Webalizer ... 412
20.2.1. Методика сбора и анализа серверной статистики. Серверные журналы ... 412
20.2.2. Установка и запуск Webalizer ... 414
20.2.3. Конфигурационные параметры Webalizer ... 417

Глава 21. Сессии в PHP ... 419
21.1. Обзор механизма сессий ... 419
21.1.1. Особенности CGI и недостатки Cookies ... 419
21.1.2. Механизм сессий: принцип работы ... 420
21.2. Инициализация сессии и регистрация в сессии переменных ... 422
21.3. Имя сессии ... 423
21.4. Еще один способ (устаревший) обойти использование Cookies в сессиях ... 424
21.5. Обработчики сессии ... 425
21.6. Сохранение в сессии ассоциативного массива ... 428

Глава 22. Работа с WWW ... 430
22.1. Функция Header() ... 430
22.2. Запрет кэширования ... 430
22.3. SSI и PHP. Включение одних сценариев и html-страниц в другие ... 431
22.3.1. Общие положения ... 431
22.3.2. SSI и PHP. Функция virtual ... 432
22.4. Загрузка файла на сервер ... 433
Multipart-формы и их описание ... 433
Сценарий загрузки файлов на сервер. Обработка multipart-форм ... 434
Организация виртуальной фотогалереи с возможностью самостоятельного размещения в ней фотографий пользователей ... 436

Глава 23. Создание и использование шаблонов. Модули ... 441
23.1. Разделение проекта на дизайнерскую часть и программирование ... 441
23.2. Использование модулей. Создание библиотекаря ... 442
23.2.1. Общие положения ... 442
23.2.2. Создание библиотекаря ... 443

23.2.3. Использование библиотекаря. Подключение модулей ... 446
23.3. Использование шаблонов ... 447

Глава 24. Система PHP-Nuke — система управления содержимым сайта (контентом). Система PHP-Nuke ... 454

24.1.Возможности и особенности PHP-Nuke ... 454
24.2. Установка PHP-Nuke ... 455
24.3. Возможности администрирования. Настройка сайта ... 457
24.4. Управление контентом ... 459
24.5. Различные настройки ... 463

Часть V. PHP для профессионалов ... 467

Глава 25. Объектно-ориентированное программирование и PHP ... 468

25.1. Основные принципы и понятия объектно-ориентированного программирования ... 468
Введение ... 468
Инкапсуляция (encapsulation) ... 469
Полиморфизм (polymorphism) ... 470
Наследование (inheritance) ... 470
25.2. Создание класса в PHP ... 470
25.3. Доступ к методам и свойствам объекта ... 471
25.4. Инициализация объектов ... 472
25.5. Еще раз о наследовании и полиморфизме ... 473
25.6. Пример класса ... 475
25.7. ООП и PHP 5 ... 477

Глава 26. Тонкая настройка интерпретатора PHP ... 479

26.1. Директивы файла конфигурации php.ini ... 479
Общий формат файла php.ini ... 479
Настройка обработки ошибок. Директива error_reporting ... 480
Максимальное время работы сценария. Директива max_execution_time ... 480
«Вредная» директива magic_quotes_gpc ... 480
Автоматическое помещение передаваемых сценарию переменных в глобальные массивы ... 481
26.2. Индивидуальная настройка обработки ошибок для конкретного сценария ... 481
26.3. Синтаксическая проверка выполнения кода ... 482
Проверка без запуска кода ... 482
Проверка с запуском кода ... 483
26.4. Финализаторы. Функция Register_shutdown_function() ... 484
26.5. Информационные функции ... 485
Функция phpinfo() ... 485
Функция phpversion() ... 486
Функция getlastmod() ... 486
26.6. «Засыпание» сценария ... 486

Глава 27. Защищенные соединения ... 487

27.1. Настройка SSL-соединения ... 487
27.2. Генерирование сертификатов ... 490
27.3. Авторизация и переменные среды окружения ... 491

Глава 28. Два проекта в одной главе ... 494

28.1. Пишем свой Интернет-магазин ... 494
28.1.1. Вводная часть ... 494

28.1.2. Структура Интернет-магазина ... 495
28.1.3. Пишем вспомогательные скрипты ... 497
28.1.4. Пишем скрипт Интернет-магазина ... 500
21.1.5. Тестируем Интернет-магазин ... 511
28.2. Дистанционное обучение – система проверки знаний ... 513
28.2.1. Общие положения ... 513
28.2.2. Параметры системы проверки знаний ... 515
28.2.3. Организация. Выработка структуры ... 515
28.2.4. Пишем код системы проверки знаний ... 519

Глава 29. Автоматизированное управление параметрами хостинга. Система Monster Controls ... 528

29.1. Общие положения. Регистрация ... 528
29.2. Использование Monster Controls ... 529
Основное меню. Статистика ... 529
Раздел Email options ... 531
Раздел Site Maintenance ... 531
Раздел Webmaster tools ... 534
29.3. Делаем мемберскую зону ... 535

Глава 30. Поддержка COM и Java ... 537

30.1. Технология COM ... 537
30.2. Поддержка Java ... 540

Часть VI. Специальные технологии PHP 5 ... 543

Глава 31. Особенности и специальные возможности PHP 5 ... 544

31.1. Обзор нововведений ... 544
Общая информация ... 544
Новые директивы настроечного файла php.ini ... 544
Новые механизмы поддержки потоков ... 545
Новые функции графической библиотеки GD ... 545
Разные новые функции ... 545
31.2. Zend Engine 2.0 ... 547
31.2.1. Новая объектная модель и объектные дескрипторы ... 547
31.2.2. Механизм возврата функциями объектов ... 548
31.2.3. Разыменование объектов ... 549
31.2.4. Генная инженерия (клонирование объектов) ... 550
31.2.5. Оператор delete ... 550
31.2.6. Конструкторы и иерархия классов ... 551
31.2.7. Дополнительные возможности ... 552

Глава 32. SQLite ... 554

32.1. Введение в SQLite ... 554
32.1.1. Установка ... 554
32.1.2. Встроенные константы SQLite ... 555
32.1.3. Список функций SQLite ... 555
32.2. Открытие и закрытие базы данных ... 556
32.3. Выполнение запросов ... 557
32.4. Функции для работы с результатом ... 557
32.5. Информационные функции ... 559
32.6. Другие функции ... 559

Вместо заключения: Горячая десятка ошибок ... 560

Предметный указатель ... 563

Введение

В последнее время многие компании просто не представляют свою работу без Internet: электронная почта, FTP и, конечно же, WWW. Всемирная паутина позволяет оперативно узнать всю необходимую информацию о деятельности компании. Ясное дело, что эта информация должна также оперативно обновляться. Не будем говорить о высоких материях, а возьмем, к примеру, тот же интерактивный прайс-лист: кому интересно читать ваш прейскурант, если он уже месяц не обновлялся? Правильно, никому.

Даже если вы — простой интернет-пользователь, а не президент какой-то корпорации, вам наверняка хочется приукрасить свой сайт, разместив на нем небольшой форум, гостевую книгу или новостную ленту.

Интерактивный прайс-лист, форум, Интернет-магазин, гостевая книга, лента новостей — для создания всего этого вам просто не обойтись без PHP. Конечно, и форум, и гостевую книгу можно написать с помощью других средств, например, на языке Perl, но PHP позволяет вам написать Internet-приложение любой сложности с минимальными затратами: PHP, как и все вспомогательное программное обеспечение, за исключением Windows, бесплатно, а разработка самого приложения не отнимет у вас много времени. Иногда разработка приложения занимает меньше времени, чем разработка дизайна.

Данная книга будет полезной, как начинающим Web-программистам, так и опытным. Только последние будут ее читать не «от корки до корки», а выборочно — на свое усмотрение.

Если вы — начинающий Web-программист, вам нужно читать книгу последовательно — главу за главой. **Первая глава** содержит основные сведения об устройстве Internet: без знания этого материала вам даже не стоит пробовать читать эту книгу. Даже если вы — опытный пользователь Internet, все равно прочитайте эту главу — вы найдете в ней много нового для себя.

Рано или поздно перед вами предстанет проблема выбора хостинг-провайдера. Выбрать «хостера» вам поможет **вторая глава**. В ней изложены общие рекомендации относительно выбора хостинга, а также описание услуг популярных в России хостинг-провайдеров.

В **третьей главе** описан интерфейс общего шлюза — CGI. Многие Internet-приложения используют этот интерфейс, поэтому знать хотя бы что это

такое просто необходимо, тем более, в этой главе вы узнаете об интересных возможностях, предоставляемых интерфейсом CGI, например, как определить браузер пользователя и удаленный IP-адрес.

Четвертая глава вам просто необходима для начала работы с PHP: в ней описывается установка и настройка необходимого программного обеспечения: Web-сервера Apache, сервера баз данных MySQL и, собственно, самого PHP. При написании книги я старался не привязываться к конкретной операционной системе — ни к Linux, ни к Windows. Ясное дело, что установка программного обеспечения под Linux и под Windows существенно отличается, поэтому в четвертой главе сначала описывается установка программ под Linux, а потом — под Windows. Точно также и в остальных главах этой книги: если работа какой-нибудь функции или программы под Linux отличается от работы аналогичной программы или функции для Windows, вы сразу же найдете соответствующий комментарий.

В **пятой главе** мы приступим к изучению самого PHP: в этой главе описаны структура программы, синтаксис PHP, константы, переменные, выражения и основные конструкции языка.

Любой программе, в том числе и PHP-программе, пользователь может передать параметры, обработав которые, программа отобразит соответствующий заданным параметрам результат. Для передачи параметров PHP-программе используются формы, поэтому данной теме посвящена целая глава — **шестая**, а не небольшой пункт, как бывает в случае с другими языками программирования.

При программировании на PHP вы очень часто, поверьте моему опыту, будете сталкиваться с массивами и списками. Чем отличаются массивы от списков и как с ними работать, вы узнаете в **седьмой главе**.

Ни одна серьезная программа не в состоянии обойтись без использования подпрограмм. В **восьмой главе** рассматривается создание собственных подпрограмм — функций.

Самыми универсальными объектами в PHP являются строки: в этом вы убедитесь, прочитав **девятую главу**.

В **десятой главе** рассматриваются полезные стандартные функции PHP, которые вы будете использовать при создании своих сценариев.

Как работать с файлами и каталогами, вы узнаете в **одиннадцатой главе**. В этой же главе подробно рассматриваются права доступа к файлам и каталогам в Linux, а также каналы.

Библиотека GD позволяет создавать на PHP различные специальные изображения: изображения для счетчиков, рекламные баннеры, логотипы. Обо всем этом вы прочитаете в **двенадцатой главе**.

Ни один серьезный Internet-проект не обходится без баз данных. **Сервер MySQL** является одним из самых удачных реализаций SQL-серверов,

поэтому не рассмотреть сервер баз данных, установленный на компьютерах большинства хостинг-провайдеров, просто невозможно. Как написать сценарий, работающий с MySQL, вы узнаете в **главе 13**.

Любое Internet-приложение по определению является сетевым. Сетевые функции PHP подробно рассмотрены в **главе 14**.

В **четвертой части** этой книги мы будем создавать свои собственные PHP-проекты:

- Рассылка почты (**глава 15**).
- Чат (**глава 16**).
- Система голосования (**глава 17**).
- Новостная лента (**глава 18**).
- Гостевая книга и форум (**глава 19**). В том числе рассмотрено использование и настройка профессионального форума PHPBB. Использование готового скрипта позволит вам максимально быстро и эффективно сделать к своему сайту отличный форум.
- Счетчик (**глава 20**).

В **двадцать первой** главе рассмотрен механизм сессий, который поможет вам при разработке сложных сценариев.

В следующей, **двадцать второй, главе** рассмотрены функции PHP для работы со всемирной паутиной, то есть с WWW.

В **двадцать третьей** главе будут рассмотрены приемы созданий больших сценариев, для создания которых нам не обойтись без библиотекаря и шаблонов страниц.

В **главе 24** рассмотрено средство управления контентом — PHP-Nuke. Эта система позволит вам с минимальными знаниями PHP и с минимальными затратами времени профессионально построить и автоматизировать свой сайт.

Пятая часть этой книги предназначена для профессиональных PHP-программистов, в ней рассматривается:

- Объектно-ориентированное программирование на PHP 4 и PHP 5.
- «Тонкая» настройка интерпретатора PHP.
- Защищенные соединения (SSL).
- Интернет-магазин.
- Система проверки знаний.
- Автоматизированное управление параметрами хостинга.
- Поддержка COM и Java.

Специальные технологии PHP, например, возможности Zend Engine 2.0, рассматриваются в последней части этой книги. Завершает книгу горячая десятка ошибок, которые часто совершают начинающие программисты.

САМОУЧИТЕЛЬ PHP 5

Часть I

Готовимся к работе с PHP

Глава 1. Что нужно знать об устройстве Интернета, прежде чем приступить к изучению PHP. Основные технологии

Глава 2. Выбор хостинга. Регистрация доменного имени

Глава 3. Интерфейс CGI

Глава 4. Описание, установка и настройка необходимого программного обеспечения

Глава 1

Что нужно знать об устройстве Интернета, прежде чем приступить к изучению PHP. Основные технологии

Прежде чем приступить к созданию Web-сайтов и Web-порталов, необходимо иметь хотя бы общее представление о том, как устроен и функционирует Интернет. Без этой информации вы не сможете изучить язык PHP, так как его технологии опираются на основные технологии Интернета и тесно взаимосвязаны с ними. Тем не менее, базовый набор знаний об устройстве и принципах Интернета не так уж велик. Вполне возможно, что вы уже частично или полностью обладаете этой информацией. Поэтому вы можете пропускать знакомые разделы в данной главе.

1.1. Архитектура сети: одноранговая и клиент/сервер

Начнем с самого главного — архитектуры сети. Существуют две основные архитектуры сети: **одноранговая** (peer-to-peer) и **клиент/сервер** (client/server), причем вторая практически вытеснила первую.

В одноранговой сети все компьютеры равноправны — имеют один ранг. Поэтому любой компьютер может выступать как в роли сервера, то есть предоставлять свои ресурсы (файлы, принтеры) другому компьютеру, так и в роли клиента — использовать предоставленные ему ресурсы.

В отличие от одноранговой сети, в сети клиент/сервер существует один или несколько главных компьютеров — серверов. Все остальные компьютеры сети называются клиентами или рабочими станциями (workstations). Сервер — это специальный компьютер, который предоставляет определенные услуги другим компьютерам. Существуют различные виды серверов (в зависимости от предоставляемых ими услуг): серверы баз данных, файловые серверы, серверы печати (принт-серверы), почтовые серверы, Web-серверы, и т.д.

1.1.1. Одноранговые сети

Одноранговые сети преимущественно распространены в домашних сетях или небольших офисах. В самом простом случае для организации такой сети нужно всего лишь пара компьютеров, снабженных сетевыми платами, и коаксиальный кабель. Нужна еще пара терминаторов (заглушек), но это уже детали.

Сначала сеть создается физически, то есть компьютеры соединяются друг с другом посредством коаксиального кабеля. После того, как это будет сделано — нужно настроить сеть программно. Для этого необходимо, чтобы на компьютерах были установлены сетевые операционные системы (Linux, FreeBSD, Windows NT/2000/XP, Windows 98/Me) или сетевые системы с поддержкой сетевых функций (Windows 95, Windows for Workgroups). Программные настройки сети выполняются средствами выбранной операционной системы.

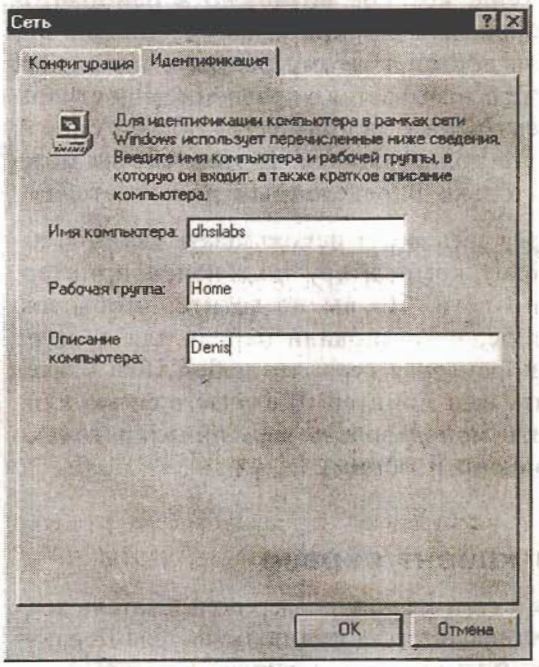


Рис. 1.1. *Идентификатор рабочей группы в ОС Windows 9x*

Компьютеры в одноранговой сети объединяются в **рабочие группы**. Каждая рабочая группа имеет свой идентификатор — имя рабочей группы. Если вы сейчас работаете в Windows 9x, узнать имя рабочей группы вы можете, запустив апплет **Сеть** с **Панели управления** (см. рис. 1.1).

Для примера допустим, что в вашей одноранговой сети три компьютера A, B, C. Первые два входят в рабочую группу WG1, а компьютер C — в рабочую группу WG2 (см. рис. 1.2).

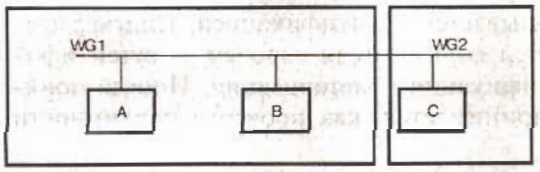


Рис. 1.2. *Схема одноранговой сети*

Даже несмотря на то, что компьютеры входят в один сегмент сети (физически подключены к одному кабелю), компьютеры A и B не будут «видеть» компьютер C, а компьютер C не будет видеть компьютеры A и B.

Если выполнить команду поиска компьютера в Windows 9x (**Пуск ⇒ Поиск ⇒ Найти компьютер**), компьютер «увидит» компьютеры A и B, но будет сообщено, что они находятся в другой рабочей группе — WG1.

Поскольку в сети работает как минимум несколько пользователей, необходимо управлять их доступом к различным ресурсам сети. И действительно, в сети может содержаться конфиденциальная информация, доступ к которой необходимо запретить для большинства пользователей. Кроме того, к сети могут быть подключены сетевые устройства, например, сетевой принтер. К нему также обычно разграничивается доступ, чтобы все подряд не могли на нем печатать из сети.

Единственное ограничение доступа, которое возможно в одноранговой сети, это использование пароля для доступа к какому-нибудь ресурсу. Для того, чтобы получить доступ к этому ресурсу, например, принтеру, нужно знать пароль. Это называется управлением доступом на уровне ресурсов. В сети клиент/сервер используется другой способ управления доступом — на уровне пользователей. В этом случае можно разрешить доступ к ресурсу только определенным пользователям.

Например, ваш компьютер A через сеть могут использовать два пользователя: Иванов и Петров. К этому компьютеру подключен принтер, который можно использовать по сети. Но вы не хотите, чтобы кто угодно печатал на вашем принтере, и установили пароль для доступа к этому ресурсу. Если у вас одноранговая сеть, то любой, кто узнает этот пароль, сможет использовать ваш принтер. В случае с сетью клиент/сервер вы можете разрешить использовать ваш принтер только Иванову или только Петрову (можно и обоим).

1.1.2. Сети клиент/сервер

Для получения доступа к ресурсу в сети клиент/сервер пользователь должен ввести свой уникальный идентификатор — имя пользователя (login — логин) и пароль (password). Логин пользователя является общедоступной информацией, и это правильно. Если кто-нибудь захочет отправить пользователю сообщение по электронной почте, то для этого ему достаточно знать его логин (естественно, и имя сервера электронной почты, который «знает» этого пользователя).

Проверка имени пользователя называется идентификацией. Подтверждение (проверка подлинности) имени пользователя паролем — аутентификация. Идентификация + аутентификация = авторизация. Иногда понятие аутентификация просто воспринимается, как проверка подлинности в широком смысле этого слова.

После рассмотрения архитектуры одноранговой сети можно придти к выводу, что единственное ее преимущество — это ее простота и деше-

визна. Сети клиент/сервер обеспечивают более высокий уровень производительности и безопасности.

В отличие от одноранговой сети, в сети клиент/сервер существует один или несколько главных компьютеров — серверов. Все остальные компьютеры сети называются клиентами или рабочими станциями (workstations). Как я уже писал выше, *сервер* — это специальный компьютер, который предоставляет определенные услуги другим компьютерам. Существуют различные виды серверов (в зависимости от предоставляемых ими услуг): серверы баз данных, файловые серверы, серверы печати (принт-серверы), почтовые серверы, Web-серверы, и т.д.

Для экономии средств, как правило, один сервер сочетает в себе несколько функций, например, почтовик может быть также и Web-сервером. Услуги, которые может предоставлять сервер, ограничиваются только его физическими возможностями — чем мощнее сервер, тем больше услуг и с большим качеством он может предоставлять, поэтому в качестве сервера выбирается довольно мощный компьютер.

Сеть клиент/сервер более дорога в построении и обслуживании, чем простая одноранговая сеть. Так почему же Internet построен с использованием технологии клиент/сервер? Неужели никто не задумывался над экономией средств?

Все дело в том, что Internet просто-напросто не может быть одноранговой сетью. Сейчас объясню почему. Можно привести множество примеров, но мы ограничимся только двумя. По определению все компьютеры в одноранговой сети равны между собой. Теперь представьте себе такую одноранговую сеть, в которой сто-двести тысяч компьютеров (а в Internet узлов намного больше!). Ясное дело, что вы не можете запомнить IP-адреса всех компьютеров, вам легче запомнить их символьное имя — `comp_vasya`, `comp_igor` и т.п. Вы вводите в браузере адрес `http://comp_vasya`.

Компьютер, в отличие от человека, «привык» оперировать с числами, а не символьными именами. То есть ему нужна программа, которая преобразует имя `comp_vasya` в IP-адрес. Написать такую программу очень просто, но сложность заключается в том, откуда эта программа будет брать сведения про имена компьютеров и соответствующие им IP-адреса?

Раньше, когда Internet не было, а была сеть ARPAnet проблема разрешения имени решалась так: администратор редактировал файл `hosts`, в который вносил имя компьютера и IP-адрес. Потом этот файл нужно было скопировать на каждый (!) компьютер сети (а у нас их 200 тысяч). Программа преобразования имени просматривала этот файл и, если ей удавалось найти нужный IP-адрес, возвращала его. Ясное дело, что такая организация доставляла массу неудобств, как администраторам, так и пользователям — забыл обновить файл, ты уже не получишь доступ к компьютеру, не зная его IP. Поэтому было принято решение разрабо-

тать службу доменных имен — DNS (Domain Name System), которая и занималась решением данной проблемы.

А служба DNS работает примерно так: к DNS-серверу обращается компьютер с просьбой преобразовать символьное имя в IP-адрес. Если DNS-сервер может разрешить имя компьютера, он возвращает его IP-адрес. Все это делается незаметно для пользователя — ему нужно ввести только имя компьютера, а все остально его не касается. Администратору тоже удобно: ему нужно всего один раз отредактировать базу данных сервера DNS. Как видите, служба DNS подразумевает использование технологии клиент/сервер — тут уж никак не отвертишься.

Второй пример связан с авторизацией. Представьте, что у нас есть те самые 200 тысяч пользователей. Один из них входит в сеть и пытается обратиться к компьютеру `comp_igor`. Ясное дело, что этот компьютер должен его авторизировать, то есть определить, кто вы такой. Для этого на компьютере `comp_igor` должна быть база данных, содержащая все имена пользователей и пароли. Такая база данных должна быть на каждом из 200 тысяч компьютеров, а иначе сеть не сможет работать!

Намного проще для авторизации использовать один какой-нибудь сервер (или несколько таких серверов). Во-первых, удобно — учетную запись пользователя нужно создать только на одном компьютере, а во-вторых, так намного безопаснее: ведь среди 200 тысяч компьютеров найдется как минимум десятка два-три незащищенных, которые проще взломать. При этом, учитывая, что каждый компьютер содержит все имена пользователей и все пароли, в руках злоумышленников окажется вся сеть. О последствиях такого взлома, думаю, не трудно догадаться.

Помните, я говорил про экономию средств? Давайте посчитаем затраты на содержание двух баз данных — имен компьютеров и пользователей для 200 тысяч компьютеров. Договоримся, что максимально допустимое имя компьютера может состоять из 32 символов, имя пользователя — из 16, а пароль — из 8.

32 (байта — имя компьютера) x 200000 (число записей в файле) x 200000 (этот файл должен быть на каждом компьютере) = 1280000000000 байт = 1220703,125 Мб дискового пространства;

16 + 8 + 1 (имя, пароль и разделитель) x 200000 x 200000 = 1000000000000 байт = 953674,32 Мб;

Итого: 2174377,445 Мб.

Представьте, что каждый день по сети будет «гулять» 2174377,445 Мб информации! Такое даже в страшном сне не приснится. Это ж какая пропускная способность должна быть!

При использовании сети клиент/сервер затраты будут ровно в 200000 раз меньше — чуть больше 10 Мб.

1.2. Что такое протокол. Протоколы Интернета

Теперь пора уже перейти к протоколам, в частности, к протоколу TCP/IP, который лежит в основе сети Интернет. **Протокол — это совокупность правил, определяющих взаимодействие абонентов вычислительной системы (в нашем случае — сети) и описывающих способ выполнения определенного класса функций.** Говоря простым языком, протокол — это набор правил, по которым взаимодействуют компьютеры между собой.

Необходимость протоколов обусловлена тем, что в сети могут взаимодействовать компьютеры с самым разным программным обеспечением (операционными системами) и самым разным аппаратным устройством. Чтобы все подключенные к сети компьютеры могли понимать друг друга, необходимы общие наборы правил. Такими наборами правил и являются протоколы. Для разного рода взаимодействий используются разные правила, а значит и разные протоколы. Давайте рассмотрим основные протоколы, используемые в Интернете.

Протокол TCP/IP. Самым главным — святыней всех святынь — является **протокол TCP/IP** (Transmission Control Protocol/Internet Protocol — Протокол Управления Передачей/Интернет-Протокол). Именно на этом протоколе основана вся сеть Интернет.

Если быть точными, TCP/IP — это не один, а два протокола:

- Протокол TCP — является транспортным протоколом, который обеспечивает гарантированную передачу данных по сети.
- Протокол IP — является адресным протоколом, который отвечает за адресацию всей сети. То есть, благодаря использованию протокола IP, каждый компьютер (устройство) в сети имеет свой индивидуальный адрес (IP-адрес). По этим адресам и осуществляется передача данных. Широко используемые в Интернете URL-адреса (www.rambler.ru, www.ozon.ru, и т.п.) являются лишь словесными обозначениями IP-адресов. Сделано это для удобства, поскольку человеку проще запомнить словесный адрес, нежели числовой IP-адрес. Однако компьютеры работают только с числовыми адресами. Когда вы вводите URL-адрес в командную строку браузера, то он автоматически преобразуется в IP-адрес. Обмен данными в Интернете осуществляется только по IP-адресам.

За сопоставление словесных URL-адресов и числовых IP-адресов отвечает специальная служба — служба DNS. И ошибка в этом деле может привести к тому, что вы будете обращаться к одному серверу, а попадать на другой или вообще никуда не попадать, а наблюдать у себя на экране сообщение об ошибке. Подробнее на рассмотрении протокола TCP/IP и службы DNS мы остановимся чуть позже, когда будем изучать передачу данных в сети Интернет. Сейчас же продолжим рассмотрение других немаловажных протоколов.

Протокол ICMP (Internet Control Message Protocol) — протокол межсетевых управляющих сообщений. С помощью этого протокола компьютеры и устройства в сети обмениваются друг с другом управляющей информацией. К примеру этот протокол используется для передачи сообщений об ошибках, проверки доступности узла, и т.д.

Протокол FTP (File Transfer Protocol) — протокол передачи файлов. Служит для обмена файлами между компьютерами. Например, вам нужно передать файл на сервер или, наоборот, скачать файл с сервера. Для этого вам нужно подключиться к файловому серверу (он же FTP-сервер) и выполнить необходимую вам операцию скачивания или закачки. Подключение к FTP-серверу обычно осуществляется с помощью FTP-клиента. Простейший FTP-клиент входит в состав практически любой операционной системы. Кстати, просматривать FTP-сервера могут и обычные браузеры.

Протокол HTTP (Hyper Text Transfer Protocol) — протокол обмена гипертекстовой информацией, то есть документами HTML. Вы наверное слышали, что HTML является базовым языком создания Web-страниц. Так вот, протокол HTTP предназначен для их передачи в сети. Таким образом, протокол HTTP используется Web-серверами. Соответственно, браузеры, используемые для блуждания по Интернету, являются HTTP-клиентами.

Протоколы POP и SMTP. Протокол **POP** (Post Office Protocol) — протокол почтового отделения. Этот протокол используется для получения электронной почты с почтовых серверов. А для передачи электронной почты служит протокол **SMTP** (Simple Mail Transfer Protocol) — протокол передачи сообщений электронной почты.

Протокол IMAP. Для чтения почты существует и другой протокол — **IMAP**. Его отличие от протокола POP состоит в том, что пользователь читает сообщения электронной почты, не загружая их на свой компьютер. Все сообщения хранятся на сервере. При удалении сообщения оно удаляется с сервера.

Протокол SLIP (Serial Line Internet Protocol) — протокол подключения к сети Интернет по последовательной линии. Используется для установления связи с удаленными узлами через низкоскоростные последовательные интерфейсы. В настоящее время вытеснен протоколом PPP и практически не используется.

Протокол PPP (Point-to-Point Protocol) — обеспечивает управление конфигурацией, обнаружение ошибок и повышенную безопасность при передаче данных на более высоком уровне, чем протокол SLIP. Поэтому при настройке сервера рекомендуется использовать именно этот протокол. Протокол PPP рассмотрен в RFC 1547 и RFC 1661.

Протокол RIP (Routing Information Protocol) — используется для маршрутизации пакетов в компьютерных сетях. Для маршрутизации также используется протокол OSPF (Open Shortest Path First), который является более эффективным, чем RIP.

1.3. Как осуществляется сетевое взаимодействие

1.3.1. В каком виде информация передается по сети

Любая сеть связи работает с использованием определенного метода коммутации абонентов. Наиболее распространены три метода коммутации:

- коммутация каналов;
- коммутация пакетов;
- коммутация сообщений.

Для использования метода коммутации каналов нужен физический канал для прямой передачи данных между узлами. Физический канал состоит из последовательно соединенных канальных участков. Участки соединяются между собой коммутаторами, которые устанавливаются между конечными узлами сети.

Коммутация каналов удобна для телефонных сетей. Для передачи компьютерного трафика был разработан метод коммутации пакетов. Для передачи компьютерных данных сети с коммутацией каналов совершенно не годятся из-за слишком большого коэффициента пульсации трафика. Рассмотрим следующую ситуацию: вы заходите на почтовый сервер и запрашиваете список сообщений. Сервер, в ответ на ваш запрос, порождает относительно небольшой трафик — ведь список содержит небольшой объем информации — кто отправил, дата, тема, размер сообщения.

Просмотрев список сообщений, вы загружаете на свой компьютер сообщение, отправленное вашим другом. Он написал про летний отдых и прислал десяток качественных фотографий общим размером в 10 Мб. Получение этого сообщения порождает интенсивный обмен данных. Просмотрев фотографии, вы, в свою очередь, отправляете ему свои фото, что также создает большую нагрузку на канал связи. Если сеть, в которой вы работаете, использует метод коммутации каналов, то канал между вами и сервером, большую часть времени будет простаивать, занимая и без того ограниченные ресурсы коммутатора.

При коммутации пакетов ваши данные будут разбиты на части — пакеты. Вы можете передавать данные без ограничения на их размер — хоть 100 Гб, но размер пакетов ограничен — от 46 до 1500 байт. Пакеты передаются, как независимые блоки. На другом компьютере переданные вами данные будут «собраны» воедино. Пакетная передача позволяет сбалансировать нагрузку на канал связи и обеспечить наиболее эффективное его использование.

Мы забыли о сетях с коммутацией сообщений. В чистом виде они сегодня не существуют, поскольку в свое время они послужили прототипом сетей с коммутацией пакетов.

Из чего состоит пакет? Пакет состоит из заголовка и поля данных. Заголовок содержит служебную информацию — адрес отправителя, адрес получателя, порт назначения и другая информация. Поле данных содержит передаваемые вами данные. Ясное дело, что не все сразу, а только их часть. Любая сеть накладывает свои ограничения на длину пакетов, например, для сети Ethernet минимальный размер пакета равен 72 байтам, а максимальный — 1526. Минимальный размер обусловлен необходимой служебной информацией, если мы ее не укажем — мы не сможем отправить даже пустой пакет. Мы еще вернемся к устройству пакетов и рассмотрим заголовок пакета подробнее.

1.3.2. Семиуровневая модель взаимодействия открытых систем OSI

В начале 80-х годов международной организацией по стандартизации (ISO — International Organization for Standardization) была разработана **модель взаимодействия открытых систем** (OSI — Open System Interconnection). В другой литературе вы можете встретить и другие названия этой модели: сокращенное — модель OSI или более полное — семиуровневая модель взаимодействия открытых систем OSI.

Средства взаимодействия (см. рис. 1.3) в модели OSI делятся на семь уровней, каждый из которых призван решать свой круг задач:

1. Физический;
2. Канальный;
3. Сетевой;
4. Транспортный;
5. Сеансовый;
6. Представительный;
7. Прикладной.

Благодаря разделению на уровни задача сетевого взаимодействия разбивается на несколько более мелких задач. Это позволяет при разработке новых способов и инструментов сетевого взаимодействия не создавать их заново целиком и полностью, а использовать уже готовые решения, заменив только некоторые его части.

Непосредственно друг с другом компьютеры (открытые системы) взаимодействуют только на физическом уровне. Все остальные уровни подготавливают данные для передачи или контролируют саму передачу данных. Напрямую эти уровни взаимодействуют только с выше- и нижележащими уровнями: пользуются услугами нижележащего и предоставляют услуги вышележащему. Друг с другом такие уровни контактируют косвенным образом, через посредство нижележащих уровней.

Примечание.

В некоторых случаях сетевого взаимодействия физический уровень, как таковой, отсутствует, при этом его функции выполняет самый нижележащий уровень.

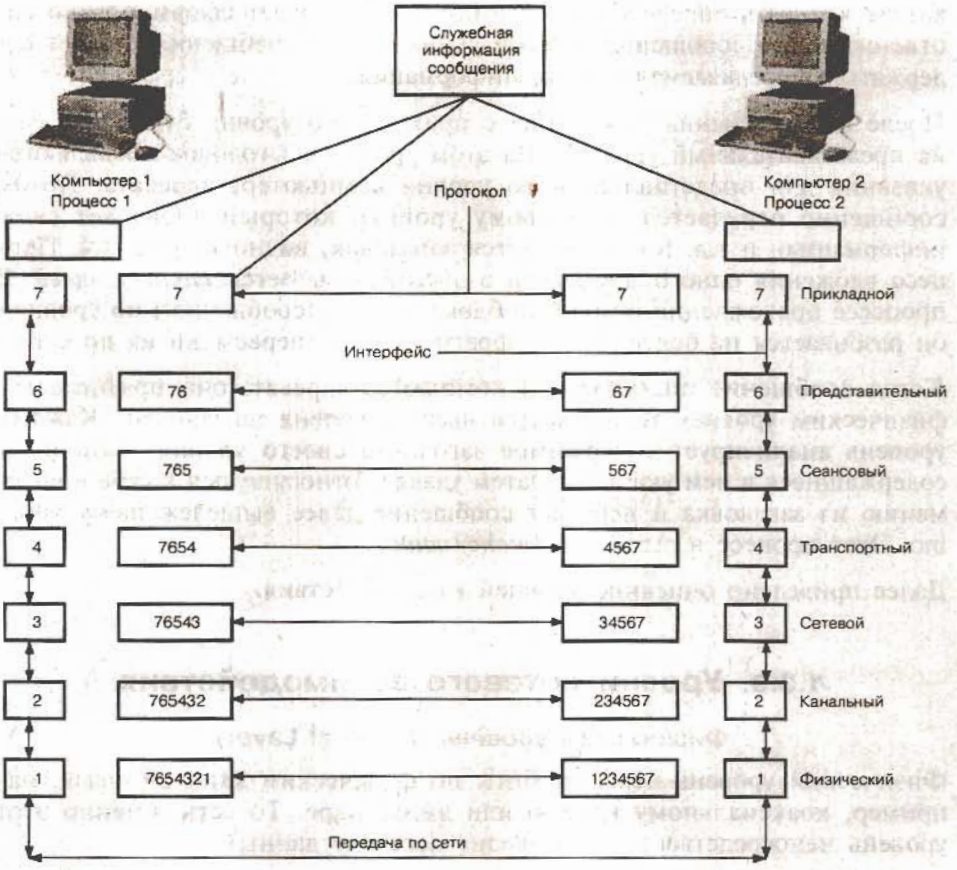


Рис. 1.3. *Модель OSI*

Из рис. 1.3 видно, что, по мере прохождения сообщения через уровни модели OSI, к пересылаемым данным добавляется служебная информация, свидетельствующая о прохождении данных через определенный уровень.

Рассмотрим взаимодействие двух компьютеров более подробно на примере файловой службы. Допустим, вам (компьютер 1) нужно записать какую-нибудь информацию в файл на удаленном компьютере 2.

Взаимодействие между компьютерами вы обычно осуществляете с помощью каких-либо программных приложений, обладающих специальным набором функций. Эти приложения работают на самом высоком уровне модели взаимодействия — прикладном. Поэтому, когда вы укажете, что

хотите записать определенные данные в файл, будет сформировано соответствующее сообщение. В поле данных этого сообщения и будет содержаться передаваемая в файл информация.

После формирования сообщение с прикладного уровня будет передано на представительный уровень. На этом уровне в заголовок добавляются указания для представительного уровня компьютера-адресата. Потом сообщение передается сеансовому уровню, который добавляет свою информацию, и т.д. Как изменяется заголовок, видно из рис. 1.4. Процесс вложения одного протокола в другой называется *инкапсуляцией*. В процессе прохождения исходного блока данных (сообщения) по уровням он разбивается на более мелкие фрагменты для пересылки их по сети.

Когда сообщение поступает на компьютер-адресат, оно принимается физическим уровнем и передается вверх с уровня на уровень. Каждый уровень анализирует содержимое заголовка своего уровня, выполняет содержащиеся в нем указания, затем удаляет относящуюся к себе информацию из заголовка и передает сообщение далее вышележащему уровню. Этот процесс называется *декапсуляцией*.

Далее приведено описание уровней взаимодействия.

1.3.3. Уровни сетевого взаимодействия

Физический уровень (Physical Layer)

Физический уровень передает биты по физическим каналам связи, например, коаксиальному кабелю или витой паре. То есть, именно этот уровень непосредственно производит передачу данных.

На этом уровне определяются характеристики электрических сигналов, которые передают дискретную информацию, например: тип кодирования, скорость передачи сигналов. К этому уровню также относятся характеристики физических сред передачи данных: полоса пропускания, волновое сопротивление, помехозащищенность.

Функции физического уровня реализуются сетевым адаптером или последовательным портом. Примером протокола физического уровня может послужить спецификация 100Base-TX (технология Ethernet).

Канальный уровень (Data link Layer)

Канальный уровень отвечает за передачу данных между узлами в рамках одной локальной сети. При этом под узлом понимается любое устройство, подключенное к сети.

Этот уровень выполняет адресацию по физическим адресам (**MAC-адресам**), «вшитым» в сетевые адаптеры предприятием-изготовителем. Каж-

дый сетевой адаптер имеет свой уникальный MAC-адрес, то есть вы не найдете две сетевые платы с одним и тем же MAC-адресом.

Канальный уровень переводит поступившую с верхнего уровня информацию в биты, которые потом будут переданы физическим уровнем по сети. Он разбивает пересылаемую информацию на фрагменты данных — кадры (frames).

На этом уровне открытые системы обмениваются именно кадрами. Процесс пересылки выглядит примерно так: канальный уровень отправляет кадр физическому уровню, который отправляет кадр в сеть. Этот кадр получает каждый узел сети и проверяет, соответствует ли адрес пункта назначения адресу этого узла. Если адреса совпадают, канальный уровень принимает кадр и передает наверх вышележащим уровням. Если же адреса не совпадают, то он просто игнорирует кадр. Таким образом, сеть на канальном уровне является широковещательной.

В используемых в локальных сетях протоколах канального уровня заложена определенная топология. Топологией называется способ организации физических связей и способы их адресации. Канальный уровень обеспечивает доставку данных между узлами в сети с определенной топологией, то есть для которой он разработан. К основным топологиям (см. рис. 1.4) относятся:

1. Общая шина;
2. Кольцо;
3. Звезда.

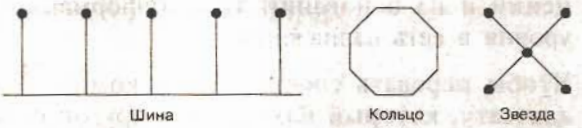


Рис. 1.4. *Основные топологии локальных компьютерных сетей*

Протоколы канального уровня используются компьютерами, мостами, маршрутизаторами. Глобальные сети (в том числе и Интернет) редко обладают регулярной топологией, поэтому канальный уровень обеспечивает связь только между компьютерами, соединенными индивидуальной линией связи.

Для доставки данных через всю глобальную сеть используются средства сетевого уровня (протоколы «точка-точка»). Примерами протоколов «точка-точка» могут послужить PPP, LAP-B. О них речь пойдет далее.

Сетевой уровень (Network Layer)

Данный уровень служит для образования единой транспортной системы, которая объединяет несколько сетей. Другими словами, сетевой уровень обеспечивает *межсетевое взаимодействие*.

Протоколы канального уровня передают кадры между узлами только в рамках сети с соответствующей топологией. Проще говоря — в рамках одной сети.

Нельзя передать кадр канального уровня узлу, который находится в другой сети. Данное ограничение не позволяет строить сети с развитой структурой или сети с избыточностью связей, а именно такой сетью является Интернет.

Построить одну большую сеть на канальном уровне также невозможно из-за физических ограничений. И хотя, например, спецификация 10Base-T позволяет использовать 1024 узла в одном сегменте, производительность данной сети не будет вас радовать, так как на канальном уровне сеть является широковещательной. То есть пакет данных (кадр) отсылается сразу всем компьютерам в сети. Если в сети немного компьютеров и быстрый канал связи, то это не страшно, нагрузка не будет критичной. А если компьютеров сети очень много (1024), то нагрузка на сеть будет очень велика, а это, в свою очередь, скажется на быстродействии сетевого взаимодействия. Все это приводит к необходимости другого решения для больших сетей. Именно такое решение и призван реализовать сетевой уровень.

На сетевом уровне термин *сеть* следует понимать как совокупность компьютеров, которые соединены в соответствии с одной из основных топологий и использующих для передачи данных один из протоколов канального уровня.

Сети соединяются специальными устройствами — *маршрутизаторами*. Маршрутизатор собирает информацию о топологии межсетевых соединений и на основании этой информации пересылает пакеты сетевого уровня в сеть назначения.

Чтобы передать сообщение от компьютера-отправителя компьютеру-адресату, который находится в другой сети, нужно совершить некоторое количество *транзитных передач между сетями*. Иногда их еще называют хопами (от англ. hop — прыжок). При этом каждый раз выбирается подходящий маршрут.

Сообщения на сетевом уровне называются пакетами. При этом на сетевом уровне работают несколько видов протоколов. Прежде всего, это сетевые протоколы, которые обеспечивают передвижение пакетов по сети, в том числе в другую сеть. Поэтому довольно часто к сетевому уровню относят протоколы маршрутизации (routing protocols) — RIP и OSPF.

Еще одним видом протоколов, работающтх на сетевом уровне, являются протоколы разрешения адреса — Address Resolution Protocol (ARP). Хотя эти протоколы иногда относят и к канальному уровню.

Классические примеры протоколов сетевого уровня: IP (стек TCP/IP), IPX (стек Novell).

Транспортный уровень (Transport Layer)

На пути от отправителя к получателю пакеты могут быть искажены или утеряны. Некоторые приложения самостоятельно выполняют обработку

ошибок при передаче данных, но большинство все же предпочитают иметь дело с надежным соединением, которое как раз и призван обеспечить транспортный уровень. Этот уровень обеспечивает требуемую приложению или верхнему уровню (сеансовому или прикладному) надежность доставки пакетов. На транспортном уровне определены пять классов сервиса:

1. Срочность;
2. Восстановление прерванной связи;
3. Наличие средств мультиплексирования нескольких соединений;
4. Обнаружение ошибок;
5. Исправление ошибок.

Обычно уровни модели OSI, начиная с транспортного уровня и выше, реализуются на программном уровне соответствующими компонентами операционных систем.

Примеры протоколов транспортного уровня: TCP и UDP (стек TCP/IP), SPX (стек Novell).

Сеансовый уровень (Session Layer)

Сеансовый уровень устанавливает и разрывает соединения между компьютерами, управляет диалогом между ними, а также предоставляет средства синхронизации. Средства синхронизации позволяют вставлять определенную контрольную информацию в длинные передачи (точки). Благодаря этому в случае обрыва связи можно вернуться назад (к последней точке) и продолжить передачу с места обрыва.

Сеанс — это логическое соединение между компьютерами. Каждый сеанс имеет три фазы:

1. Установление соединения. Здесь узлы «договариваются» между собой о протоколах и параметрах связи.
2. Передача информации.
3. Разрыв связи.

Не нужно путать сеанс сетевого уровня с сеансом связи. Пользователь может установить соединение с Интернетом, но не устанавливать ни с кем логического соединения, то есть не принимать и не передавать данные.

Представительный уровень (Presentation Layer)

Представительный уровень изменяет форму передаваемой информации, но не изменяет ее содержания. Например, средствами этого уровня может быть выполнено преобразование информации из одной кодировки в другую. Также на этом уровне выполняется шифрование и дешифрование данных.

Примером протокола представительного уровня может служить протокол SSL (Secure Socket Layer). Данный протокол обеспечивает секретный обмен данными.

Прикладной уровень (Application Layer)

Данный уровень представляет собой набор разнообразных протоколов, с помощью которых пользователи сети получают доступ к совместно используемым ресурсам. Единица данных называется сообщением.

Примеры протоколов: HTTP, FTP, TFTP, SMTP, POP, SMB, NFS.

1.3.4. Интернет и модель OSI

При взаимодействии открытой системы и Интернет модель OSI упрощается, так как некоторые протоколы Интернет включают в себя функции нескольких уровней (табл. 1.1). Если к сети Интернет подключается один пользователь, а не вся сеть, то автоматически исчезают канальный и физический уровни, потому что нет сетевых адаптеров, а значит, нет и физических адресов. В данном случае конечным протоколом будет протокол типа «точка-точка», например, PPP. В этот протокол будут вложены все остальные.

Соответствие уровней стека TCP/IP модели OSI — *Таблица 1.1*

Уровень модели OSI	Протокол	Уровень стека TCP/IP
7,6	WWW (HTTP), FTP, TFTP, SMTP, POP, telnet, WAIS, SNMP	1
5,4	TCP, UDP	2
3	IP, ICMP, RIP, OSPF, ARP	3
2,1	Ethernet, PPP, SLIP	4

Стек TCP/IP разработан по заказу Министерства обороны США для сети ARPAnet, которая стала прототипом современного Интернета. Стек стал популярным, благодаря развитию операционной системы Unix, которая его использовала.

На нижнем уровне стек поддерживает все популярные стандарты физического и канального уровней: Ethernet, Fast Ethernet, Token Ring, FDDI, SLIP, PPP, ISDN.

Основными протоколами стека TCP/IP, как нетрудно догадаться, являются протоколы TCP и IP. Протокол TCP гарантирует надежность доставки пакета, а IP — его продвижение по сети. В модели OSI протокол IP относится к сетевому уровню, а TCP — к транспортному.

За долгие годы развития стека TCP/IP для него было разработано много протоколов прикладного уровня: HTTP (Протокол Передачи Гипертекстовой Информации), FTP (Протокол передачи файлов), TFTP (Тривиальный FTP), SMTP (Простой Протокол Передачи Почты), telnet (протокол эмуляции терминала), и др. Все эти протоколы относятся к представительскому и прикладному уровням модели OSI.

1.4. Адресация в сети Интернет

1.4.1. Вид IP-адреса и способы его назначения

Любому компьютеру в IP-сети (TCP/IP-сети) назначен уникальный адрес, который называется IP-адресом. **IP-адрес** — это 32-разрядное число, которое принято записывать в десятеричном или шестнадцатеричном формате в виде четырех чисел, разделенных точками, например:

1. 111.111.213.232
2. 127.0.0.1
3. 192.168.9.2

При условии, что ваша сеть подключена к Интернет, протокол TCP/IP обеспечивает работу вашей сетевой программы с любым компьютером в мире, как будто тот находится в локальной сети. Уникальность IP-адреса достигается достаточно просто — IP-адреса назначаются централизованно **Сетевым Информационным Центром** (NIC — Network Information Center).

Для понимания остальной информации нужно отметить, что существуют локальные (LAN — Local Area Networks) и региональные (Wide Area Networks) сети. Сеть Интернет сначала была региональной (Arpanet), а потом стала глобальной, объединив все региональные сети мира. Если ваша локальная (или даже региональная) сеть не соединена с Интернет, то внутри сети вы можете использовать любые IP-адреса без согласования с NIC. Обычно в локальных сетях используются особые IP-адреса, о которых мы поговорим немного позже.

Любую сеть, независимо от типа — LAN или WAN, можно разделить на подсети. Причины разбиения сети на подсети кроются в ранних версиях протокола IP. Тогда существовало несколько сетей класса A, содержащих несколько миллионов узлов (о классах читайте далее). Помимо всего прочего, в таких сетях очень велика вероятность коллизий, то есть одновременного доступа двух или более узлов к среде передачи данных. Управлять такой сетью крайне неудобно, да и сеть будет перегружена собственным трафиком. Поэтому основной принцип разделения — «разделяй и властвуй».

К другим причинам разделения относят создание маленьких подсетей с использованием разных технологий — Ethernet, Token Ring, FDDI, ATM.

Вы не можете смешивать эти технологии в одной сети, однако они могут быть взаимосвязаны с помощью разделения на подсети. Разделение на подсети может быть также произведено из соображений безопасности.

Как я уже писал, каждый компьютер в сети имеет свой уникальный адрес. Но оказывается, что и сеть (подсеть) также имеет свой уникальный адрес. Под сетью можно понимать «пачку» IP-адресов, идущих подряд, то есть 192.168.1.0...192.168.1.255. Самый младший и самый старший адреса резервируются. Младший (192.168.1.0) является адресом сети, а старший является широковещательным (broadcast) адресом сети. Адрес сети может потребоваться, когда нужно указать всю сеть (подсеть), например, при задании маршрутизации для этой сети.

Представьте, что у вас есть две отдельные сети и вам нужно объединить их в одну. Тогда эта одна «большая» сеть станет называться сетью, а две «маленькие» — подсетями. Устройство, которое будет обеспечивать связь этих сетей (маршрутизацию), называется, как уже было отмечено выше, маршрутизатором. Маршрутизатор может быть, как аппаратным (отдельное устройство), так и программным. В роли программного маршрутизатора может выступать любой компьютер с двумя (или более) сетевыми интерфейсами, например, двумя сетевыми платами.

Для каждой подсети определена ее маска. Фактически, маска — это размер сети, то есть число адресов в сети. Маску принято записывать в десятично-побайтном виде:

1. 255.255.255.0 — маска на 256 адресов (0...255);
2. 255.255.255.192 — маска на 64 адреса (192...255);
3. 255.255.0.0 — маска на 65536 адресов (256×256).

1.4.2. Классы IP-сетей

В общем случае IP-сети делятся на классы: A, B, C, D и E.

Сети класса A — это огромные сети. Маска сети класса A: 255.0.0.0. В каждой сети такого класса может находиться 16777216 адресов. Адреса таких сетей лежат в промежутке 1.0.0.0...126.0.0.0, а адреса хостов (компьютеров) имеют вид: 125.*.*.*

Сети класса B — это средние сети. Маска такой сети — 255.255.0.0. Эта сеть содержит 65536 адресов. Диапазон адресов таких сетей 128.0.0.0...191.255.0.0. Адреса хостов имеют вид: 136.12.*.*

Сеть класса C — маленькие сети. Содержат 256 адресов (на самом деле всего 254 хоста, так как номера 0 и 255 зарезервированы). Маска сети класса C — 255.255.255.0. Интервал адресов: 192.0.1.0...223.255.255.0. Адреса хостов имеют вид: 195.136.12.*

Класс сети определить очень легко. Для этого нужно перевести десятичное представление адреса сети в двоичное. Например, адрес сети 128.11.1.0 в двоичном представлении будет выглядеть так:

```
10000000 00001011 00000001 00000000
A 192.168.1.0:
11000000 10101000 00000001 00000000
```

Если адрес начинается с последовательности битов 10, то данная сеть относится к классу B, а если с последовательности 110, то — к классу C.

Если адрес начинается с последовательности 1110, то сеть является сетью **класса D**, а сам адрес является особым — групповым (multicast). Если в пакете указан адрес сети класса D, то этот пакет должны получить все хосты, которым присвоен данный адрес.

Адреса **класса E** зарезервированы для будущего применения. В табл. 1.2 приведены сравнительные характеристики сетей классов A, B, C, D и E.

Характеристики сетей различных классов *Таблица 1.2*

Класс	Первые биты	Диапазон адресов	Количество узлов
A	0	1.0.0.0...126.0.0.0	16777216 (2^{24})
B	10	128.0.0.0...191.255.0.0	65536 (2^{16})
C	110	192.0.1.0...223.255.255.0	256 (2^{8})
D	1110	224.0.0.0...239.255.255.255	Multicast
E	11110	240.0.0.0...247.255.255.255	Зарезервирован

Теперь самое время немного сказать о специальных адресах, о которых я упомянул немного выше. Если весь IP-адрес состоит из нулей (0.0.0.0), то он обозначает адрес того узла, который сгенерировал этот пакет.

Адрес 255.255.255.255 — это широковещательный адрес. Пакет с таким адресом будет рассылаться всем узлам, которые находятся в той же сети, что и источник пакета. Это явление называется ограниченным широковещанием. Существует также другая рассылка, которая называется широковещательным сообщением. В этом случае вместо номера узла стоят все единицы в двоичном представлении (255). Например, 192.168.2.255. Это означает, что данный пакет будет рассылаться всем узлам сети 192.168.2.0.

Особое значение имеет IP-адрес 127.0.0.1 — это адрес локального компьютера. Он используется для тестирования сетевых программ и взаимодействия сетевых процессов. При попытке отправить пакет по этому адре-

су данные не передаются по сети, а возвращаются протоколам верхних уровней, как только что принятые. При этом образуется как бы «петля». Этот адрес называется **loopback**. В IP-сети запрещается использовать IP-адреса, которые начинаются со 127. Любой адрес подсети 127.0.0.0 относится к локальному компьютеру, например: 127.0.0.1, 127.0.0.5, 127.77.0.6.

Существует также специальные адреса, которые зарезервированы для несвязанных локальных сетей — это сети, которые используют протокол IP, но не подключены к Интернет. Вот эти адреса:

1. 10.0.0.0 (сеть класса A, маска сети 255.0.0.0).
2. 172.16.0.0 — 172.31.0.0 (16 сетей класса B, маска каждой сети 255.255.0.0).
3. 192.168.0.0 — 192.168.255.0 (256 сетей класса C, маска каждой сети 255.255.255.0).

В этой книге я старался использовать именно такие адреса, чтобы не вызвать пересечение с реальными IP-адресами.

1.4.3. Доменные имена, URL-адреса и их использование

URL (Universal Resource Locator) — это универсальный идентификатор ресурса. Чтобы просмотреть какую-нибудь Web-страницу, вы должны знать ее имя и адрес, а также протокол, с помощью которого происходит передача Web-страницы. Все это вместе (протокол, адрес и имя) называются URL. Для Web-страниц обычно указывается протокол http.

Примеры URL:

```
http://www.pubnit.com/
ftp://server.ru/file1.zip
```

Рассмотрим следующий URL:

```
http://www.firma.ru/my_favorite_page.html
```

`Http` — протокол, `www.firma.ru` — сервер, `my_favorite_page.html` — имя страницы на сервере.

Вдумаемся в название сервера **www.firma.ru**. Что обозначает **www**, что **firma**, а что — **ru**? Если подумать логически, то **www** — это стандартное имя для всех Web-серверов. Наш сервер принадлежит российской (**ru**) фирме **firma**.

Однако не все так просто, как кажется на первый взгляд. Вспомним, для чего предназначена служба DNS. Правильно, для преобразования символьного имени компьютера в его IP-адрес. Так как компьютеров в Интернете очень много, то база данных серверов DNS должна быть иерар-

хической. Предположим, что компьютеров у нас мало. Тогда базу данных можно было бы сделать плоской, например, такой:

```
Имя       IP-адрес
host1     111.111.111.111
host2     111.111.111.112
....
hotN      111.111.NNN.NNN
```

Представляете, какой был бы файл, содержащий такую таблицу? А сколько бы длился поиск нужного имени? Поэтому для удобства (и конечно же для оптимизации поиска) все компьютеры в Интернете разбиты на группы — зоны или домены. Обычно на поддержку одного домена выделяют два компьютера — первичный и вторичный серверы DNS. Если домены маленькие, можно настроить один сервер так, чтобы он поддерживал несколько доменов. Так обычно и делается в относительно небольших корпоративных сетях.

Вернемся к имени сервера **www.firma.ru**. Здесь **ru** — это домен, принадлежащий России. Домен может содержать поддомены: **firma** — это поддомен домена **ru**. А что такое **www**? А это и есть имя компьютера в домене **firma.ru**. Возможно, в домене не будет настоящего имени **www** (ну никто из пользователей не захотел так обозвать свой компьютер). Тогда администратор домена создаст псевдоним **www** для какого-нибудь компьютера, например, для компьютера **main**, на котором запущен Web-сервер. В следующем пункте мы подробнее рассмотрим работу службы DNS.

1.5. Служба доменных имен DNS

Для того чтобы подключиться к какому-нибудь другому компьютеру, например, Web-серверу, нужно знать его IP-адрес. Это не очень удобно, потому что человеку намного проще запомнить символьное название сервера, чем последовательность чисел. Представьте, что вместо **http://dkws.narod.ru** в окне браузера вам нужно было бы вводить **http://213.180.199.4**. Оба способа будут работать, но первый запоминается намного проще. Фактически, нужно запомнить только слово из четырех букв — dkws и выражение «на Народе». Компьютеру же, наоборот, проще обрабатывать числа, а не символьную информацию.

Для преобразования IP-адреса в символьное имя и обратно используется служба доменных имен — **DNS** (Domain Name System). Обычно на любом сервере устанавливается своя служба DNS, даже если этот сервер не поддерживает домена. В отличие от одноранговой сети, в IP-сети компьютеры объединяются в домены, а не в рабочие группы. На самом деле, понятие домен гораздо шире, чем рабочая группа, но пока остановимся на таком определении.

Предположим, адрес Web-сервера вашего подразделения выглядит так: **http://www.department.firma.isp.ru**. Рассмотрим, что происходит, когда пользователь вводит в окне браузера этот адрес. Сначала отправляется запрос на разрешение (преобразование) имени в IP-адрес серверу DNS, который принадлежит провайдеру пользователя. Если такое имя есть в кэше DNS-сервера провайдера (для определенности назовем его **user-dns**), он возвращает IP-адрес и браузер устанавливает соединение с этим компьютером. Если же такого адреса в кэше сервера DNS не оказалось, DNS-сервер провайдера обращается к серверу, который содержит домен наивысшего уровня, то есть к корню дерева (см. рис. 1.5). Тот обращается к домену **ru** (а этот сервер пусть называется **ru-dns**).

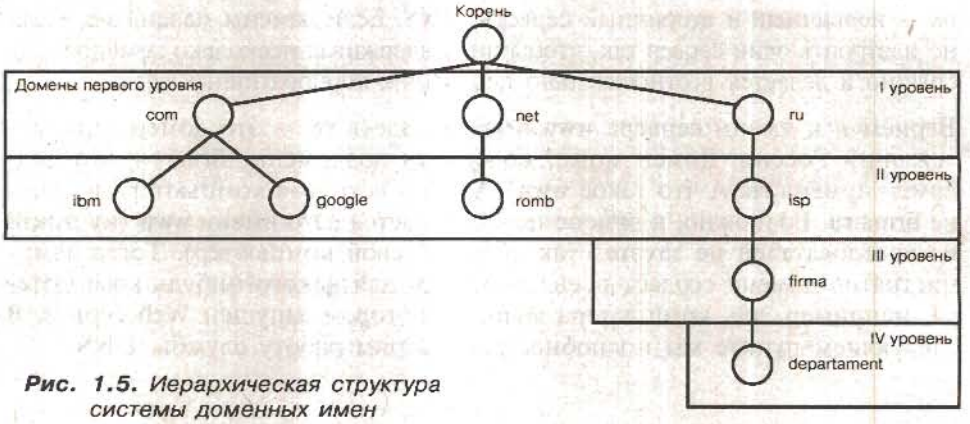


Рис. 1.5. *Иерархическая структура системы доменных имен*

Сервер **ru-dns**, в свою очередь, обращается к серверу, который делегирует домен isp (это ваш провайдер). Сервер **isp** обращается к серверу, который администрирует домен firma, а он уже к серверу, отвечающему за домен **department**, который и возвращает IP-адрес компьютера **www.department.firma.isp.ru**. Таким образом, получается своеобразная цепочка. Ясно, что если эта цепочка оборвется на каком-нибудь звене, то пользователю, точнее, серверу DNS **user-dns**, будет сообщено о невозможности разрешения имени компьютера в IP-адрес. На самом деле все немного сложнее, но это уже тема для отдельной книги.

Вся структура службы DNS является иерархической. Существуют домены первого, второго, третьего, n-го уровней. В рассмотренном примере доменом первого уровня является **ru**, **isp** — второго, **firma** — третьего, а **department** — четвертого уровня (см. рис. 1.5).

Корневой домен управляется центром InterNIC. Домены верхнего (первого) уровня назначаются для каждой страны (см. табл. 1.3).

Обозначения стран по стандарту ISO 3166 *Таблица 1.3*

Домен	Страна	Домен	Страна	Домен	Страна
ru	Россия	id	Индонезия	ir	Иран
by	Белоруссия	gr	Греция	tm	Туркменистан
lv	Латвия	at	Австрия	it	Италия
md	Молдова	hu	Венгрия	gb	Великобритания
tr	Турция	ua	Украина	de	Германия
iq	Ирак	lt	Литва	vn	Вьетнам
il	Израиль	ee	Эстония	va	Ватикан
pl	Польша	kz	Казахстан	co	Колумбия
es	Испания	ro	Румыния	mx	Мексика
fr	Франция				

Для США и Канады единый домен отсутствует, но иногда используется обозначение us. Обозначения стран соответствуют международному стандарту ISO 3166. Данные сведения могут быть получены по адресу **ftp:/ftp.ripe.net/iso3166-countrycodes**.

Для различных типов организаций могут использоваться такие обозначения:

1. com — коммерческие организации (например, yahoo.com);
2. edu — образовательные учреждения (например, mit.edu);
3. gov — правительственные организации (например, nasa.gov);
4. org — некоммерческие организации (например, linux.org);
5. net — обычно провайдеры (например, ukr.net).

По данным ISC (Internet Software Consortium) по состоянию на январь 2002 года зарегистрировано около 150 миллионов узлов сети Интернет (см. рис. 1.6). Данные сведения публикуются с разрешения ISC.

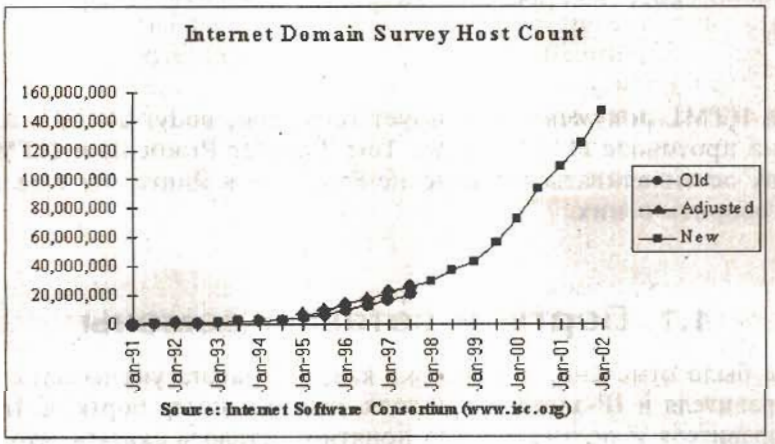


Рис. 1.6. *Динамика роста узлов в сети Интернет*

1.6. World Wide Web и HTML-документы

World Wide Web переводится с английского, как «Всемирная паутина». Сейчас попробуем разобраться, почему именно так, а не иначе. WWW-серверы (их часто называют просто Web-серверами) предназначены для передачи гипертекстовой информации. Для отображения этой информации предназначены специальные программы — браузеры. Вы запускаете браузер, вводите имя Web-сервера и созерцаете Web-страницу.

Чем же гипертекстовая информация отличается от самой обыкновенной — текстовой. Самое основное отличие заключается в том, что гипертекстовый документ может содержать ссылку на другой такой же документ (или любой другой файл). Причем документ, на который ссылаются, может находиться на совершенно другом сервере. В свою очередь, этот документ может содержать ссылки на другие документы. Если представить все это в глобальном плане, то получается симпатичная паутинка.

Язык HTML (Hyper Text Markup Language) используется для написания HTML-документов (гипертекстовых документов). Кроме ссылок на другие документы, HTML-документ может содержать форматирование информации для лучшего ее восприятия. Служебная информация заключается в специальные теги. Вот пример небольшого HTML-документа:

```
<html>
<body>
<h1>Привет, Мир!</h1>
<a href="doc1.html">Это ссылка на документ doc1.html, находящийся на нашем сервере</a>
<br><a href="http://server.ru/doc2.html">Это ссылка на документ doc2.html, находящийся на сервере server.ru</a>
</body>
</html>
```

Данный HTML-документ использует теги html, body, h1, br и a. Подробно на протоколе HTTP (Hyper Text Transfer Protocol) и HTML-документах останавливаться сейчас не будем — в книге мы еще не раз будем говорить о них.

1.7. Порты и сетевые демоны

Как уже было отмечено, в заголовке каждого пакета указывается IP-адрес отправителя и IP-адрес получателя, а также номер порта. С IP-адресом отправителя и получателя все понятно, осталось сказать, что же та-

кое порт. Дело в том, что сразу несколько приложений на одном компьютере могут осуществлять обмен данными через сеть. При этом, если в качестве адресата указывать только IP-адрес получателя, то приложения выполняемые на нем, не смогут разобраться кому из них предназначены присланные данные.

Можно, конечно, каждому приложению назначать свой IP-адрес, однако такой подход приведет к перерасходу адресного пространства.

Чтобы решить эту проблему, используется механизм портов. Каждый порт имеет свой номер. Каждой сетевой программе, которая работает по протоколу TCP/IP, сопоставлен свой номер порта. Например, 80 — это порт WWW-сервера (обычно это Apache), а 53 — это порт системы доменных имен.

Термин **демон** происходит от английского слова demon (или daemon) и означает программу, которая выполняется в фоновом режиме и дополняет операционную систему каким-нибудь сервисом. Как правило, пользователь не замечает работу демона: он даже и не подозревает, что данная программа запущена. Предположим, что наш компьютер получил пакет, в заголовке которого сказано, что данный пакет предназначен для программы с номером 21 (это номер порта демона FTP). Суперсервер (это тоже демон), запущенный на нашем компьютере, ищет программу с номером 21 и запускает ее, передав ей пакет. Если программа с номером 21 не установлена на нашем компьютере, суперсервер сообщает удаленному компьютеру, что «заданный порт недоступен». Программа с номером 21 может работать в автономном режиме — как демон. То есть она сама будет контролировать входящие пакеты, обрабатывая только пакеты, в заголовке которых указан порт с номером 21.

1.8. Структура пакетов IP и TCP

Когда мы рассматривали метод коммутации пакетов, мы узнали, что пакет состоит из заголовка и поля данных, содержащего передаваемые данные. Тогда мы были не готовы к подробному рассмотрению структуры заголовка пакета, но сейчас мы можем сделать это.

Протокол IP не ориентирован на соединение, поэтому не обеспечивает надежную доставку данных. Поля, описание которых приведено в табл. 1.4, представляют собой IP-заголовок и добавляются к пакету при его получении с транспортного уровня.

Протокол TCP, в отличие от протокола IP, ориентирован на установление соединения и обеспечивает надежную доставку данных. Структура TCP-пакета описана в табл. 1.5.

Структура заголовка IP-пакета *Таблица 1.4*

Поле	Описание
Source IP-address (IP-адрес отправителя)	Отправитель пакета
Destination IP-address (IP-адрес получателя)	Получатель пакета
Protocol (Протокол)	TCP или UDP
Checksum (Контрольная сумма)	Значение для проверки целостности пакета
TTL (Time to Live, время жизни пакета)	Определяет, сколько секунд пакет может находиться в сети. Предотвращает бесконечное блуждание пакетов в сети. Значение TTL автоматически уменьшается на одну или более секунд при проходе через каждый маршрутизатор сети
Version	Версия протокола IP — 4 или 6. Шестая версия протокола IP рассматривается в гл. 8 (4 бита)
Header Length (Длина заголовка)	Минимальный размер заголовка — 20 байт (4 бита)
Type of Service (Тип обслуживания)	Обозначение требуемого для этого пакета качества обслуживания при доставке через маршрутизаторы IP-сети. Здесь определяются: приоритет, задержки, пропускная способность (8 бит)
Total Length (Общая длина)	Длина дейтаграммы IP-протокола (16 бит)
Identification (Идентификация)	Идентификатор пакета. Если пакет фрагментирован (разбит на части), то все фрагменты имеют одинаковый идентификатор (16 бит)
Fragmentation Flags (Фрагментационные флаги)	3 бита для флагов фрагментации и 2 бита для текущего использования
Fragmentation Offset (Смещение фрагмента)	Указывает на положение фрагментов относительно начала поля данных IP-пакета. Если фрагментации нет, смещение равно 0×0 (13 бит)
Options and Padding (Опции и заполнение)	Опции

Структура TCP-пакета *Таблица 1.5*

Поле	Описание
Source port (Порт отправителя)	Порт TCP узла-отправителя
Destination Port (Порт получателя)	Порт TCP узла-получателя
Sequence Number (Порядковый номер)	Номер последовательности пакетов
Acknowledgement Number (Номер подтверждения)	Порядковый номер байта, который локальный узел рассчитывает получить следующим
Data Length (Длина данных)	Длина TCP-пакета
Reserved (Зарезервировано)	Зарезервировано для будущего использования
Flags (Флаги)	Описание содержимого сегмента
Window (Окно)	Показывает доступное место в окне протокола TCP
Checksum (Контрольная сумма)	Значение для проверки целостности пакета
Urgent Pointer (Указатель срочности)	При отправке срочных данных (поле Flags) в этом поле задается граница области срочных данных

1.9. Глоссарий

В этом небольшом пункте приведены описания основных терминов, которые использовались в первой главе, а также некоторых новых.

Аутентификация — проверка подлинности.

Интерфейс — средства и правила взаимодействия компонент системы между собой.

Маршрутизатор (router) — устройство для пересылки пакетов. Маршрутизатор собирает информацию о топологии межсетевых соединений и на основании этой информации пересылает пакеты сетевого уровня в сеть назначения. Маршрутизатор может быть программным и аппаратным.

Маршрутизация (routing) — процесс передачи пакетов данных между двумя подсетями.

Порт — физическое или логическое устройство, через которое осуществляется процесс приема/передачи данных.

Протокол — совокупность правил, определяющая взаимодействие абонентов вычислительной системы и описывающая способ выполнения определенного класса функций.

Сервер — специальный компьютер, который предоставляет определенные услуги другим компьютерам.

Сокет (soket) — во многом аналогичен дескриптору файла (file handle). Сокет обеспечивает конечную точку соединения. Приложение, создавая сокет, указывает три параметра: IP-адрес узла, протокол (TCP или UDP) и порт, используемый приложением.

Узел — устройство, подключенное к сети.

Шлюз (Gateway) — межсетевой преобразователь. Выполняет функции, аналогичные мосту, но используется для связи сетей разных типов, например, LAN и WAN. Обычно для устройства, которое связывает две локальные сети, используется термин «маршрутизатор», а для устройства, которое соединяет вашу локальную сеть с Интернет, — шлюз, хотя оба эти устройства выполняют одну и ту же функцию — маршрутизацию пакетов.

Глава 2

Выбор хостинга. Регистрация доменного имени

2.1. Хостинг и хостинг-провайдеры

Организация, которая занимается созданием Internet-хостов (физических или виртуальных) с целью их продажи, называется **хостинг-провайдером** или **хостером**.

Сам процесс создания Internet-узла с последующим предоставлением к нему доступа другим пользователям Internet называется **хостингом**.

Наверное, вы обратили внимание на выражение «с целью продажи» в первом предложении. А как же так называемый «бесплатный» хостинг. Бесплатным является только сыр в мышеловке: если за хостинг не платите вы явно, то это совсем не говорит о том, что он бесплатен. За него платят другие пользователи, которые размещают рекламу в Internet. Эта реклама всплывает в небольших окнах, которые мелькают перед глазами пользователей, когда те заходят на ваш сайт. Это и есть плата за «бесплатный» хостинг.

Как будущему Web-программисту я рекомендую зарегистрировать виртуальный узел в какой-нибудь организации, предоставляющей хостинг. Попросту говоря, создать свой сайт. Даже если вам пока нечего туда выкладывать. Собственный сайт пригодится вам для тестирования собственных проектов. Конечно, первоначальное тестирование вы будете производить на своей домашней машине, а затем — на своем сайте: мало ли что может случиться. И только после того, как вы убедитесь, что ваш сценарий безупречно работает не только на домашней машине, но и на вашем сайте, его можно показывать заказчику.

В этой главе мы поговорим о компаниях, предоставляющих хостинг.

2.2. Обзор хостинг-провайдеров

2.2.1. Платный хостинг

AvaHost.Ru — http://www.avahost.ru

Компания AvaHost.Ru предоставляет услуги по регистрации доменных имен и предоставлению высококачественного хостинга. Преимущества хостинга этой компании заключаются в следующем:

- Регистрация домена в домене .COM (your_name.com).
- На сервере хостера установлена операционная система Red Hat Linux.
- 200 Мб свободного места (можно увеличить до 1100 Mb).
- 3500 Мб трафика (за месяц).
- Доступ к каталогу cgi-bin.
- Панель Управления с Web-интерфейсом.
- Бесплатный хостинг вашего домена.
- Неограниченная переадресация электронной почты.
- 10 почтовых учетных записей (POP3).
- Web-интерфейс WebMail.
- Собственный SMTP-сервер.
- Доступ по протоколу FTP.
- Дополнительные учетные записи пользователей.
- Неограниченные автоответчики.
- Поддержка Microsoft Frontpage.
- Сервер баз данных MySQL.
- Поддержка PHP.
- Поддержка Macromedia Flash and Shockwave.
- Поддержка Server Side Includes (SSI).
- Языки Perl, Python & Php.
- Ограничение доступа к каталогам.

За все это вам придется выложить $7.25 в месяц! Если вам мало 200 Мб дискового пространства и 3500 Мб трафика, вы можете заказать пакет Premium, который включает в себя 1100 Мб дискового пространства и 15000 Мб трафика, а также неограниченное число почтовых учетных записей. Это удовольствие обойдется вам в $29.25/месяц.

К тому же, если вас не устаивают услуги компании, в течение 30 дней вам вернут ваши деньги. Хостинг этой компании подойдет не только для домашнего сайта, но и для сайта солидной коммерческой компании.

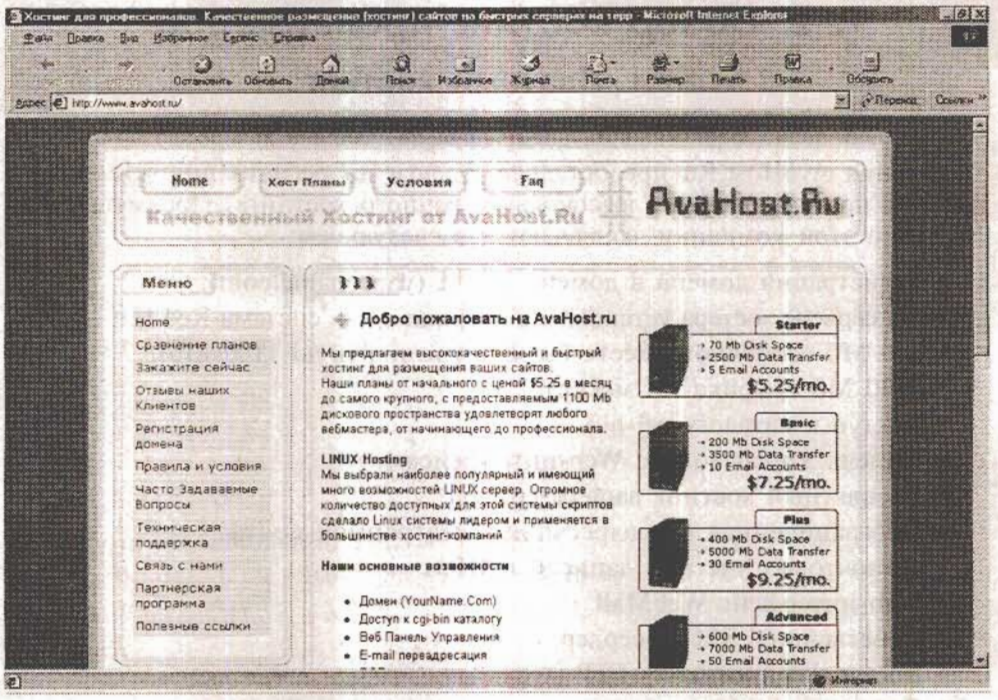


Рис. 2.1. *Хостинг от компании AvaHost.Ru*

AlexHost.Ru — http://www.alexhost.ru

Услуги компании AlexHost обойдутся вам немного дороже, а именно — $8.95 в месяц. При этом вы получите 100 Мб дискового пространства для размещения ваших страниц. Преимущество данного хостинга заключается в следующем:

- ежедневное резервное копирование (backup);
- неограниченный трафик.

Согласитесь, оба эти фактора чрезвычайно важны. Вам не нужно беспокоиться, если по каким-то причинам вылетела ваша база данных, в которой находились сведения о тысячи ваших клиентов. А также вам не нужно платить за трафик.

Также вы получите:

- Web-интерфейс для управления сервером и почтой.
- Независимые почтовые ящики с разными паролями (до 15 ящиков).
- FTP-доступ к сайту.
- Доступ к серверу MySQL.

- Поддержку PHP, Perl, SSI, SSH, WAP (Wireless Application Protocol).
- Доступ к планировщику Crond.

Последнюю возможность довольно редко встретишь в списке услуг хостинг-провайдера. Если доступ по SSH и SSI поддерживают почти все хостеры, то редактирование таблиц демона Crond встречается далеко не всегда. То же самое можно сказать и о WAP-доступе.

Agava.Ru — http://www.agava.ru, http://www.holm.ru

Компания Agava Software может зарегистрировать доменное имя в доменах: .ru, .com.ru, .net.ru, .net и других. Компания предлагает три тарифных плана:

- Lite.
- Normal.
- Pro.

Для Web-программиста больше подойдет план Normal, потому что Lite не поддерживает CGI, PHP и MySQL. «Нормальный» виртуальный узел вам обойдется в $9/месяц при оплате до шести месяцев сразу. При этом вы получите:

- Доменное имя.
- Неограниченный трафик.
- Web-интерфейс для почты (POP3, SMTP, IMAP).
- Антивирус.
- Неограниченное число почтовых ящиков.
- Поддержку CGI.
- Поддержку PHP и MySQL.
- Поддержку Perl.
- Server Side Includes (SSI).
- Доступ к узлу по ssh и по FTP.

2.2.2. Бесплатный хостинг

Бесплатный хостинг от Agava

Возможно, у вас совсем нет желания выкладывать $102,6 в год (при оплате сразу за весь год «абонплата» ниже), поэтому компания Agava Software может предоставить вам бесплатный хостинг:

- Доменное имя третьего уровня (your_name.holm.ru), при этом возможна покупка доменного имени второго уровня (это не будет стоить дорого).
- Неограниченное место (*).

- Поддержка CGI, SSI, Perl.
- Поддержка PHP.
- Доступ к базе данных MySQL (*).
- Форвардинг всей почты вашего домена на один адрес.
- SSH- и FTP-доступ к узлу.
- Доступ к журналам сервера.

Довольно неплохо, как для бесплатного хостинга. В качестве оплаты вам нужно будет разместить баннер Agava Software на своем сайте.

Примечание.

Для получения неограниченного места и доступа к базе данных MySQL нужно обосновать их необходимость.

2.3. Поиск оптимального хостинг-провайдера

Конечно, кроме описанных выше компаний существует множество других, предоставляющих качественный хостинг. Я лишь упомянул наиболее известные из них. Если вас по каким-либо причинам не устраивают предлагаемые компаниями услуги, в любом поисковике введите слово «хостинг» или выражение «бесплатный хостинг». Вы также можете воспользоваться каталогом хостинг-провайдеров `http://internethosting.ru/main.asp`. Средство для поиска хостеров (`http://internethosting.ru/hosting/search/`) поможет вам выбрать нужного хостинг-провайдера по предлагаемым критериям. Например, я выполнил поиск по таким критериям:

- Доступ по FTP.
- Почта.
- Поддержка CGI.
- Остальные параметры меня не интересовали.

Получил такой результат (см. рис. 2.2).

домой	новости	[хостинг]	вопросы	контакты
персональный хостинг	найдено 4 серверов этой категории			страница 1 из 1

Сервер	Язык	Объем	Загрузка	Реклама	CGI - сценарии	Адрес
Demented	English	20 Mb	FTP Email	No	CGI-BIN	Subdomain
eSmartStart	English	100 Mb	FTP Browser Email	Topbanner	CGI-BIN	various
Hosticon.com	English	15 Mb	FTP Browser Email	Popup	CGI-BIN	Subdomain
Loopback.dk	Danish	5 Mb	FTP Email	No	CGI-BIN	Subdomain

назад | 1 | вперед

Рис. 2.2. *Результат поиска бесплатного хостинга с помощью http://internethosting.ru/*

2.4. На что следует обратить внимание

Прежде всего, нужно просчитать бюджет вашего будущего сайта. Сколько денег вы планируете тратить в месяц на содержание сайта? Скорее всего, счет за хостинг не превысит вашу зарплату, но все может быть... Поэтому займемся подсчетами прямо сейчас. Нужно попытаться спрогнозировать входящий и исходящий трафик — хотя бы примерно. Когда вы платите за хостинг, в счет входят содержание хостинга (дисковое пространство) и трафик. Если суммарный трафик превысит максимально допустимое значение для выбранного вами тарифного плана, вам придется доплатить вашему хостеру, иначе ваш сайт будет закрыт.

Какая информация будет размещена на вашем сайте? Если там будет только реклама вашей фирмы (или лично ваша — как разработчика), то даже при большом «наплыве» посетителей тарифный план не будет превышен. А вот если на сайте выложена последняя версия вашей популярной программы, которая в архиве занимает несколько мегабайт... Даже при средней посещаемости вы рискуете превысить тарифный план.

Что же делать? В первом случае (реклама) выбирайте наиболее дешевый хостинг. Например, самый простой хостинг от AvaHost.Ru с поддержкой PHP и MySQL обойдется вам в $5.25/месяц. По-моему, это оптимальный вариант. А во втором случае целесообразно выбрать хостинг с неограниченным трафиком, даже если он стоит чуть дороже. Подойдет хостинг от AlexHost.Ru.

Просчитав бюджет сайта, можно перейти к рассмотрению следующих критериев: наличие всех необходимых вам сервисов, дисковое пространство и операционная система.

Для нас наиболее критичными являются PHP и MySQL. Если хостер разрешает их использовать, значит, он нам подходит. Также обратите внимание на наличие службы расписаний crond (или аналогичной ей). Даже если сейчас она вам не нужна, в дальнейшем она вам понадобится, например, для рассылки сообщений пользователям сайта в одно и то же время.

При выборе тарифного плана задумайтесь о дисковом пространстве. Заплатив на 2 доллара дороже, вы получите в два (или даже в три) раза больше места под ваш сайт. Но стоит ли это делать? Пока ваш сайт не превысил минимального размера (например, 70 Мб для AvaHost.RU), не стоит переплачивать. Со временем вы можете перейти на другой тарифный план, если ваш сайт «разрастется».

Большинство хостеров используют операционную систему Linux. Во-первых, она более надежна, а во-вторых — бесплатна (или наоборот?). В этой книге я буду ориентироваться на операционную систему Linux. Тем более, что некоторые функции PHP не работают или работают некорректно в среде Windows.

2.5. Регистрация доменного имени

Обычно при регистрации коммерческого (платного) хостинга вам предоставляется доменное имя второго уровня в доменах (как правило, в доменах RU или COM). Некоторые хостеры могут брать плату за регистрацию доменного имени, а другие — нет. Так вот, если вы заплатили за имя отдельно, значит, оно ваше. Это означает, что если вы вдруг смените хостера, ваш домен никуда не денется. А вот если вы получили имя «в подарок» от хостера, скорее всего, он зарегистрировал это имя на себя, а вам сдал его в аренду. Если вы уйдете от этого хостера, вы потеряете имя. В любом случае лучше узнать заранее, на кого именно регистрируется доменное имя.

Лучше всего обратиться к компаниям, которые занимаются регистрацией доменных имен, и зарегистрировать имя самостоятельно, а потом «прийти» к хостеру с готовым именем. В этом случае при смене хостера вы имя не потеряете.

Регистрация имени стоит недорого. Стоимость зависит от уровня домена — чем выше уровень, тем больше вы заплатите. Это означает, что домен в домене COM вам обойдется дороже, чем в домене COM.RU. Следующий фактор, который влияет на стоимость домена, это «престиж марки». Домены RU и COM являются доменами первого уровня, но второй стоит раза в два дороже первого — вот вам и престиж.

Глава 3

Интерфейс CGI

3.1. Принципы работы CGI

3.1.1. Общие положения

CGI (Common Gateway Interface — Общий Шлюзовой Интерфейс) — это набор правил и соглашений, которые должны соблюдаться различными Web-серверами при выполнении Web-приложений.

В этой главе мы рассмотрим традиционное CGI-программирование, не касаясь PHP. Это сделано преднамеренно, чтобы вы впоследствии почувствовали разницу между написанием CGI-программы и PHP-программы. Вскоре вы должны понять, почему язык PHP получил такое распространение в Web-программировании. Причем вы это узнаете не из моих слов, а исходя из собственной практики.

Предположим, что в поле «Адрес» браузера вы ввели следующий URL-адрес `http://www.server.ru/path/picture.gif`

После этого сервер отобразил запрошенную картинку. У меня есть к вам небольшой вопрос: вы уверены, что запросили именно картинку? Что я имею в виду? Ведь браузер отобразил действительно картинку в формате GIF!

На самом деле файл `picture.gif`, как и путь `path` могут вообще не существовать. Дело в том, что администратор сервера может настроить псевдонимы (alias) для любых объектов на сервере. В результате вы можете так и не узнать действительное имя ресурса. При этом все производимые над объектом манипуляции не будут чем-либо отличаться. Просто вместо его имени будет использоваться псевдоним.

Однако это еще не самое удивительное. Самое интересное может заключаться в том, что файл `picture.gif` может быть вовсе не картинкой, а программой, которая передаст браузеру картинку в формате GIF. Вот это уже настоящее проявление CGI. Если первого эффекта мы могли достичь *средствами сервера, то во втором случае без CGI нам просто не обойтись.*

Получается, что мы получили мощный механизм для формирования документов «на лету». Кроме того, данный механизм абсолютно незаметен для

пользователя, которому все равно, каким образом в его браузере появилась картинка — браузер загрузил файл или файл передала ему программа.

3.1.2. Как происходит взаимодействие сервера с браузером

Что же происходит на самом деле, когда вы вводите какой-нибудь URL в строку адреса браузера? Браузер извлекает из URL протокол (например, http) и имя сервера. Согласно полученной информации он обращается к указанному вами серверу. Как вы знаете, протокол — это набор правил, согласно которым браузер будет общаться с Web-сервером (если используется протокол http).

Компьютер — это «железо», которое ничего не может сделать без программного обеспечения. На компьютере, к которому обращается браузер должна быть установлена программа — Web-сервер, которая в действительности будет обслуживать наши запросы. Должно быть понятным, что когда мы говорим сервер, должны иметь в виду *программу*, а *не компьютер*. Прошу на это обратить особое внимание.

Браузер определил, что имя сервера есть в базе данных DNS, узнал его IP-адрес и, используя его, подключился к серверу. Сервер ждет команды от нашего браузера. Браузер посылает запрос — что-то наподобие: «Мне нужен документ /path/doc1.html». Сервер просматривает содержимое каталога `/path`, находит там файл `doc1.html` и возвращает его браузеру, как обыкновенный текст. Кроме текста сервер также возвращает тип документа. Бразуер отображает документы согласно информации о типе, переданной сервером. Например, если сервер скажет, что файл `doc1.html` является текстом, то в окне браузера мы увидим код HTML, но обычно сервер так не делает, и мы видим нормальную Web-страницу — с картинками и форматированием.

Если запрашиваемый документ не найден, сервер может возвратить документ, содержащий сообщение об ошибке или просто код ошибки, например, 500 — внутренняя ошибка сервера. В последнем случае, браузер, получив код ошибки, возвращает встроенное в него сообщение об этой ошибке.

3.1.3. Зачем нужен CGI-интерфейс и какие цели он выполняет

Интерфейс CGI выполняет функции шлюза (переходника) между различными программами, установленными на Web-сервере, и браузером пользователя. Когда браузер запрашивает определенный ресурс, программа-сер-

вер запускает программу, расположенную в каталоге cgi-bin, а она уже возвращает результат, который передается пользователю.

Примечание.

Необходимо отметить, что запросы браузера не всегда обрабатываются cgi-программами. Очень часто запрашиваемые ресурсы выдаются браузеру напрямую. То есть, если браузер запросил `http://www.server.ru/path/picture.gif`, то ему и выдается `picture.gif` безо всяких премудростей (cgi-программ, псевдонимов и т.п).

Использование CGI-программ обычно требуется, когда необходимо в браузере отобразить Web-страницу, сформированную на основе некоторых данных или на основе каких-то действий пользователя на сайте. Например, вы ввели в форму какие-то данные, а на их основе формируется выдаваемая вам страница. Или еще один пример, зачастую сайты создаются сразу на нескольких языках. Так вот CGI-программа может по IP-адресу определить вашу страну и сразу отобразить сайт на родном вам языке.

3.1.4. Немного подробнее о CGI

Давайте разберемся, что именно происходит при запуске CGI-программы. CGI-программа, как уже упоминалось, это обычная программа, выводящая что-то. Механизм CGI перенаправляет вывод программы на Web-сервер, а тот, в свою очередь, в браузер пользователя. С точки зрения программы нет ничего необычного — она «думает», что выводит информацию на экран (stdout), даже не подозревая, что ее вывод будет перенаправлен и его увидит весь Интернет.

Правда, есть одно НО. Абсолютно любая программа не может быть CGI-программой, потому что перед своим выводом она должна вывести определенные заголовки сервера — указать хотя бы тип выводимой информации. Например, если мы хотим вывести картинку, мы должны отправить заголовок `Content-type: image/gif`.

Если CGI-программе нужно передать параметры, то делается это очень просто — как и в случае с обыкновенной программой. Например, нам нужно передать строку запроса `name=Denis&email=denis@localhost` сценарию `cgi.exe`, для этого нужно вызвать программу `cgi.exe` с параметром `name=Denis&email=denis@localhost`:

```
cgi.exe name=Denis&email=denis@localhost
```

Если программа предназначена для CGI, то нужно позаботиться о взаимодействии с сервером. Простейшее взаимодействие заключается в том, что программист должен знать информацию о сервере. Данная информация передается с помощью переменных окружения.

Примерно так работает механизм CGI. Чуть позже мы рассмотрим, как превратить обыкновенную программу на языке C в CGI-программу (см. листинг 3.3).

3.1.5. Способы создания CGI-сценариев

CGI-программы очень часто называют CGI-сценариями или просто сценариями из-за того, что они написаны, как правило, на каком-нибудь языке-интерпретаторе (Perl, Bash и др.). Однако никто вам не мешает написать обыкновенную CGI-программу, например, на языке C. В этом случае у вас будет исполнимый файл (exe-файл в Windows), который будет обрабатывать данные пользователя.

Попробуем создать небольшой CGI-сценарий (см. листинг 3.1). Мы не будем писать его на языке C — ведь его еще нужно устанавливать. Напишем его на языке командного интерпретатора COMMAND.COM. По сути, это будет обыкновенный BAT-файл, который будет запущен системой.

Листинг 3.1. Файл test.bat

```
@echo off
echo Content-type: text/html
echo.
echo.
echo Hello!
```

Файл `test.bat` нужно поместить в каталог `cgi-bin` вашего Web-сервера (Web-сервер может запускать программы, находящиеся только в этом каталоге).

Рассмотрим, что делает данное приложение. Сначала наш CGI-сценарий сообщает серверу формат передаваемых данных: `Content-type: text/html`. Затем мы выводим два символа перевода строки (LF или \n) — команда «echo.» сообщает серверу, что мы больше не будем выводить никаких заголовков — и приступаем к выводу данных.

Вы, наверное, обратили внимание на расширение файла .bat. Это небольшой сценарий, обработкой которого занимается командный интерпретатор `command.com`.

В Linux нет понятия «расширение файла». Поэтому вы легко можете написать следующий сценарий на языке Bash (Bourne Again Shell):

```
#!/bin/bash
echo Hello!
```

Затем данный файл (`test.gif`) нужно поместить в каталог CGI и сделать его исполнимым:

```
chmod +x test.gif
```

Несмотря на то, что мы назвали файл `test.gif`, система запустит его, и в окне браузера вы увидите строку Hello! Точно таким же образом поступают программы, выводящие изображения: сначала они выводят формат (например, `gif`), а затем — содержимое файла в формате GIF.

Благодаря тому, что в Linux отсутствует понятие «расширение файла», мы можем достичь еще большего уровня прозрачности. В Windows, чтобы запустить программу, она должна иметь расширение .com, .exe, .bat. Понятное дело, что пользователь догадается, что он имеет дело с программой. В Linux запуск программы происходит совершенно незаметно для пользователя.

3.2. Методы передачи данных GET и POST

3.2.1. Зачем и в каком виде происходит передача данных серверу и CGI-сценариям

Программы предназначены для того, чтобы решать определенные задачи. Но кто же будет писать программу, которая решает конкретную задачу? Программы должны решать все задачи определенного класса. Например, следующая программа не является универсальной:

```
a = 2 + 2;
print a;
```

В данном исполнении программа каждый раз будет выводить одно и то же число. Сделаем программу универсальной: для этого будем передавать ей два параметра — числа, которые она должна сложить:

```
read a, b;
c = a + b;
print c;
```

Точно такая же ситуация сложилась вокруг CGI-сценариев. Для работы сценарию нужно передать определенные параметры, он их обработает и передаст ответ. Поскольку браузер не позволяет вводить параметры напрямую — с клавиатуры, он передает их с помощью форм: вы вводите параметры в форму, затем браузер складывает все параметры в одну строку — строку запроса — и передает ее сценарию.

Алгоритм работы любой CGI-программы (или сценария — как кому больше нравится) выглядит следующим образом:

1. Получить данные, переданные пользователем.
2. Обработать данные.
3. Вывести результат.
4. Завершить работу.

Передача данных пользователя в HTML, как вы уже знаете, реализуется с помощью форм. Рассмотрим небольшую форму, передающую имя пользователя (см. листинг 3.2).

Листинг 3.2. Пример формы

```
<form action= "/cgi/delete.cgi" method= "post">
   <input type= "text" name= "user_name">
   <input type= "submit" name= "go" value= "Go!">
</form>
```

Если данную форму сохранить как файл в формате HTML, то в окне браузера вы увидите поле для ввода имени и кнопку для передачи параметра на сервер (рис. 3.1).

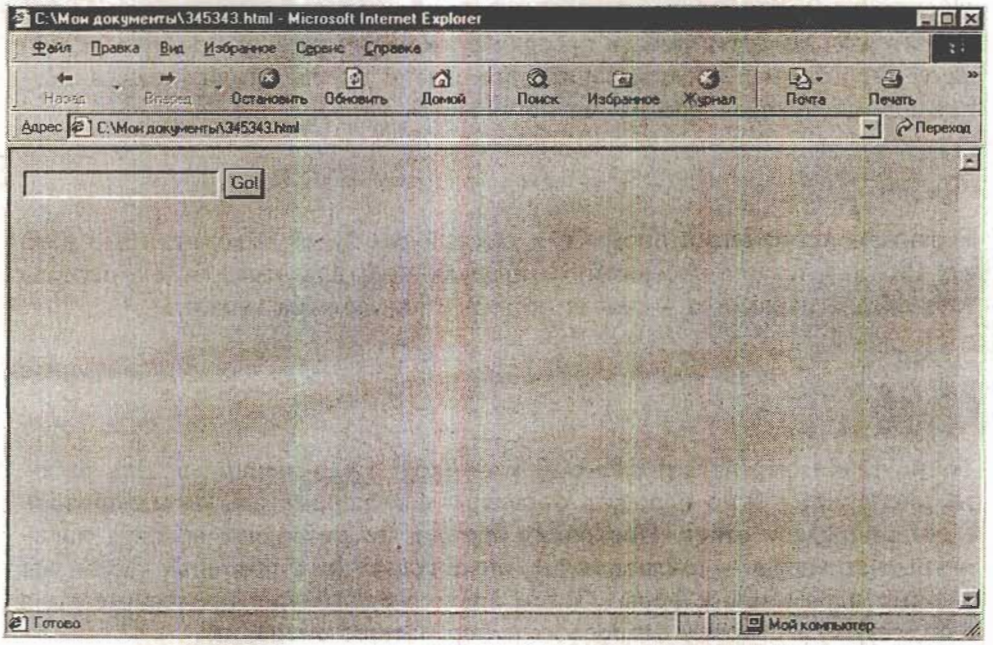


Рис. 3.1. *Форма в окне браузера*

Теперь давайте разберемся с самой формой. Когда пользователь введет имя и нажмет кнопку «Go», введенное имя будет передано сценарию `delete.cgi` в качестве параметра. Этот параметр будет называться `user_name`. Метод передачи данных задается параметром **method** тэга <form>. Сейчас мы используем метод **Get**. После того, как будет нажата кнопка «Go», в поле **Адрес** браузера будет показано примерно следующее:

```
http://localhost/cgi/delete.cgi?user_name=Denis
```

Как видите, после имени узла и пути к CGI-программе ей передаются введенные пользователем данные в виде параметров. Параметр при пересылке присоединяется к URL-адресу сценария через знак вопроса (?). Если параметров несколько, то они разделяются амперсантом (&):

```
http://localhost/cgi/delete.cgi?user_name=Denis&sex=Male
```

Метод передачи определяет, как именно данные будут переданы сценарию. «На глаз» определить какой метод используется можно следующим образом. Метод, при котором передаваемые параметры отображаются в строке браузера, является методом GET. Если вы не видите передаваемых параметров, значит, используется метод POST.

После того, как параметры переданы сценарию, задача сценария состоит в том, чтобы выполнить разбор строки, то есть выделить пары **имя_параметра=значение**. Теперь рассмотрим оба эти метода подробнее.

3.2.2. Метод GET

Формирование запроса браузером

Ранее было сказано, что взаимодействие сервера (серверной программы) и браузера осуществляется в виде запросов и ответов. Когда мы вводим в адресной строке браузера строку `string`, браузер последовательно выполняет следующие действия:

1. Выделяет из нее имя сервера и порт (если он указан).
2. Устанавливает соединение с сервером и посылает ему запрос:

```
GET string HTTP/1.0\n
.....разная информация.....
\n\n
```

В запросе `\n` означает перевод на новую строку, а `\n\n` — маркер окончания запроса. Если сервер не получит маркера окончания, то запрос он обрабатывать не будет.

После строки, начинающейся со слова GET (так называемой GET-строки) в запросе могут содержаться другие строки. Эти строки называются

заголовками, впрочем, как и сама GET-строка. Заголовков может быть довольно много. Их формирует браузер. Однако для запроса документа нужен только один GET. Чуть позже мы рассмотрим заголовки подробнее, а сейчас займемся разбором переданных сценарию параметров.

Если браузер вызывает сценарий, то, как правило, он передает ему какие-нибудь параметры:

```
GET сценарий?параметр=значение&параметр=значение HTTP/1.0\n
```

Как видите, после имени запрашиваемого документа следует знак вопроса, а за ним — строка параметров.

Формирование ответа сервером

Для формирования ответа сервера будем использовать специальный сценарий, который мы напишем на языке Си. Этот сценарий должен быть откомпилирован и помещен в каталог CGI-сценариев. Все, что будет делать сценарий, это отображать присланные ему параметры в окне браузера. При этом он не будет заниматься непосредственным разбором параметров, а только выведет все переданные параметры в браузер. Если вы не знаете языка Си, ничего страшного (хотя ничего хорошего в этом нет). Ниже приведено подробное описание сценария, так что вы без труда поймете назначение каждой строки сценария.

Листинг 3.3. Простейшее CGI-приложение

```
/* cgi.c - простейшее CGI-приложение */
#include <stdio.h>
#include <stdlib.h>

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

printf("Content-type: text/html\n\n");

printf("<html><body><h1>Привет!</h1><p>");
printf("Ваш IP: %s",Remote_IP);
printf("<p>Переданные вами параметры: %s",Query_String);
printf("</body></html>");

return 0;
}
```

Давайте разберемся, что делает сценарий. Сначала он получает значение переменной окружения REMOTE_ADDR, а потом — значение переменной QUERY_STRING. Первая содержит значение IP-адреса удаленного пользователя, а вторая — строку переданных параметров. Для обращения к переменным окружения в Си используется функция **getenv()**.

Затем сценарий выводит на стандартный вывод значения обеих переменных. Вот здесь и заключается огромная хитрость CGI. Если запустить программу просто в консоли, а не через браузер, то на экране мы увидим:

```
Content-type: text/html
<html><body><h1>Привет!</h1><p>Ваш IP: <p>Переданные вами пара-
метры: </body></html>
```

Значения переменных окружения REMOTE_ADDR и QUERY_STRING будут недоступны, так как эту программу запустили мы сами, а не сервер. Сервер, запустив программу, считает все данные, которые программа выводит на стандартный вывод (stdout) и передает программе пользователя — браузеру.

Таким образом, если запустить программу, используя браузер, то в окне браузера мы увидим (при условии, что передали такие параметры: `name=Denis&email=den@localhost`):

```
Привет!

Ваш IP: 127.0.0.1
Переданные вами параметры: name=Denis&email=den@localhost
```

Вам, как программисту, осталось сделать малое — только разобрать пары `параметр = значение`. Это значит, что вы должны вручную (в смысле сами написать программу разбора) прочитать строку параметров, создать двумерный массив, содержащий имена параметров и их значения. Потом, когда вам понадобится имя параметра, должны будете перебрать все элементы массива в поисках нужного вам имени параметра. Найдя его, вы узнаете его значение. Но вам не нужно этого делать! Почему? Язык PHP сам выполняет разбор параметров командной строки. Для обращения к значению параметра можно использовать имя параметра:

```
echo "Ваше имя: $name";
echo "Email: $email";
```

Кроме того, как видно из этого сценария, PHP сам создает переменные, которые называются так же, как и переданные параметры. Это значит, что если сценарию передать параметр `email`, то в процессе работы сценария (а именно — сразу после его запуска) появится переменная `$email`, с которой можно будет работать, как с обыкновенной переменной.

3.2.3. Метод POST

Мы только что рассмотрели метод передачи GET и все, что с ним связано. Теперь приступим к следующему методу — методу POST. Как я уже говорил, если используется метод POST, переданные сценарию параметры не отображаются в окне браузера. Да и сама передача происходит иначе:

```
POST сценарий HTTP/1.0\n
Content-length: 5\n
\n
Hello
```

Как видите сам блок пересылаемых данных находится уже после маркера окончания. Благодаря этому достигается принципиальная особенность метода POST. Эта особенность заключается в том, что при такой передаче данных сервер никак их не интерпретирует, а сразу передает сценарию. Это означает, что сценарий будет запущен сразу же после окончания заголовков (\n\n) и сервер просто передает ему POST-данные. Потом сценарий должен самостоятельно прочитать определенное число байтов, указанное в заголовке **Content-length** (длина слова Hello составляет пять символов). Поэтому при использовании метода POST всегда необходимо указывать значение переменной **Content-length**.

Но вы можете не беспокоиться: всю грязную работу за вас выполнит PHP, а вы сможете обращаться к переданным параметрам так же, как и в прошлом случае: параметры, переданные методом POST, будут преобразованы в переменные, с которыми вы сможете работать в своем сценарии.

Метод POST обычно используется для передачи больших объемом данных, например, при загрузке файлов на сервер или же, когда нужно скрыть значения передаваемых параметров от глаз пользователя. Дело в том, что передаваемые параметры и их значения не отображаются в окне браузера, как это происходит с методом GET.

3.3. Переменные окружения CGI

Мы уже знакомы с двумя переменными окружения — REMOTE_ADDR и QUERY_STRING. Переменные окружения в языке PHP можно использовать, как самые обыкновенные переменные:

```
echo "IP: $REMOTE_ADDR";
echo "Параметры: $QUERY_STRING";
```

В этом пункте мы поговорим об остальных переменных окружения, которые можно разбить на четыре большие группы:

- Формируемые сервером переменные.
- Специальные переменные сервера Apache.
- Переменные HTTP-полей запроса.
- Переменные SSL-соединения (защищенного соединения).

Первые три группы мы рассмотрим сейчас, а последнюю — в главе «Защищенные соединения». Все эти переменные представлены в табл. 3.1...3.3, соответственно.

Формируемые сервером переменные *Таблица 3.1*

Переменная	Описание
AUTH_TYPE	Используемая схема аутентификации. Обычно 'BASIC'
CONTENT_LENGTH	Длина содержимого (например, при передаче POST-данных)
CONTENT_TYPE	MIME-тип содержимого, например, text/html
GATEWAY_INTERFACE	Версия CGI, например, CGI/1.1
PATH_INFO	HTTP-путь к сценарию
PATH_TRANSLATED	Полный путь к сценарию
REMOTE_ADDR	IP-адрес запрашивающего компьютера-клиента
REMOTE_HOST	Доменное имя запрашивающего компьютера (если доступно). Доменное имя определятся Web-сервером с помощью службы DNS. Директива HostnameLookups сервера Apache разрешает (или запрещает) преобразование IP-адреса в доменное имя
REMOTE_PORT	Порт, закрепленный за браузером для получения ответа от сервера
REMOTE_USER	Имя пользователя, прошедшего аутентификацию
QUERY_STRING	Строка переданных серверу параметров
SERVER_ADDR	IP-адрес сервера
SERVER_NAME	Доменное имя сервера. Определяется директивой ServerName файла конфигурации
SERVER_PORT	TCP-порт Web-сервера. Обычно 80
SERVER_PROTOCOL	Версия протокола HTTP. Например, HTTP/1.1
SERVER_SOFTWARE	Программное обеспечение сервера
SCRIPT_NAME	HTTP-путь к сценарию
SCRIPT_FILENAME	Имя файла сценария в файловой системе сервера (физический путь). Например, /var/www/cgi-bin/script.cgi

Специальные переменные сервера Apache *Таблица 3.2*

Переменная	Описание
DOCUMENT_ROOT	Физический путь к корневому WWW-каталогу сервера. Например, /var/www/html
SERVER_ADMIN	Адрес электронной почты администратора сервера
SERVER_SIGNATURE	Подпись сервера. Например, «Apache/1.3.3 Server at www.firma.ru Port 80»

Переменные HTTP-полей запроса *Таблица 3.3*

Переменная	Ог.исание
HTTP_HOST	Имя виртуального хоста, которому адресован запрос
HTTP_USER_AGENT	Программное обеспечение удаленного пользователя. Обычно данная переменная содержит название и версию браузера
HTTP_ACCEPT	Список поддерживаемых клиентов типов содержимого. В последнее время вместо списка браузеры возвращают значение *.*, что означает «все типы»
HTTP_ACCEPT_LANGUAGE	Список поддерживаемых языков в порядке предпочтения, например, ru,en
HTTP_ACCEPT_ENCODING	Список поддерживаемых методов сжатия
HTTP_ACCEPT_CHARSET	Список поддерживаемых кодировок
HTTP_CONNECTION	Тип соединения. Возможны два варианта: • Keep-alive, если после ответа на запрос не нужно разрывать соединения; • Close, если нужно закрыть соединение сразу после ответа на запрос
HTTP_REFERER	Значение поля REFERER. В этом поле браузер передает URL-ресурса, который ссылался на наш сервер. Например, если пользователь перешел наш сайт со страницы http://dkws.narod.ru/index.html, то значение поля REFERER будет http://dkws.narod.ru/index.html
HTTP_X_FORWARDED_FOR	Если пользователь работает через прокси-сервер, то в этом поле будет IP-адрес узла, обратившегося к прокси-серверу. Если данное поле уже содержит значение, то новое значение будет добавлено через запятую

3.4. Заголовки HTTP

3.4.1. Описание основных заголовков HTTP

Заголовки HTTP используются для «общения» браузера и Web-сервера, например, когда браузер запрашивает какой-нибудь документ, он посылает заголовок GET, а когда сервер возвращает тип документа, то он делает это ни как-нибудь, а в заголовке Content-type.

В этой главе мы уже говорили о заголовках протокола HTTP, но не рассматривали их достаточно подробно.

Заголовок Accept

Заголовок Accept предназначен для информирования сервера о типах данных, которые поддерживаются клиентом. Используется переменная окружения HTTP_ACCEPT. Пример использования:

```
Accept: text/html, text/plain, image/jpeg
```

В последнее время вместо списка указывается значение *.*, что означает «все типы».

Заголовок Content-type

Данный заголовок предназначен для идентификации типа передаваемых данных. При этом заголовок `Content-type` использует переменную окружения CONTENT_TYPE. Обычно для этого заголовка указывается значение `application/x-www-form-urlencoded`. Таким образом, указывается формат, в котором все управляющие символы (т.е. символы, не являющиеся алфавитно-цифровыми) специальным образом кодируются. О некоторых других MIME-типах вы можете узнать в п. 4.6.

Пример: `Content-type: text/plain`

Заголовок Content-length

Этот заголовок содержит строку, в которой записана длина передаваемых данных в байтах при использовании метода передачи POST. За заголовком закреплена одноименная переменная CONTENT_LENGTH.

Заголовок Cookie

В этом заголовке хранятся все Cookies. О Cookies более подробно мы поговорим позже, а пока лишь отметим, что данный заголовок использует переменную окружения HTTP_COOKIE. Для установки Cookies используется заголовок `Set-Cookie`.

Заголовок GET

На этом заголовке мы останавливались ранее. Формат заголовка следующий:

```
GET сценарий?параметры HTTP/1.0
```

Заголовок GET использует следующие переменные окружения:

- REQUEST_URI — запрашиваемый идентификатор ресурса (именно URI, а не URL).
- QUERY_STRING — передаваемые сценарию параметры.
- REQUEST_METHOD — метод передачи информации. В данном случае эта переменная будет содержать значение GET.

Заголовок Location

Получив заголовок `Location` вместе с указанным в нем URL, сервер обязан немедленно перейти по указанному URL, не дожидаясь, пока тело документа загрузится:

Пример: `Location: http://dkws.narod.ru`

Заголовок POST

Этот заголовок использует те же переменные окружения, что и заголовок GET (переменная REQUEST_METHOD содержит значение POST). Как уже отмечалось ранее, данные методом POST можно передавать в конце всех заголовков.

Формат заголовка POST таков:

```
POST сценарий?параметры HTTP/1.0
```

Заголовок Pragma

Данный заголовок используется для различных целей, одна из которых — это запрет кэширования документа. Пример: `Pragma: no-cache`

Заголовок Server

Данный заголовок содержит название и версию программного обеспечения сервера. Например:

```
Server: Apache/1.3.23 (Unix)  (Red-Hat/Linux) mod_ssl/2.8.7
OpenSSL/0.9.6b DAV/1.0.3 PHP/4.3.4 mod_perl/1.26 configured
```

Заголовок Referer

С помощью этого заголовка мы можем узнать ссылающийся на нас сервер. Это помогает выявлять обратные ссылки, что особенно бывает полезно при анализе эффективности сетевой рекламы. Переменная окружения: HTTP_REFERER.

Заголовок User-Agent

Уточняет версию браузера, например, User-Agent: Mozilla/5.0 (compatible; Konqueror/3.0.0-10; Linux). Переменная окружения: HTTP_USER_AGENT.

3.4.2. Некоторые особенности

Ознакомившись с названиями заголовков и соответствующим им переменных окружения, вы, наверное, заметили одно просто правило, согласно которому присваиваются имена переменных окружения:

- Все символы переводятся в верхний регистр.

- В начале имени добавляется HTTP_.
- Символы «–» заменяются знаком подчеркивания «_».

Некоторые заголовки не используют переменные окружения или имена переменных не соответствуют вышеприведенному правилу, но ведь правила и существуют для того, чтобы их нарушать...

3.5. Коды ответов HTTP

Знание кодов возврата в большей степени полезно для администратора, но их знание не повредит и рядовому Web-программисту, поскольку последний является администратором своего домашнего Web-сервера.

Не имеет значения, являетесь ли вы администратором вашего Web-сервера, например, Apache, или вы простой пользователь Интернета. Скорее всего, уже знакомы с некоторыми кодами возврата протокола HTTP. По крайней мере, код `404 – Not Found` — знают все. Иногда с этой ошибкой сталкиваешься несколько раз в день.

На самом деле, ошибка 404 является самой распространенной не только потому, что, например, при работе над сложным проектом Web-мастер забыл правильно изменить ссылки или нужный документ был попросту удален. В большинстве случаев ошибка 404 является «той единственной», которая показывается пользователю. Например, код нормального завершения операции (то есть страница найдена) — 200, но вы ведь его не видите, когда получаете страницу. А иногда все ошибки из диапазона 400...415 вообще расцениваются как ошибка 404.

Кроме уже известных нам кодов 404 и 200 определено множество других кодов возврата. Всего определено пять категорий кодов, каждая из которых может содержать 100 кодов возврата.

Коды диапазона 100...199 называются информационными. С помощью этих кодов возвращается различная информация, относящаяся к запросу.

Диапазон 200...299 является «диапазоном успеха». Коды, которые относятся к этому диапазону, сообщают об удачном завершении той или иной операции. К этому диапазону относятся также коды, сообщающие, что запрашиваемый файл пуст (no content), и сообщающие, что запрос был принят.

Коды 300...399 являются кодами перенаправления. Коды 400...499 являются ошибками клиента. Например, пользователь может запросить несуществующий документ. К этому же диапазону относятся ошибки «Forbidden» и «Unauthorized». И первая, и вторая сообщают, что у вас нет доступа к какому-нибудь ресурсу, но в первом случае — доступ просто запрещен, а во втором — вы являетесь неавторизированным клиентом.

Коды 500...599 называются ошибками сервера. Данные ошибки возникают при внутренней ошибке сервера или же при неправильном его конфигурировании. Во втором случае сервер просто не может правильно возвратить ответ на запрос. Этот тип ошибок встречается обычно, когда ошибку совершает сценарий, расположенный на вашем сервере.

Некоторые интерпретаторы сценариев, например, PHP никогда не допускают такой ошибки. Если в вашем PHP-сценарии есть ошибка, интерпретатор PHP просто возвратит клиенту сообщение об этой ошибке. Со стороны сервера это будет выглядеть как нормальный ответ клиенту, поэтому ошибка 5xx при использовании PHP у вас практически никогда не возникнет. В отличие от PHP, интерпретатор Perl генерирует ошибку 500 и записывает сообщение об ошибке в файл протокола сервера HTTP.

Краткий (основной) список кодов возврата приведен в табл. 3.4.

Таблица кодов возврата HTTP *Таблица 3.4*

Код	Описание
100	Продолжить (Continue)
101	Переключение протоколов (switching protocols)
200	OK
201	Создан (created)
202	Принят
203	Неавторитетная информация (non-authoritative information)
204	Файл пуст (no content)
205	Сброс содержимого (reset content)
206	Частичное содержимое (partial content)
300	Многократный выбор (multiple choices)
301	Перемещен постоянно (moved permanently)
302	Перемещен временно (moved temporarily)
303	Смотри другой (see other)
304	Неизмененный (not modified)
305	Использовать прокси-сервер (use proxy)
400	Неправильный запрос (bad request)
401	Неавторизирован (unauthorized)
402	Payment Required
403	Нет доступа (forbidden)
404	Не найден
405	Метод не разрешен (method not allowed)
406	Неприемлемый (not acceptable)
407	Требуется аутентификация на прокси-сервере (proxy authentication required)
408	Превышение тайм-аута запроса (request time-out)

Таблица кодов возврата HTTP *Таблица 3.4 (продолжение)*

Код	Описание
411	Требуется длина (при использовании метода POST)
412	Не выполнено предыдущее условие (precondition failed)
413	Объект запроса слишком велик (request entity too large)
414	Запрашиваемый URL слишком велик (request-URL too large)
415	Неподдерживаемый тип информации (unsupported media type)
500	Ошибка сервера
501	Не реализован (not implemented)
502	Неправильный шлюз (Bad Gateway)
503	Нехватка ресурсов (out of resources)
504	Превышен тайм-аут шлюза
505	Неподдерживаемая версия HTTP (HTTP version not supported)

3.6. Типы MIME

Типы MIME используются для идентификации типа содержимого. Благодаря этому приложения смогут определять, какого вида данные присланы и в каком соответствии сети проводить их обработку. Перечень всех типов MIME, поддерживаемых вашей системой, находится в файле `/etc/mime.types` (`C:\Program Files\Apache Group\Apache\conf` — в Windows). Тип MIME указывается в формате: тип/подтип. Вот основные типы MIME:

- **application** — приложение (программа или документ, связанный с программой). Например, application/rtf.
- **audio** — аудиофайл (audio/mpeg).
- **image** — графический файл (audio/png).
- **text** — текст (text/html — в формате HTML, text/plain — простой).

MIME-тип пересылаемых по сети данных указывается в заголовке **Content-type**.

3.7. PHP и CGI

PHP является основным конкурентом CGI. Сейчас, хоть и написано много CGI-приложений, они постепенно вытесняются PHP. Почему? Да потому, что PHP предназначен именно для Web-программирования — вам не нужно выдумывать свои собственные функции, например, для разбора строки, получения значений переменных окружения, и т.д. При ис-

пользовании, к примеру C, вам придется это делать самому. А зачем заново изобретать велосипед?

В PHP переданные сценарию параметры автоматически станут переменными сценария, с которыми можно будет работать, как с обыкновенными переменными. То же самое происходит с переменными окружения сервера. Вам не нужно изобретать функции для работы с различными базами данных — MySQL, PostgresSQL, Sybase, Informix и др. Все это уже есть в PHP.

К тому же PHP является полноценным языком программирования, синтаксис которого очень напоминает C, что способствует быстрому освоению языка. Библиотека стандартных функций PHP очень богата — все, что нужно программисту, всегда под рукой.

Глава 4

Описание, установка и настройка необходимого программного обеспечения

4.1. Выбор необходимого программного обеспечения

4.1.1. Выбор операционной системы

Для начала необходимо определиться с операционной системой. По большому счету все равно какую операционную систему вы будете использовать. Ведь PHP работает и под Windows и под Linux. Для абсолютного большинства проектов так оно и есть. Вы можете спокойно создавать PHP-сценарии на своем Windows-компьютере, а затем отправлять его на сервер. При этом также не заморачиваясь по поводу того, под какой операционной системой работает сервер (Linux или Windows). Главное, чтобы на нем поддерживались PHP и MySQL.

Однако, если вы всерьез решили заняться созданием профессиональных сайтов на основе PHP-сценариев, то вам не обойтись без учета того, на какой операционной системе работает сервер. При этом стоит иметь ввиду, что PHP изначально создавался под Unix-подобные операционные системы — Linux, FreeBSD, и т.п. Конечно, PHP и на Windows-хостинге работает исправно. Но существуют некоторые детали и возможности, которые нельзя учесть в силу разности ОС Windows и ОС Linux (Unix). Таким образом, использование PHP под Windows не позволяет полностью реализовать, как возможности операционной системы, так и возможности PHP. Прошу заметить, что здесь идет речь о профессиональных тонкостях, знание которых подразумевает серьезную подготовку и глубокое понимание функционального устройства ОС. Однако именно эти тонкости отличают профессиональный проект от непрофессионального.

Под Windows существует свой аналог PHP, который называется ASP (Active Server Pages). Он полностью ориентирован на Windows и позволяет учитывать и использовать все особенности этой операционной системы. При этом в качестве HTTP-сервера вместо Apache используется IIS (Internet Information Server) — специальный продукт компании Microsoft.

Итак, профессиональное использование PHP ориентировано на Linux (Unix). Поэтому желательно иметь соответствующий хостинг. Тем не менее, в книге будут рассмотрены оба варианта установки, настройки и использования PHP: и под Linux, и под Windows.

Желательно также, чтобы на сервере и на вашем компьютере (на котором вы будете создавать и тестировать свои PHP-сценарии) стояла одинаковая операционная система. Иногда бывает так, что написанная PHP-программа отлично работает на вашем домашнем Windows-компьютере, но отказывается работать или работает некорректно под Linux на сервере.

Ранее было показано, что для использования PHP-проектов желательно использовать Linux (Unix)-хостинг. Соответственно, желательно (для большей совместимости), чтобы на вашем домашнем компьютере стояла ОС Linux. Однако это не является необходимым. Если у вас установлена Windows и вы не хотите от нее отказываться — на здоровье. Я не буду вас агитировать, хотя я свой выбор в пользу Linux давно уже сделал. Еще раз повторяю, что в книге будут рассмотрены и Linux-вариант и Windows-вариант. Однако, если вы рано или поздно решите установить у себя Linux, то рекомендую вам свою книгу «Самоучитель Linux. Установка, настройка, использование», вышедшую в издательстве «Наука и Техника». В этой книге вы найдете всю необходимую информацию для работы с этой операционной системой.

Примечание.

На самом деле в издательстве «Наука и Техника» вышла еще одна моя книга «Linux-сервер своими руками», но она уже является более сложной и предназначена тем, кто хочет создать и настроить свой сервер на основе Linux. Ориентирована она на будущих администраторов. Однако, поскольку вы при тестировании PHP-сценариев у себя на компьютере будете моделировать сервер, то вам эта книга может очень пригодиться. В любом случае вам будет полезно узнать, как работает сервер вашего хостера.

4.1.2. Выбор программ

Для того чтобы изучить язык PHP, вам понадобятся следующие программы:

- Web-сервер Apache — это и есть та самая программа, которая превратит ваш домашний компьютер в Web-сервер.
- Собственно сам PHP 4 или PHP 5 — данная книга как раз об этом, поэтому это нужно установить в первую очередь.
- *Сервер баз данных MySQL — практически ни один серьезный проект* не обходится без использования баз данных, поэтому не нужно откладывать установку MySQl «на завтра».

- Дополнительные модули для Apache (mod_php) и PHP (php_mysql) — первый модуль «встраивает» PHP в сервер Apache, благодаря чему Apache обрабатывает PHP-сценарии «на лету». Второй модуль добавляет в PHP поддержку MySQL — без него вы не сможете работать с MySQL.

4.2. Web-сервер Apache

4.2.1. Какие функции выполняет Apache (и зачем он нам сейчас нужен на клиентской машине)

Сервер **Apache** разработан и поддерживается организацией Apache Project. Первоначально сервер Apache был разновидностью сервера Web-сервера NCSA, разработанного в Национальном центре разработок суперкомпьютеров Иллинойского университета. Возникновение Apache было связано с тем, что в 1994 году ушел из проекта главный разработчик NCSA, оставив многих последователей самостоятельно разбираться в своем сервере. Со временем начали появляться исправления и дополнения к серверу NCSA — так называемые patches (патчи). А в апреле 1995 года вышла первая версия сервера Apache, основанного на версии 1.3 сервера NCSA. Первая версия Apache просто вобрала в себя все известные исправления сервера NCSA. Да и само название Apache именно от этого — «А PatCHy». Позже Apache стал самостоятельной разработкой. Сейчас сервер Apache поддерживается группой программистов-добровольцев Apache Group.

Сервер Apache разрабатывался для ОС Linux и Unix, но со временем были выпущены его версии и для ОС Windows и OS/2.

Хочу также отметить, что кроме Apache, для ОС Linux существуют другие Web-серверы: Red Hat Secure Server, Apache-SSL, Netscape Enterprise Server, и др.

Помните, еще в первой главе было сказано, что функции Web-сервера выполняет не компьютер, а программа, установленная на нем? Так вот Apache и является той самой программой. Когда браузер пользователя подключается к Web-серверу и посылает заголовок GET (это просьба передать файл), его запрос обрабатывает именно Apache. Apache проверяет, есть ли указанный в заголовке GET файл, и, если есть, отправляет его вместе с заголовками браузеру.

Apache — это своего рода стандарт Web-сервера в Интернете. Его основным конкурентом является IIS (Internet Information Server) от Microsoft, который работает под Windows. Apache, хотя и существуют его версии под Windows, преимущественно устанавливается на Unix-подобных операционных системах — Linux и FreeBSD. Нужно также заметить, что боль-

шинство хостеров используют именно Apache, а не IIS. IIS устанавливается в основном на корпоративных серверах, работающих под Windows. Сейчас мы не будем останавливаться на характеристиках и особенностях работы Apache и тем более не будем сравнивать его с IIS. Лучше перейдем сразу к его настройке и установке.

Сервер Apache нам просто необходим — без него мы не сможем протестировать наши сценарии. Конечно, можно тестировать и без сервера, но тогда нужно будет представить себя браузером, или же использовать сервер хостера, но в этом случае затраты на Интернет сильно возрастут.

4.2.2. Конфигурационные файлы сервера Apache

Сервер Apache имеет три конфигурационных файла: `httpd.conf`, `srm.conf`, `access.conf`. Обычно эти файлы находятся в каталоге `/etc/httpd/conf`. Вся настройка сервера заключается в редактировании этих трех файлов. Рассмотрим, какие функции выполняют эти файлы.

Файл `httpd.conf` — это основной файл конфигурации сервера. В нем содержится техническое описание работы сервера. В файле `srm.conf` задаются параметры документов, которые размещены на сервере. Файл `access.conf` содержит параметры доступа к серверу.

Начиная с версии 1.3, рекомендуется все директивы, которые раньше находились в файлах `srm.conf` и `access.conf`, помещать в файл `httpd.conf`. Я использую сервер Apache версии 2.0, который входит в состав дистрибутива Linux Red Hat 9.0. В этой версии существует еще несколько конфигурационных файлов: `apache-mime.types`, `vhosts/vhosts.conf`, `vhosts/VirtualHomePages.conf`, `vhosts/DynamicVHosts.conf`. В файле `apache-mime.types` содержатся типы MIME, поддерживаемые сервером Apache. Файлы `vhosts.conf`, `VirtualHomePages.conf`, `DynamicVHosts.conf` относятся к конфигурированию виртуальных Web-серверов, о которых речь пойдет немного позже.

4.2.3. Конфигурационный файл httpd.conf и его директивы

Директивой называется параметр сервера, влияющий на его поведение. Директивы бывают простыми и сложными, которые также называются блоками директив. Простые директивы изменяют только одно свойство сервера, например, его имя, а блоки директив позволяют изменять сразу несколько свойств какого-нибудь объекта, например, каталога.

Как уже отмечалось ранее, этот файл содержит практически все директивы, необходимые для работы сервера. Директивы конфигурационного файла сервера Apache можно условно разделить на такие группы:

1. **Общие.** К общим директивам относятся глобальные директивы, влияющие на работу всего Web-сервера. Это директивы ServerName, ServerType, Port, User и Group, ServerAdmin, ServerRoot, PidFile, DocumentRoot, UserDir.
2. **Директивы протоколирования**: ErrorLog, TransferLog, HostnameLookups.
3. **Директивы ограничения доступа**: AllowOverride, Options, Limit.
4. **Директивы управления производительностью**: StartServers, MaxSpareServers, MinSpareServers, а также директива CacheNegotiatedDocs.
5. **Директивы обеспечения постоянного соединения с клиентом**: Timeout, KeepAlive, KeepAliveTimeout.
6. **Директивы настройки отображения каталогов**. Оформить отображение каталогов можно с помощью директив настройки отображения каталогов: DirectoryIndex, FancyIndexing и AddIconByType.
7. **Директивы обработки ошибок**. Директивой обработки ошибок HTTP-сервера является директива ErrorDocument. С ее помощью можно установить реакцию на любую ошибку сервера, например, на ошибку 404 (документ не найден).
8. **Директивы перенаправления**: Redirect, Alias и ScriptAlias.
9. **Директивы для работы с многоязычными документами**: AddLanguage и LanguagePriority.
10. **Директивы обработки MIME-типов**. Настроить свой сервер для обработки различных MIME-типов можно с помощью директив DefaultType, AddEncoding, AddType, AddHandler и Action.
11. **Директивы создания виртуальных узлов**: VirtualHost, Listen, BindAddress.

Все эти директивы редактировать вам вряд ли придется — нужно будет задать только значения директив **ServerName** и **ServerAdmin**. По умолчанию остальные директивы содержат вполне разумные значения. Далее приведено описание директив, используемых в файле `httpd.conf`.

Директивы управления доступом к отдельным каталогам вынесены в отдельный раздел (п. 4.2.4).

Общие директивы

Общие директивы изменяют глобальные параметры сервера — его имя, тип, порт, адрес администратора. Значения, указанные глобальными директивами, влияют на работу всего сервера.

- **ServerName** — директива, которая определяет имя сервера Apache. Причем, здесь должно быть задано официальное имя сервера в *таком виде, в котором оно должно появляться в строке адреса* браузера. Данное имя должно быть зарегистрировано в сервере DNS вашей сети.

- **ServerType** — директива, которая определяет тип сервера. По умолчанию используется значение **standalone**. Если вы хотите достичь максимальной производительности вашего Web-сервера, не изменяйте эту опцию.
- **ServerAdmin** — директива, которая задает электронный адрес Web-мастера вашего Web-узла. Если возникнут какие-то проблемы, связанные с работой сервера, то по этому адресу будет отправлено соответствующее сообщение. Обычно используется значение `webmaster@Your_Host.com`. Пользователь webmaster, как правило, не существует реально в системе. Для определения имени (псевдонима) webmaster используется файл псевдонимов электронной почты `/etc/aliases`. Данный файл используется для определения псевдонимов пользователей. Формат файла `/etc/aliases` следующий:

  псевдоним: имя

  После модификации этого файла нужно ввести команду **newaliases**. Данную команду нужно вводить, зарегистрировавшись в системе, как суперпользователь. В Windows данного файла нет, поэтому вам ничего не нужно вводить. В Windows псевдонимы создаются с помощью программы настройки почтового сервера, который установлен и используется.
- **Port** — директива, задающая номер порта, который будет использоваться для установки соединения. По умолчанию используется порт 80. Если вы хотите запустить сервер Apache с использованием этого или любого другого порта, номер которого меньше 1024, вы должны обладать правами суперпользователя. Но даже если у вас нет таких прав, вы можете запустить сервер для работы с портом, номер которого превышает значение 1024. Обычно используется номер 8080 или 8000.
- **User** и **Group**. Директивы **User** и **Group** определяют идентификаторы пользователя и группы, от имени которых будет работать сервер. Данные идентификаторы присваиваются серверу, если он запущен в автономном режиме. Можно использовать, как имена пользователей, так и их числовые эквиваленты — UID. По умолчанию используется имя пользователя *nobody*. Из соображений безопасности не рекомендуется изменять это значение и присваивать имя реального пользователя. В этом случае Web-сервер получит доступ только к тем файлам, которые разрешены для чтения для всех пользователей. Нужно заметить, что указанный пользователь и группа должны существовать в вашей системе. Ни в коем случае не запускайте сервер от имени пользователя root!
- **ServerRoot** — в этой директиве указывается местонахождение файлов конфигурации сервера Apache. По умолчанию для этих целей используется каталог `/etc/httpd`.

- **PidFile** — с помощью этой директивы указывается имя файла, в котором исходный процесс сервера будет регистрироваться. Этот файл содержит свой идентификатор процесса (PID). Данную информацию можно использовать для останова или перезапуска сервера при написании собственных сценариев. Данный файл будет создан, только если сервер Apache запущен в автономном режиме.
- **DocumentRoot** — директива, определяющая местонахождение корневого каталога документов вашего сервера.
- **UserDir** — эта директива задает названия подкаталога в домашнем каталоге пользователя, из которого берутся документы. В этом случае, вы активизируете возможность использования пользовательских каталогов. Если вы не хотите включать эту возможность, укажите UserDir DISABLED. Более подробно эта директива будет рассмотрена позже.

Директивы протоколирования

Директивы протоколирования управляют процессом протоколирования работы сервера. С помощью этих директив вы можете определить, что нужно протоколировать (записывать в журналы), а что — нет.

- **HostnameLookups on | off**. Сервер Apache ведет журнал доступа других компьютеров. Если вы включите данную опцию **(on)**, то в журнал будет записано доменное имя компьютера-клиента. Если эта опция выключена **(off)**, в журнал будет записан IP-адрес клиента. Включение данной опции замедляет работу сервера, так как требуется дополнительное время на ожидание ответа от сервера DNS.
- **ErrorLog** и **TransferLog** — эти директивы определяют расположение журналов сервера Apache. Обычно для этих целей используется каталог `/etc/httpd/logs`, который является ссылкой на каталог `/var/log/httpd` или на любой другой. В журнале **errorlog** протоколируются диагностические сообщения, а также сообщения об ошибках, которые порождают CGI-сценарии. В журнале **trasferlog** протоколируются запросы клиентов. Если включена директива **HostNameLookups**, то вместо IP-адреса клиентов будут регистрироваться имена компьютеров.

Директивы управления производительностью

Поэкспериментировав с этими директивами и выбрав оптимальные значения, можно добиться существенного повышения производительности вашего сервера.

Сервер Apache для каждого соединения запускает отдельную копию, которая будет обрабатывать запросы клиента. Управлять запущенными копиями позволяют директивы **StartServers**, **MinSpareServers**, **MaxSpareServers**.

- **StartServers, MaxSpareServers, MinSpareServers.** Как уже отмечалось выше, для каждого нового соединения создается новая копия процесса сервера. Директива **StartServers** задает количество копий, которые будут созданы при запуске исходной копии сервера. При этом исходная копия сервера получает запросы и передает их свободным копиям. Это позволяет равномерно распределить нагрузку между отдельными процессами и повысить производительность сервера, однако на практике все не так хорошо, как хотелось бы. Существенного прироста производительности можно добиться только в случае большой загрузки сервера. По умолчанию запускается пять копий сервера.

  Если число поступающих запросов превышает количество запущенных копий сервера, запускаются дополнительные процессы-серверы. Эти процессы не завершаются после обработки своего запроса, а продолжают находиться в памяти. Директива **MaxSpareServers** позволяет указать максимальное число таких процессов. Если это количество превышено, то лишние процессы завершаются. Если количество «серверов на подхвате» меньше, чем задано директивой **MinSpareServers**, запускаются дополнительные копии.

  Для работы этих директив необходимо, чтобы сервер был запущен в автономном режиме.

- **CacheNegotiatedDocs.** Данная директива позволяет прокси-серверу, например, SQUID, не кэшировать документы, которые не генерируются автоматически, то есть в процессе выполнения различных сценариев. Согласно протоколу HTTP/1.0, сервер Apache с каждым пакетом посылает заголовок **Pragma: no-cache** прокси-серверу, что позволяет отключить кэширование документов (в протоколе HTTP/1.1 вместо **Pragma** используется **Cache-Control**). Если вы включите данную директиву, то разрешите прокси-серверу кэшировать документы. К сожалению, далеко не все прокси-серверы отключают кэширование после получения данного заголовка. При написании своего CGI-сценария, вам, скорее всего, придется самому выводить заголовок Pragma (или Cache-Control) и мета-теги, которые указывают на дату последнего обновления документа.

Директивы обеспечения постоянного соединения с клиентом

Данные директивы обеспечивают постоянное соединение с клиентом, а также управляют параметрами установленного соединения.

- **Timeout** — задает промежуток времени в секундах, в течение которого сервер продолжает попытки возобновления приостановленной передачи данных. Значение директивы **Timeout** распространяется не только на передачу, но и на прием данных. Если вам нужно получать большие файлы, рекомендую увеличить данное значение.

- **KeepAlive** — разрешает постоянные соединения, то есть такие соединения, в которых производится более одного запроса за один раз.
- **KeepAliveTimeOut** — данная директива определяет таймаут для постоянного соединения.
- **MaxClients**. Иногда поступающих запросов настолько много, что компьютеру не хватает ресурсов для загрузки новых копий сервера в память и их выполнения. Директива **MaxClients** (значение по умолчанию — 150) определяет максимальное число копий сервера, которые могут выполняться одновременно.
- **MaxRequestsPerChild**. После обработки определенного количества запросов, указанного в директиве **MaxRequestsPerChild**, копия сервера завершается, а вместо нее запускается новая.

Директивы создания виртуальных хостов

Довольно часто пользователь не может позволить себе такую роскошь, как собственный физический Web-сервер. То есть отдельный компьютер, постоянно подключенный к Интернет, на котором запущена программа Web-сервер — Apache. Поэтому современные Web-серверы поддерживают виртуальные хосты. У вас будет собственное имя, например, `www.firma.ru`, но в действительности ваши файлы не будут помещены на отдельном компьютере `www.firma.ru`, а будут находиться на сервере провайдера (или хостера) — `www.isp.ru`.

Благодаря технологии виртуальных узлов один сервер хотинг-провайдера обслуживает сотни или даже тысячи сайтов.

На своем домашнем Web-сервере вы также можете создать один или несколько виртуальных узлов, если, конечно, вам это нужно.

- **Listen.** Позволяет вам связывать Apache с определенным IP-адресом и (или) дополнительными портами.
- **BindAddress**. Данная директива используется для поддержки виртуальных хостов и применяется, чтобы сообщить серверу, какой IP-адрес следует прослушивать. Значением данной директивы может быть «*» (любой адрес), IP-адрес или полное имя домена.

Блок директив **VirtualHost** будет рассмотрен позднее, при описании создания и использования виртуальных хостов (см. п. 4.2.6).

Директивы настройки отображения каталогов

Данные директивы позволяют «украсить» ваш Web-сервер, например, сопоставить значок с типом файла или определить оформление каталога.

- **DirectoryIndex** — позволяет задать название документа, который будет возвращен по запросу, который не содержит имя документа. С помощью данной директивы можно задать несколько имен файлов.

Значениями по умолчанию являются: `index.html`, `index.php`, `index.htm`, `index.shtml`, `index.cgi`, `Default.htm`, `default.htm`, `index.php`. Например, если вы введете в строке адреса браузера `http://localhost`, то будет возвращен один из указанных в директиве **DirectoryIndex** документов.

- **FancyIndexing**. При получении запроса, не содержащего имя документа, сервер передаст один из файлов, указанных в директиве **DirectoryIndex**. Если такой файл не существует, клиенту будет возвращено оглавление каталога. При включении директивы **FancyIndexing**, в оглавлении каталога будут использованы значки и описания файлов. Если директива **FancyIndexing** выключена, оглавление будет представлено в более простом виде.
- **AddIconByType** — сопоставляет значок типу файла. Значок будет использоваться при выводе каталога, если включена директива **FancyIndexing**. Директива **AddIconByType** имеет следующий формат:

  AddIconByType (TEXT, URL) mime-type

  Параметр TEXT определяет текстовое описание типа, которое увидят пользователи, использующие текстовый браузер или пользователи, у которых отключено отображение рисунков. Параметр URL определяет адрес значка, а параметр **mime-type** — это тип файла, с которым нужно сопоставить значок. Полный перечень MIME-типов приведен в файле `apache-mime.types`. В качестве имени файла можно задать не только MIME-тип, но и символы, которыми заканчивается имя файла (см. листинг 4.1), но для этого нужно использовать директиву **AddIcon** вместо **AddIconByType**.

Листинг 4.1. Фрагмент файла httpd.conf

```
AddIconByType (VID,/icons/movie.gif) video/*
AddIcon /icons/binary.gif .bin .exe
```

Первая директива в листинге сопоставляет типу **video** значок `/icons/movie.gif`. Вторая директива сопоставляет бинарным файлам ***.bin** и ***.exe** значок `/icons/binary.gif`. Значок по умолчанию задается директивой **DefaultIcon**.

Директивы обработки MIME-типов

Как вы помните, в Windows существует такое понятие, как расширение (или тип) файла. По расширению можно связать какую-либо программу с определенным типом файла. Например, когда вы щелкаете на файле с расширением .txt, запускается Блокнот, а при щелчке на файле с расширением .doc будет запущен Word. Точно такой же механизм реализован в Apache. Вы можете сопоставить тип файла и обработчик этого типа — программу, которая будет запущена для обработки файла. Когда не был

разработан модуль mod_php, программа PHP объявлялась обработчиком файла с расширением .php. Сервер запускал php, передавал ему файл, а потом возвращал пользователю результат.

- **DefaultType**. Если запрашиваемый клиентом тип не соответствует ни одному из MIME-типов, используется MIME-тип, указанный в директиве **DefaultType**.
- **AddEncoding**. Для сокращения времени передачи файла клиентам используется сжатие данных. Браузеры имеют встроенные программы для распаковки, запускаемые при получении архивов определенных MIME-типов. Именно эти MIME-типы и указываются в директиве **AddEncoding**.
- **AddType** — с помощью этой директивы можно добавить новый MIME-тип, который не указан в файле apache-mime.types.
- **AddHandler** и **Action**. Директива **AddHandler** позволяет сопоставить определенному MIME-типу какой-нибудь обработчик. А с помощью директивы **Action** можно определить какое-нибудь действие для обработчика. Например, вы можете запустить какую-нибудь программу для обработки файла данного типа. Использование этих директив, я думаю, лучше всего продемонстрировать на примере (см. листинг 4.2).

Листинг 4.2. Применение директив AddHandler и Action

```
AddHandler text/dhtml dhtml
Action text/dhtml /cgi-bin/dhtml-parse
```

Директивы для работы с многоязычными документами

Если на вашем сервере будут выложены многоязыковые документы, например, у вас будут две версии вашего сайта — Русская и Английская, целесообразно настроить директивы управления языком. Тогда пользователю не нужно будет щелкать мышкой на ссылке «In English» или «На Русском» — сервер, исходя из настроек браузера, сам определит, какой язык предпочитает пользователь, и отобразит нужную страницу.

- **AddLanguage**. В большинстве браузеров можно задать предпочитаемый язык. Благодаря этому вы можете предоставлять документы на разных языках. Директива **AddLanguage** сопоставляет расширение файла аббревиатуре языка. Для русского языка используется аббревиатура *ru*, для английского — *en*. При этом в корневом каталоге вашего сервера могут находиться несколько индексных файлов на разных языках. Например, для русского языка нужно использовать имя файла `index.html.ru`, а для английского — `index.html.en`.
- **LanguagePriority**. Если на вашем сервере размещены документы на разных языках, то с помощью директивы **LanguagePriority** можно

указать приоритеты различных языков. Например, вы установили директиву **LanguagePriority** так:

LanguagePriority en ru

Клиент вводит в строке адреса своего браузера адрес `http://www.server.com/`. Если в свойствах браузера имеется возможность задать предпочитаемый язык, то возвращен будет файл на нужном языке, если такой существует. Если браузер клиента не поддерживает такую возможность, будет возвращен файл на языке, имеющем наиболее высокий приоритет. В рассмотренном случае (см. пример) это английский язык. Если файл на нужном языке, например, на немецком, не существует, то будет возвращен файл на английском языке. Для того, чтобы сервер поддерживал нужный вам язык, предварительно установите правильное значение директивы **AddLanguage**.

Директивы перенаправления

Довольно часто нужно перенаправить пользователя на другой ресурс, например, вы сменили хостера и из-за этого изменилось имя вашего сайта. Обычно при таком «переезде» у вас есть определенное время, чтобы сообщить вашим пользователям об этом. Проще всего установить на старом сервере перенаправление на новый — во-первых, пользователи узнают ваш новый адрес, а во-вторых, им не нужно будет вводить этот адрес вручную — сервер сделает все автоматически.

Возможно, вы просто перенесли файлы в другой каталог — вам так удобнее, но пользователи не знают об этом и по-прежнему обращаются к старому каталогу. Тогда создайте редирект на новый каталог, и сервер автоматически перенаправит пользователей на него.

- **Redirect.** Используйте директиву **Redirect**, когда нужно перенести документы в другой каталог или на другой сервер. Например, вам нужно перенести данные из каталога `/users/den` в каталог /den. Если при этом старый URL-адрес был `http://www.host.com/users/den`, то новый станет `http://www.host.com/den/`. Используйте для этого следующую директиву:

  Redirect /users/den /den

  Можно также перенаправить запрос на другой сервер: **Redirect/ users/den/ www.den.domain.com**

  При этом допускается использование, как нового, так и старого URL-адреса.
- **Alias** — с помощью директивы **Alias** можно предоставить доступ не только к файлам, находящимся в каталоге, указанном директивой **DocumentRoot**, и его подкаталогах, но и в других каталогах. По умолчанию определен только псевдоним для каталога `/icons`.

- **ScriptAlias** — аналогична директиве **Alias**, но позволяет задать месторасположение каталога для CGI-сценариев.

Директивы обработки ошибок

Директива обработки ошибок всего одна, но она очень полезна. Например, произошла ошибка 404 (файл не найден). Вы можете сопоставить этой ошибке URL, на который будет перенаправлен браузер пользователя, если данная ошибка произошла. Обычно перенаправление устанавливают на документ, содержащий логотип сайта и сообщение об ошибке.

- **ErrorDocument** — директива, сопоставляющая коды ошибок сервера URL-адресам на этом же сервере.

4.2.4. Директивы управления доступом к отдельным каталогам

Теперь рассмотрим директивы управления доступом к отдельным каталогам. Вы можете определить отдельные параметры для каждого каталога вашего сервера — оформление каталога, параметры доступа к этому каталогу.

Блок директив Directory

Directory — обычно определяет свойства каталога (см. листинг 4.3).

Листинг 4.3. Директива Directory

```
<Directory />
     Options Indexes Includes FollowSymLinks
     AllowOverride None
</Directory>
```

Свойства каталога можно указывать в директиве **Directory** или в файле `.htaccess`, который находится в том каталоге, для которого необходимо установить нужные параметры.

В блоке **Directory** могут находиться директивы управления доступом. К ним относятся директивы **AllowOverride**, **Options**, **Limit**. Рассмотрим по порядку все эти директивы. Директива **AllowOverride** может принимать значения, указанные в табл. 4.1.

С помощью директивы **Options** можно определить функции сервера, которые будут доступны для использования в определяемом каталоге. Данную директиву можно использовать, как в файле `httpd.conf`, так и в файлах `.htaccess`. Допустимые опции для директивы **Options** представлены в табл. 4.2.

Значения директивы AllowOverride *Таблица 4.1*

Значение	Описание
None	Сервер Apache будет игнорировать файлы .htaccess. Рекомендую установить данную опцию, так как это повысит производительность сервера
All	Пользователи имеют право переопределять в файлах .htaccess глобальные параметры доступа. Из соображений безопасности лучше не использовать этот режим
Options	Разрешает использовать директиву **Options**
Limit	Разрешает использовать директиву **Limit**
AuthConfig	Разрешает использование директив **AuthName, AuthType, AuthUserFile** и **AuthGroupFile**
FileInfo	Разрешает использовать в файлах .htaccess директивы **AddType** и **AddEncoding**

Допустимые опции директивы Options *Таблица 4.2*

Опция	Описание
None	Не разрешается использование каких-либо функций
All	Разрешаются все функции
FollowSymLinks	Разрешается использовать символические ссылки. С точки зрения безопасности не рекомендуется использовать этот режим
SymLinksIfOwnerMatch	Разрешается использование символических ссылок, если ссылка указывает на объект, который принадлежит тому же пользователю, что и ссылка
ExecCGI	Разрешается выполнение CGI-сценариев
Indexes	Если эта опция выключена, сервер не будет передавать содержимое каталога при отсутствии файла index.html
Includes	Разрешено использование серверных включений. Рекомендую отключить эту опцию, поскольку это сильно нагружает сервер
IncludesNoExec	Разрешает использование серверных включений, но запрещает запуск из них внешних программ

Использование директивы Limit имеет следующий вид: Limit, *метод*

Директива **Limit** ограничивает доступ к файлам в определенном каталоге. Параметр **метод** определяет метод передачи, например, GET или POST. Директиву Limit можно использовать внутри блоков **Directory**, **Location** или в файле `.htacccess`.

Блок директив Limit

В блоке **Limit** можно использовать такие директивы: **allow** (разрешить), **deny** (запретить), **order** (порядок), **require** (требуется). После директивы **allow** следует слово **from**, после которого можно указать IP-адрес, адрес сети, домен или просто имя компьютера. Слово **all** обозначает все компьютеры. Например, вам требуется запретить доступ всем компьютерам, кроме компьютеров, которые входят в домен ru (см. листинг 4.4).

Листинг 4.4. Директивы allow, deny

```
order deny, allow
deny from all
allow from ru
```

Следующий пример показывает, как разрешить доступ компьютерам только из вашей сети (см. листинг 4.5). Пусть, при этом, ваша сеть имеет адрес 192.168.1.0.

Листинг 4.5. Разрешения доступа подсети 192.168.1.0

```
order deny, allow
deny from all
allow from 192.168.1.
```

Директива **order** определяет порядок выполнения директив **allow** и **deny**. Кроме значений `allow, deny` и `deny, allow`, директива **order** может содержать значение **mutual-failure**. В этом случае доступ будет отказан всем компьютерам, которые явно не указаны в списке **allow**.

Директиву **require** можно использовать для защиты каталога паролем. После названия директивы должен следовать список элементов: имена пользователей, групп, которые заданы в директивах **AuthUserFile** и **AuthGroupFile**. Можно использовать параметр **valid-user**, который укажет серверу предоставить доступ любому пользователю, имя которого имеется в директиве **AuthUserFile**, если он введет правильный пароль. Пример использования приведен в листинге 4.6.

Листинг 4.6. Использование директивы require

```
<Directory *>
     AuthUserFile /var/secure/.htpasswd
     AuthName Security
     AuthType Basic
     <Limit GET>
   order     deny,allow
   deny      form all
   allow     from mydomain.ru
   require   valid-user
     </Limit>
</Directory>
```

В листинге 4.6 для аутентификации используется файл паролей `.htpasswd`, который можно создать с помощью программы **htpasswd**. Директивы блока **Limit** разрешают доступ к любому каталогу сервера только пользователям домена `mydomain.ru`.

Кроме параметра **valid-user** допускается использование параметра **users** или **grpoups**. Данные параметры разрешают доступ только определенным пользователям или группам пользователей. Пример использования параметра **users** приведен в листинге 4.7.

Листинг 4.7. Применения параметра users

```
<Directory /users>
     AuthType Basic
     AuthUserFile /var/users/.htpasswd
     AuthName UsersDir
     <Limit GET POST>
          require users denis igor evg
     </Limit>
</Directory>
```

Блок директив Location

С помощью директив, расположенных в блоке **Location,** можно задать определенный URL-адрес, который предназначен для обозначения каталогов, файлов или групп файлов. Обозначить группу файлов можно с помощью шаблонов, например, шаблон ***.html** определяет все файлы, имена которых заканчиваются на .html. В URL-адрес не включается протокол и имя сервера. Пример описания блока **Location** представлен в листинге 4.8.

Листинг 4.8. Блок Location

```
<Location URL>
     директивы управления доступом
</Location>
```

4.2.5. Каталоги пользователей

Директива **UserDir** включает поддержку пользовательских каталогов. Эта директива определяет общее название подкаталога в домашних каталогах всех пользователей. По умолчанию используется каталог `public_html`. Данная возможность очень удобна при использовании ее в большой корпорации, где каждый сотрудник имеет собственную страничку. Раньше эта возможность часто использовалась на серверах, предоставляющих бесплатных хостинг. Может быть, помните адреса вида `http://www.chat.ru/~mypage?`

Сейчас же все чаще используется технология виртуальных серверов, которую мы рассмотрим в следующем пункте, но знать, что такое каталоги пользователей и как с ними работать тоже не помещает. Тем более, что домашние каталоги настраиваются намного быстрее и проще, чем виртуальный сервер — нужно всего лишь определить директиву **UserDir** и указать месторасположения домашних каталогов.

Доступ к файлам, расположенным в этих каталогах, производится с помощью указания через наклонную черту пользователя после имени сервера. Например, пусть имя сервера `www.server.com`, имя пользователя — `denis`, тогда URL-адрес будет выглядеть так: `http://www.server.com/~denis/`. При этом сервер самостоятельно определит, где именно расположен домашний каталог пользователя. Если домашний каталог пользователя `/home/den`, то сервер передаст клиенту файл `/home/den/public_html/index.html`.

4.2.6. Виртуальные HTTP-серверы

Концепция виртуальных хостов позволяет серверу Apache поддерживать несколько Web-узлов. Получается, что один Web-сервер заменяет несколько серверов, и вместо одного узла пользователи видят отдельные Web-узлы. Это очень удобно, если нужно организовать персональные Web-узлы пользователей или собственные Web-узлы подразделений компании, например, `develop.mycompany.com`.

Сервер Apache можно настроить несколькими способами:

- чтобы запускался один сервер, который будет прослушивать ВСЕ обращения к виртуальным серверам,
- или запускать отдельный процесс для каждого виртуального сервера.

В первом случае один сервер будет одновременно обслуживать все виртуальные. Если вас интересует такой вариант, нужно настраивать виртуальные сервера с помощью директивы **VirtualHost**. Настройка отдельных процессов для каждого сервера осуществляется с помощью директивы **Listen** и **BindAddress**.

В этом разделе я буду рассматривать именно первый случай, то есть с использованием директивы **VirtualHost** (точнее блока директив). Внутри блока директив **VirtualHost** можно использовать любые директивы, кроме **ServerType, BindAddress, Listen, NameVirtualHost, ServerRoot, TypesConfig, PidFile, MinRequestPerChild, MaxSpareServers, MinSpareServers**, так как некоторые из них относятся к основному HTTP-серверу (например, ServerType), а некоторые — ко второму варианту настройки виртуальных серверов и здесь неприемлемы. Обязательно должны присутствовать директивы **ServerName, DocumentRoot, ServerAdmin** и **ErrorLog**.

В зависимости от версии и от настроек Apache виртуальные узлы могут прописываться или в файле `httpd.conf`, или в файле `vhosts.conf`.

Виртуальные серверы можно идентифицировать по имени или по IP-адресу.

Виртуальные серверы с идентификацией по имени

Идентификация по имени имеет существенное преимущество перед идентификацией по IP-адресу: вы не ограничены количеством адресов, имеющемся у вас в распоряжении. Вы можете использовать любое количество виртуальных серверов, и при этом вам не потребуются дополнительные адреса. Такое возможно благодаря использованию протокола HTTP/1.1. Данный протокол поддерживается всеми современными браузерами.

Поддержка виртуальных хостов обеспечивается директивами **VirtualHost** и **NameVirtualHost**. Если ваша система имеет только один IP-адрес, его нужно указать в директиве **VirtualHost**. Внутри блока директивы **VirtualHost** записывается директива **ServerName**. Эта директива задает доменное имя для создаваемого виртуального сервера. Это обязательно нужно сделать, чтобы избежать поиска службой DNS — вы же не хотите, чтобы при неудачном поиске виртуальный сервер был заблокирован? Все директивы **VirtualHost** используют один и тот же IP-адрес, заданный директивой **NameVirtualHost**. В блоке **VirtualHost** записываются параметры виртуального сервера, причем они записываются для каждого виртуального сервера отдельно. Пример приведен в листинге 4.9.

Структура каталогов виртуального узла задается директивами **DocumentRoot**, **ErrorLog** и **TransferLog**. Из листинга 4.9 видно, что корневым каталогом для нашего виртуального узла будет `/var/httpd/www/html`, а протоколы будут находиться в каталоге `/var/httpd/www/logs/error.log`.

Листинг 4.9. Два виртуальных сервера — www и lib

```
ServerName den.dhsilabs.com

<NameVirtulaHost 192.168.1.1>

<VirtualHost 192.168.1.1>
   ServerName www.dhsilabs.com
   ServerAdmin webmaster@den.dhsilabs.com
   DocumentRoot /var/httpd/www/html
   ErrorLog /var/httpd/www/logs/error.log
   TransferLog logs/access.log
</VirtualHost>

<VirtualHost 192.168.1.1>
```

```
   ServerName lib.dhsilabs.com
   ServerAdmin webmaster@den.dhsilabs.com
   DocumentRoot /var/httpd/lib/html
   ErrorLog /var/httpd/lib/logs/error.log
   TransferLog logs/access.log
</VirtualHost>
```

Если ваша система имеет только один IP-адрес, значит доступ к основному серверу станет невозможным, то есть вы не сможете использовать его напрямую. Можно основной сервер использовать в качестве виртуального, что я и сделал в листинге 4.9: основной сервер www является виртуальным. При наличии двух IP-адресов можно один присвоить основному серверу, а другой — виртуальному.

Сервер Apache позволяет использовать несколько доменных имен для доступа к одному серверу, например:

```
ServerAlias www.dhsilabs.com www2.dhsilabs.com
```

При этом запросы, посланные по IP-адресам, которые присвоены вашим виртуальным хостам, должны соответствовать одному из указанных доменных имен. Чтобы зафиксировать запросы, не соответствующие ни одному их этих имен, нужно с помощью опции **default:*** создать виртуальный хост, который будет обслуживать такие запросы:

```
<VitrualHost _default_:*>
```

Обратите внимание на то, что в рассмотренном примере адреса `www.dhsilabs.com` и `lib.dhsilabs.com` должны быть прописаны в DNS.

Виртуальные серверы с идентификацией по IP-адресу

В директиве **VirtualHost** в качестве адресов можно использовать доменные имена, но лучше указывать IP-адрес, причем действительный, а не виртуальный. В этом случае вы не будете зависеть от DNS при разрешении имени. Также потребуется один IP-адрес для вашего основного сервера. Если же распределить все адреса между виртуальными серверами, то нельзя будет получить доступ к основному серверу.

Листинг 4.10. Идентификация по IP-адресу

```
<VitrualHost 192.168.1.2>
   ServerName www.dhsilabs.com
   ServerAdmin webmaster@den.dhsilabs.com
   DocumentRoot /var/httpd/www/html
   ErrorLog /var/https/www/logs/error.log
</VirtualHost>
```

```
<VirtualHost lib.dhsilabs.com>
  ServerName lib.dhsilabs.com
  ServerAdmin webmaster@den.dhsilabs.com
  DocumentRoot /var/httpd/lib/html
  ErrorLog /var/https/lib/logs/error.log
</VirtualHost>
```

В приведенном примере (см. листинг 4.10) сконфигурированы два виртуальных сервера с идентификацией по IP-адресу. Один из них использует сам IP-адрес, а другой — доменное имя, соответствующее IP-адресу.

При конфигурировании виртуальных серверов можно использовать опцию **ExecCGI**, которая разрешает выполнение CGI-скриптов на виртуальном сервере. Ниже приведен пример для почтового Web-интерфейса (см. листинг 4.11)

Листинг 4.11. Подключение почтового Web-интерфейса

```
# Файл httpd.conf

<Directory /home/httpd/mail>
  order deny,allow
  deny from all
  allow from localhost
  allow from 192.168
  allow from 123.123.123.123
  Options ExecCGI
</Directory>

# Файл vhosts.conf

<VirtualHost 123.123.123.123>
  ServerAdmin webmaster@den.dhsilabs.com
  DocumentRoot /home/httpd/mail
  ServerPath /mail
  ServerName wwwmail.dhsilabs.com
  ErrorLog logs/error_log
  TransferLog logs/access_log
  ErrorDocument 403 http://www.dhsilabs.com/messages/error403.html
</VirtualHost>

# Error 403 — доступ извне, то есть почтовый интерфейс будет
# доступен только из локальной сети
```

4.2.7. Пример конфигурационного файла httpd.conf

В этом разделе приведен пример стандартной конфигурации сервера Apache (см. листинг 4.12). Каждому блоку листинга сопутствуют комментарии на русском языке, которые помогут вам разобраться с различными опциями сервера.

В зависимости от версии Apache и PHP возможны незначительные отличия, но в целом картина в листинге отражена верно.

Листинг 4.12. Пример файла httpd.conf

```
##
## httpd.conf - файл конфигурации сервера HTTP Apache
##
# ------------------------------------------------------

# Установите имя сервера
ServerName www.dhsilabs.com
ResourceConfig /dev/null
AccessConfig /dev/null

# Поддержка динамических разделяемых объектов (Dynamic Shared
# Object - DSO)
# Для более подробной информации о DSO прочтите файл
# README.DSO, входящий в дистрибутив Apache.
# Модуль расширяет возможности сервера Apache, добавляет в его
# состав новые функции.
# Подключить модуль можно так
# LoadModule foo_module libexec/mod_foo.so
# Вы можете найти документацию по модулям в файле
# "/var/www/manual/mod"

#LoadModule mmap_static_module modules/mod_mmap_static.so
LoadModule env_module          modules/mod_env.so

### The first module activates buffered logs.
# Первый модуль обеспечивает протоколирование. Он запишет
# информацию в протокол, когда буфер объемом 4К переполнится.
# Используется файл журнала access_log
# LoadModule config_buffered_log_module  modules/
# mod_log_config_buffered.so
LoadModule config_log_module  modules/mod_log_config.so

LoadModule agent_log_module    modules/mod_log_agent.so
LoadModule referer_log_module modules/mod_log_referer.so
```

```
# LoadModule mime_magic_module   modules/mod_mime_magic.so
LoadModule mime_module           modules/mod_mime.so
LoadModule negotiation_module modules/mod_negotiation.so
LoadModule status_module         modules/mod_status.so
LoadModule info_module           modules/mod_info.so

# Вы должны выбрать директиву mod_include или mod_include_xssi,
# но не обе одновременно! Директива mod_include более безопасна,
# но xxsi содержит больше функций.
LoadModule includes_module      modules/mod_include.so
# LoadModule includes_module      modules/mod_include_xssi.so

LoadModule autoindex_module    modules/mod_autoindex.so
LoadModule dir_module           modules/mod_dir.so
LoadModule cgi_module           modules/mod_cgi.so
LoadModule asis_module          modules/mod_asis.so
LoadModule imap_module          modules/mod_imap.so
LoadModule action_module        modules/mod_actions.so
#LoadModule speling_module       modules/mod_speling.so
LoadModule userdir_module       modules/mod_userdir.so
LoadModule proxy_module         modules/libproxy.so
LoadModule alias_module         modules/mod_alias.so

# Модуль mod_jserv должен быть объявлен до mod_rewrite.
<IfDefine HAVE_JSERV>
LoadModule jserv_module         modules/mod_jserv.so
</IfDefine>

LoadModule rewrite_module       modules/mod_rewrite.so
LoadModule access_module        modules/mod_access.so
LoadModule auth_module          modules/mod_auth.so
LoadModule anon_auth_module    modules/mod_auth_anon.so
#LoadModule dbm_auth_module      modules/mod_auth_dbm.so
#LoadModule db_auth_module       modules/mod_auth_db.so
LoadModule digest_module        modules/mod_digest.so
#LoadModule cern_meta_module     modules/mod_cern_meta.so
LoadModule expires_module       modules/mod_expires.so
LoadModule headers_module       modules/mod_headers.so
LoadModule usertrack_module    modules/mod_usertrack.so
#LoadModule example_module       modules/mod_example.so
#LoadModule unique_id_module     modules/mod_unique_id.so
LoadModule setenvif_module      modules/mod_setenvif.so
```

```
# В полном списке модулей должны быть перечислены
# все доступные модули
#(статических или общедоступных), чтобы достичь правильного
#порядка выполнения.
ClearModuleList
#AddModule mod_mmap_static.c
# AddModule mod_php.c
# AddModule mod_php3.c
#AddModule mod_php4.c
#AddModule mod_perl.c
# LoadModule php_module modules/mod_php.so
# LoadModule php3_module modules/mod_php4.so
#LoadModule php4_module modules/mod_php4.so
AddModule mod_env.c
AddModule mod_log_config.c
#AddModule mod_log_config_buffered.c
AddModule mod_log_agent.c
AddModule mod_log_referer.c
#AddModule mod_mime_magic.c
AddModule mod_mime.c
AddModule mod_negotiation.c
AddModule mod_status.c
AddModule mod_info.c
AddModule mod_include.c
#AddModule mod_include_xssi.c
AddModule mod_autoindex.c
AddModule mod_dir.c
AddModule mod_cgi.c
AddModule mod_asis.c
AddModule mod_imap.c
AddModule mod_actions.c
#AddModule mod_speling.c
AddModule mod_userdir.c
AddModule mod_proxy.c
AddModule mod_alias.c

# Модуль mod_jserv должен быть объявлен до mod_rewrite.
<IfDefine HAVE_JSERV>
  AddModule mod_jserv.c
</IfDefine>

AddModule mod_rewrite.c
AddModule mod_access.c
AddModule mod_auth.c
```

```
AddModule mod_auth_anon.c
#AddModule mod_auth_dbm.c
#AddModule mod_auth_db.c
AddModule mod_digest.c
#AddModule mod_cern_meta.c
AddModule mod_expires.c
AddModule mod_headers.c
AddModule mod_usertrack.c
#AddModule mod_example.c
#AddModule mod_unique_id.c
AddModule mod_so.c
AddModule mod_setenvif.c


#--------- Name Space and Server Settings ----------
# Настройки пространства имен и сервера
# В этом разделе вы определяете, какие имена будут видеть
# пользователи вашего HTTP-сервера. Этот файл также определяет
# настройки сервера, которые раньше содержались в отдельном
# файле srm.conf. Теперь этот
# файл входит в состав httpd.conf

# Директива DocumentRoot определяет местонахождение корневого
# каталога документов вашего сервера.

DocumentRoot /var/www/html

# Директива UserDir задает названия подкаталога в домашнем
# каталога пользователя, из  которого берутся документы в том
# случае, вы активизируете возможность  использования
# пользовательских каталогов.
UserDir public_html

# Директива DirectoryIndex позволяет задать название документа,
# который будет возвращен по запросу, который не содержит имя
# документа.
DirectoryIndex index.html index.php index.htm index.shtml
index.cgi Default.htm default.htm index.php3

# Директива FancyIndexing определяет оформление каталога -
# стандартное или индексируемое.

FancyIndexing on
```

```
# Директивы AddIcon* указывают серверу, какие пиктограммы
# использовать для показа различных типов файлов.
AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip

AddIconByType (TXT,/icons/text.gif) text/*
AddIconByType (IMG,/icons/image2.gif) image/*
AddIconByType (SND,/icons/sound2.gif) audio/*
AddIconByType (VID,/icons/movie.gif) video/*

AddIcon /icons/binary.gif .bin .exe
AddIcon /icons/binhex.gif .hqx
AddIcon /icons/tar.gif .tar
AddIcon /icons/world2.gif .wrl .wrl.gz .vrml .vrm .iv
AddIcon /icons/compressed.gif .Z .z .tgz .gz .zip
AddIcon /icons/a.gif .ps .ai .eps
AddIcon /icons/layout.gif .html .shtml .htm .pdf
AddIcon /icons/text.gif .txt
AddIcon /icons/c.gif .c
AddIcon /icons/p.gif .pl .py
AddIcon /icons/f.gif .for
AddIcon /icons/dvi.gif .dvi
AddIcon /icons/uuencoded.gif .uu
AddIcon /icons/script.gif .conf .sh .shar .csh .ksh .tcl
AddIcon /icons/tex.gif .tex
AddIcon /icons/bomb.gif core

AddIcon /icons/back.gif ..
AddIcon /icons/hand.right.gif README
AddIcon /icons/folder.gif ^^DIRECTORY^^
AddIcon /icons/blank.gif ^^BLANKICON^^

# Директива DefaultIcon определяет пиктограмму по умолчанию.

DefaultIcon /icons/unknown.gif

# Директива AddDescription задает описание файла
# Формат: AddDescription "описание" filename
# Директива ReadmeName определяет имя файла README по умолчанию
# Формат: ReadmeName name
ReadmeName README
HeaderName HEADER
```

```
# Директива IndexIgnore определяет набор файлов, которые
# будут проигнорированы при индексировании
# Формат: IndexIgnore name1 name2...
IndexIgnore .??* *~ *# HEADER* README* RCS

# Директива AccessFileName определяет имя файла, содержащего
# директивы управления  доступом.

AccessFileName .htaccess

# Директива TypesConfig задает местонахождение файла mime.types

TypesConfig /etc/httpd/conf/apache-mime.types

# С помощью директивы DefaultType можно указать MIME-тип по
# умолчанию, для  документов, тип которых сервер
# определить не может.

DefaultType text/plain

# Директива AddEncoding разрешает вашему браузеру распаковывать
# информацию «на лету».

AddEncoding x-compress Z
AddEncoding x-gzip gz

# AddLanguage разрешает определять язык документа

AddLanguage en .en
AddLanguage fr .fr
AddLanguage de .de
AddLanguage da .da
AddLanguage el .el
AddLanguage it .it

# Директива LanguagePriority определяет приоритет языков

LanguagePriority en fr de

# Директива Redirect позволяет перенаправить клиента на другой URL
# Вы можете перенаправить клиента на другой url или же url,
# который находится в вашем пространстве имен, то есть на
# любой документ, который находится в одном из подкаталогов
```

```
# каталога DocumentRoot. Вы не можете, например, перенаправить
# клиента к каталогу /etc, потому что он не
# находится в вашем пространстве имен.
# URL - это идентификатор ресурса, поэтому вы должны его
# указывать в виде протокол://адрес.домен, например,
# http://www.linux.ru. Если вы укажете просто каталог, например,
# /images, этот каталог должен быть подкаталогом каталога
# DocumentRoot, а не  корневого каталога вашей основной
# файловой системы. Формат: Redirect несуществующий_url url

# С помощью директивы Alias можно предоставить доступ не
# только к файлам, находящимся в каталоге, указанном директивой
# DocumentRoot, и его подкаталогах, но и в других каталогах.
# Формат: Alias несуществующее_имя нормальное_имя

Alias /icons/ /var/www/icons/

# ScriptAlias определяет расположение каталога сценариев CGI
# Формат: ScriptAlias подставное_имя настоящее_имя

ScriptAlias /cgi-bin/ /var/www/cgi-bin/
ScriptAlias /protected-cgi-bin/ /var/www/protected-cgi-bin/


# С помощью директивы AddType можно добавить новый тип MIME, #
# который не указан в файле apache-mime.types.
# Формат: AddType type/subtype ext1

# Обычно для модуля PHP3 (он не является частью Apache)
# директива AddType используется так:
AddType application/x-httpd-php4 .php3 .phtml .php .php4
# AddType application/x-httpd-php3-source .phps
# Для PHP/FI (PHP2):
# AddType application/x-httpd-php .phtml
# ScriptAlias /_php/ /usr/bin/php
# Action application/x-httpd-php /usr/bin/php
# Action application/x-httpd-php3 /usr/bin/php
# Action application/x-httpd-php4.

# Директива AddHandler позволяет сопоставить определенному типу
# MIME какой-нибудь обработчик.
# Формат: AddHandler action-name ext1
```

```
# Для использования сценариев CGI:
AddHandler cgi-script .cgi

# Для использования генерируемых сервером файлов HTML
AddType text/html .shtml
AddHandler server-parsed .shtml

# Раскомментируйте нижерасположенную строку, чтобы включить
# функцию Apache «отправь-как-есть» (send-as-is)
#AddHandler send-as-is asis

# Если вы хотите использовать карты изображений:
AddHandler imap-file map

# Для включения карт типов, используйте:
#AddHandler type-map var

# С помощью директивы Action можно определить какое-нибудь
# действие для обработчика.
# Например, вы можете запустить какую-нибудь программу
# для обработки файла данного типа.
# Формат: Action media/type /cgi-script/location
# Формат: Action handler-name /cgi-script/location

# Директива MetaDir определяет имя каталога, в котором сервер
# Apache может найти информационные файлы meta. Эти файлы
# содержат дополнительные заголовки HTTP, которые будут
# добавлены к документу перед его передачей клиенту.

#MetaDir .web

# Директива MetaSuffix определяет имя суффикса файла, который
# содержит meta-тэги.

#MetaSuffix .meta

# Здесь можно определить сообщения об ошибках. Это можно
# сделать тремя методами:
#     1) обыкновенный текст
# ErrorDocument 500 "The server made a boo boo.
#  n.b.  the (") marks it as text, it does not get output
#
#     2) локальное перенаправление
# ErrorDocument 404 /missing.html
```

```
#  to redirect to local url /missing.html
# ErrorDocument 404 /cgi-bin/missing_handler.pl
#  n.b. can redirect to a script or a document using server-
# side-includes.
#
#     3) внешнее перенаправление
# ErrorDocument 402 http://some.other_server.com/
# subscription_info.html
#
<Location /manual>
ErrorDocument 404 "The document you requested has not been installed
on your system. Please install the apache-manual package."
</Location>

# Модуль mod_mime_magic позволяет серверу использовать
# различные подсказки из файла для определения его типа.
# MimeMagicFile /etc/httpd/conf/magic

# Следующие директивы необходимы для браузеров Netscape 2.x и
# Internet Explorer  4.0b2

BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-
response-1.0

# Следующие директивы отключают ответы HTTP/1.1 для браузеров,
# которые не поддерживают протокол HTTP/1.1

BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0


#------------- Настройки доступа -------------

# В этом разделе определяются настройки сервера, которые
# управляют доступом к серверу. Раньше эти настройки находились
# в файле access.conf.

# Каждый каталог, к которому Apache может получить доступ,
# может быть сконфигурирован определенным образом. Можно
# запретить выполнение некоторых операций, доступ определенных
# пользователей или узлов сети.
# Установки доступа распространяются на весь каталог и на
```

```
# все его подкаталоги.
# Прежде всего, конфигурируем корневой каталог для
# установки полномочий доступа.

<Directory />
Options Indexes Includes FollowSymLinks
AllowOverride None
</Directory>


<Directory /home>

# Здесь должны быть определены директивы "Includes",
# "FollowSymLinks", "ExecCGI",  "MultiViews"
# или любая комбинация "Indexes".

Options Indexes Includes FollowSymLinks

AllowOverride All

# Разрешает доступ всем
order allow,deny
allow from all

</Directory>

# Каталоги /var/www/cgi-bin и /var/www/protected-cgi-bin должны
# быть определены с помощью директивы ScriptAliased.

<Directory /var/www/cgi-bin>
AllowOverride All
Options ExecCGI
</Directory>

<Directory /var/www/protected-cgi-bin>
order deny,allow
deny from all
allow from localhost
#allow from .your_domain.com
AllowOverride None
Options ExecCGI
</Directory>
# Разрешает отчеты о состояния сервера
```

```
<Location /server-status>
SetHandler server-status

order deny,allow
deny from all
allow from localhost, 127.0.0.1
# Установите здесь имя ваше домена
# allow from .your_domain.com
</Location>

# Разрешает доступ к файлам документации для локальной машины.
Alias /doc /usr/share/doc
<Directory /usr/share/doc>
order deny,allow
deny from all
allow from localhost, 127.0.0.1
# allow from .your_domain.com
Options Indexes FollowSymLinks
</Directory>

#-------------- Конфигурация сервера ----------------

# Тип сервера: inetd или standalone.

ServerType standalone

# Если вы используете тип inetd, перейдите к директиве
# "ServerAdmin"

# Директива Port - только для standalone-сервера. Если вы
# хотите запустить сервер Apache с использованием этого или
# любого другого порта, номер которого меньше 1024, вы
# должны обладать правами суперпользователя. Но даже если
# у вас нет таких прав, вы можете запустить сервер для
# работы с портом, номер которого превышает значение 1024.
# Обычно используется номер 8080 или 8000.

Port 80

# Сервер Apache ведет журнал доступа других компьютеров. Если
# вы включите данную опцию, то в журнал будет записано
# доменное имя компьютера-клиента. Если эта опция выключена,
# то в журнал будет записан IP-адрес клиента. Включение данной
# опции замедляет работу сервера, так как ему требуется
```

```
# дополнительное время на ожидание ответа от сервера DNS.
HostnameLookups off

# Директивы User и Group определяют идентификаторы пользователя
# и группы. Данные идентификаторы присваиваются серверу, если
# он запущен в автономном режиме. Можно использовать как имена
# пользователей, так и их числовые эквиваленты - UID. По
# умолчанию используется имя пользователя nobody или apache.
# Из соображений безопасности не рекомендуется изменять
# это значение и присваивать имя реального пользователя.

User apache
Group apache

# Директива ServerAdmin задает электронный адрес Web-мастера
# вашего Web-узла. В случае возникновения ошибок именно по
# этому адресу будет отправлено сообщение.

ServerAdmin root@localhost

# В директиве ServerRoot указывается местонахождение файлов
# конфигурации сервера Apache. По умолчанию используется
# каталог /etc/httpd.

ServerRoot /etc/httpd

# Данная директива используется для компьютеров, которые имеют
# несколько IP-адресов. Обычно данная директива используется
# для конфигурирования виртуальных хостов. BindAddress *

# Прослушивать порт 80
Listen 80

# Директивы ErrorLog и TransferLog определяют расположение
# журналов сервера Apache. Обычно используется каталог
# /etc/httpd/logs, который является ссылкой на каталог
# /var/log/httpd или на любой другой.

ErrorLog logs/error_log

# LogLevel: устанавливает уровень протоколирования.
```

```
# Протоколируются предупреждающие сообщения сервера (warn)
# и ошибки. Если вы хотите протоколировать только ошибки,
# установите error
LogLevel warn

# Определяет формат файлов протокола, то есть информация,
# которая будет протоколироваться. Обычно их не нужно изменять.

LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-
Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent

# Можно определить тип протокола
# Если вы хотите протоколировать общую информацию
# CustomLog logs/access_log common
# Если вы хотите протоколировать referer
# CustomLog logs/referer_log referer
# Если вы хотите протоколировать название пользовательских
# агентов (браузеров)CustomLog logs/agent_log agent
# По умолчанию используется комбинированный тип
# протоколирования, то есть
# будет протоколироваться вся информация.

CustomLog logs/access_log combined

# С помощью директивы PidFile указывается имя файла, в котором
# исходный процесс сервера будет регистрироваться.

PidFile /var/run/httpd.pid

# ScoreBoardFile: Этот файл используется для сохранения
# внутренней информации процесса сервера.

ScoreBoardFile /etc/httpd/httpd.scoreboard

# Директива LockFile определяет файл блокировки, который
# используется сервером. Сервер должен быть скомпилирован
# с опцией USE_FCNTL_SERIALIZED_ACCEPT или
# USE_FLOCK_SERIALIZED_ACCEPT. Файл блокировки должен быть
```

```
# сохранен НА ЛОКАЛЬНОМ ДИСКЕ.
#
LockFile /etc/ittpd/httpd.lock

# Директива ServerName позволяет вам установить имя узла вашего
# сервера, которое будет использоваться клиентами, то есть
# установить «www» вместо реального имени  сервера. Имя,
# которое вы укажете, должно быть указано в сервере DNS вашего
# домена.

ServerName new.host.name

# Директива UseCanonicalName появилась в версии Apache 1.3.
# Она разрешает использовать каноническое имя для сервера узла.

UseCanonicalName on

# Данная директива позволяет прокси-серверу, например SQUID,
# не кэшировать документы, которые не генерируются автоматически,
# то есть в процессе выполнения различных сценариев.

CacheNegotiatedDocs

# Директива Timeout задает промежуток времени в секундах, в
# течение которого сервер продолжает попытки возобновления
# приостановленной передачи данных. Значение директивы Timeout
# распространяется не только на передачу, но и на прием данных.

Timeout 300

# Директива KeepAlive разрешает постоянные соединения, то есть
# такие соединения, в которых производится более одного
# запроса за один раз.

KeepAlive off

# MaxKeepAliveRequests: Максимальное количество запросов,
# разрешенное в течение постоянного соединения. Установите 0
# для снятия ограничения. Для повышения производительности
# рекомендуется установить это число сравнительно большим.

MaxKeepAliveRequests 100

# Директива KeepAliveTimeout  определяет тайм-аут для
```

```
# постоянного соединения.

KeepAliveTimeout 15

# Минимальное и максимальное число серверов в пуле
MinSpareServers 8
MaxSpareServers 20

# Количество серверов для запуска

StartServers 10

# Ограничивает общее количество клиентов. Когда это число будет
# превышено, новые клиенты получат отказ, поэтому это число
# не должно быть слишком маленьким.

MaxClients 150

# После обработки определенного количества запросов, указанного
# в директиве MaxRequestsPerChild, копия сервера завершается,
# а вместо нее запускается новая.

MaxRequestsPerChild 500

# Директивы конфигурации прокси-сервера

# ProxyRequests On

# Для включения кэширования, раскомментируйте строки ниже:

# CacheRoot /var/cache/httpd
# CacheSize 5
# CacheGcInterval 4
# CacheMaxExpire 24
# CacheLastModifiedFactor 0.1
# CacheDefaultExpire 1
# NoCache a_domain.com another_domain.edu joes.garage_sale.com

####################################
# Настройки производительности SGI #
####################################
#
#
```

```
# Для использования этой функции, раскомментируйте модуль
# mod_mmap_static в разделе описания модулей.
<IfModule mod_mmap_static.c>
QSC on
</IfModule>
# Если вы хотите использовать буферизированное протоколирование,
# раскомментируйте модуль mod_log_config_buffered в разделе
# описания модулей.

# Для использования карты памяти, раскомментируйте эту строку:
# mmapfile /var/www/html/file_to_map_in_memory
#
# Если вы хотите настроить процессы потомки, пожалуйста,
# прочитайте документацию на вашем сервере
# http://localhost/manual/misc/perf-mja.html.
# Эта страница объясняет как привязать определенный IP-адрес
# или порт к другому процессору. SingleListen On

####################################
# Виртуальные серверы              #
####################################
#
# Поддержка модуля perl
# Замечание: не удаляйте расположенные далее строки, иначе это
# может разрушить вашу конфигурацию.
<IfDefine PERLPROXIED>
 ProxyPass    /perl/ http://127.0.0.1:8200/
 ProxyPassReverse /perl/ http://127.0.0.1:8200/
</IfDefine>

# Файл, в котором находятся директивы конфигурирования
# виртуальных узлов.
Include conf/vhosts/Vhosts.conf

#
# Для поддержки динамических виртуальных хостов и виртуальных
# домашних каталогов, раскомментируйте следующие строки:
# LoadModule vhost_alias_module   modules/mod_vhost_alias.so
# AddModule mod_vhost_alias.c
# Include conf/vhosts/DynamicVhosts.conf
# Include conf/vhosts/VirtualHomePages.conf

# Директивы конфигурирования PHP
Include conf/addon-modules/php.conf
```

4.3. Установка Apache + PHP + MySQL под Linux (Unix)

Все необходимое программное обеспечение входит в состав Linux — серверы Apache и MySQL, PHP, а также дополнительные модули **mod_php** и **php_mysql**. Однако, если вы хотите установить последние версии, рекомендую посетить следующие сайты:

- `www.apache.org` — официальный сайт Apache Group;
- `www.mysql.org` — сайт разработчиков MySQL;
- `www.php.net` — официальный сайт разработчиков PHP.

4.3.1. Установка Apache

Установка пакетов

Для установки сервера Apache необходимо установить пакеты **apache** и **apache-docs.** В первом из них находится сам сервер, а во втором — документация. Желательно устанавливать самую новую версию. В последних версиях вам нужно установить еще и пакет **apache-common**, содержащий необходимые файлы для запуска сервера. Проще всего установку Apache можно произвести, введя следующую команду:

```
rpm -ih apache*
```

Примечание.
В Red Hat 9.0 пакет, содержащий сервер Apache, называется `httpd-2.0`.

После установки сервер конфигурируется для запуска в режиме **standalone**, то есть он будет постоянно находиться в памяти. Я не рекомендую изменять этот режим. Для запуска и останова сервера Apache вы можете воспользоваться командами:

```
/etc/rc.d/init.d/httpd start;
/etc/rc.d/init.d/httpd stop.
```

или командами

```
service httpd start;
service httpd stop.
```

После успешной установки сервера отредактируйте файл `/etc/httpd/conf/httpd.conf`. В нем исправьте всего одну директиву — **ServerName**. При этом, на данном этапе (пока новое имя сервера Apache не зарегистрировано на вашем сервере DNS), вместо того имени, которое будет использоваться в дальнейшем (например, `www.host.domain`), установите обык-

новенное имя вашего сервера, например, `server.firma.ru`. Данное имя должно быть зарегистрировано на DNS-сервере вашей сети. После этого запустите сервер. Затем откройте любой браузер и попробуйте обратиться к серверу (см. рис. 4.1):

```
netscape http://localhost
```

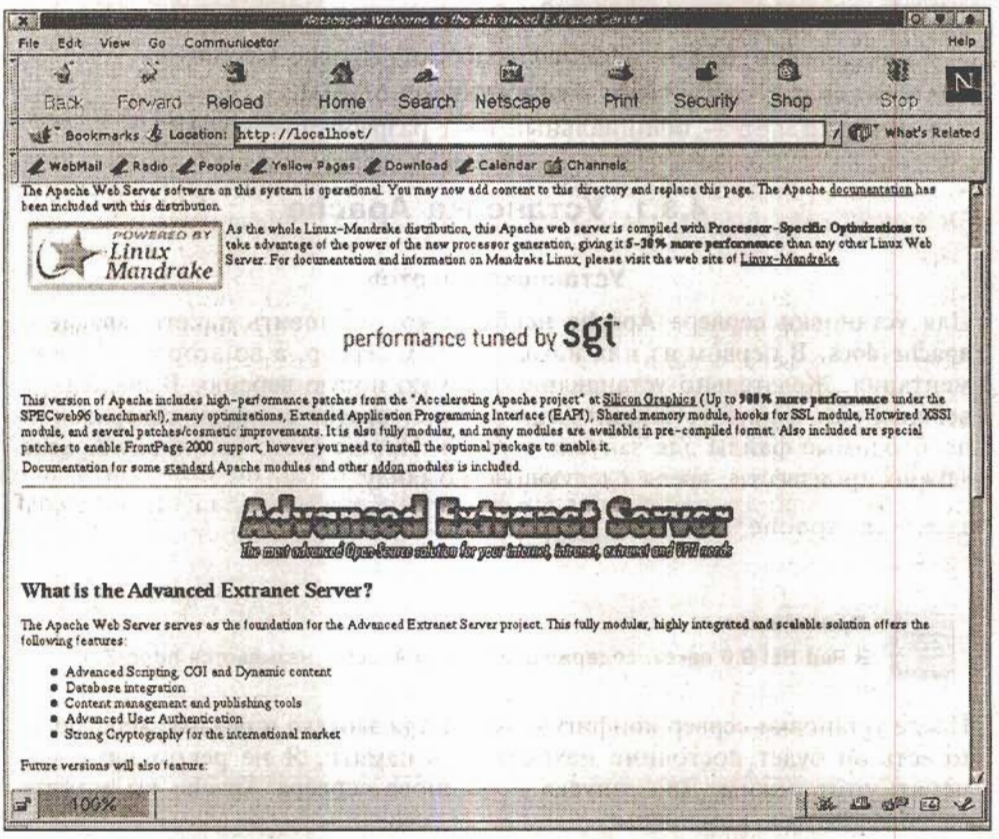


Рис. 4.1. *Первое обращение к серверу Apache*

Теперь попробуйте обратиться к этому серверу с другого компьютера вашей сети:

```
netscape http://server.firma.ru
```

Если вы увидите приветствие сервера в первом и во втором случаях, значит ваш сервер Apache нормально работает и можно приступать к его дальнейшему конфигурированию. Если в первом случае у вас произошла ошибка, значит искать ее нужно на локальном уровне. При этом, если сеть нормально работает, то, скорее всего, вы просто забыли запустить сервер. Появление ошибки во втором случае может быть связано с не-

правильной установкой директивы **ServerName**, или же используемое вами имя не прописано в системе DNS.

В отличие от Windows-версии Apache, в Linux-версии вам не нужно редактировать какие-либо другие директивы, поскольку стандартные значения устраивают большинство пользователей. Вам также не нужно вручную прописывать PHP, поскольку он встраивается в Apache автоматически при установке пакета **mod_php**. Единственное, что вам нужно знать — это структуру каталогов — что и где лежит.

Корневым каталогом вашего Web-сервера является каталог `/var/www/html`. Когда пользователь введет `http://your_server.ru/` в строке браузера, ваш Web-сервер будет искать файл `index.html` именно в этом каталоге. Если данный каталог вас не устраивает, измените директиву **DocumentRoot** в файле конфигурации `httpd.conf`, после чего перезапустите Apache: команда **service httpd restart**.

Вернемся к нашей структуре каталогов. Каталог `/var/www/cgi-bin` содержит CGI-приложения: из этого каталога возможен запуск программ. Журналы сервера находятся в каталоге `/var/log/httpd`.

Конфигурирование Apache с помощью netconf

Практически все параметры Web-сервера Apache можно установить, используя конфигуратор **netconf** (или **linuxconf**). Для этого запустите **netconf** и перейдите на вкладку **Server Tasks**, а затем нажмите на кнопку «Apache Web-server» (см. рис. 4.2).

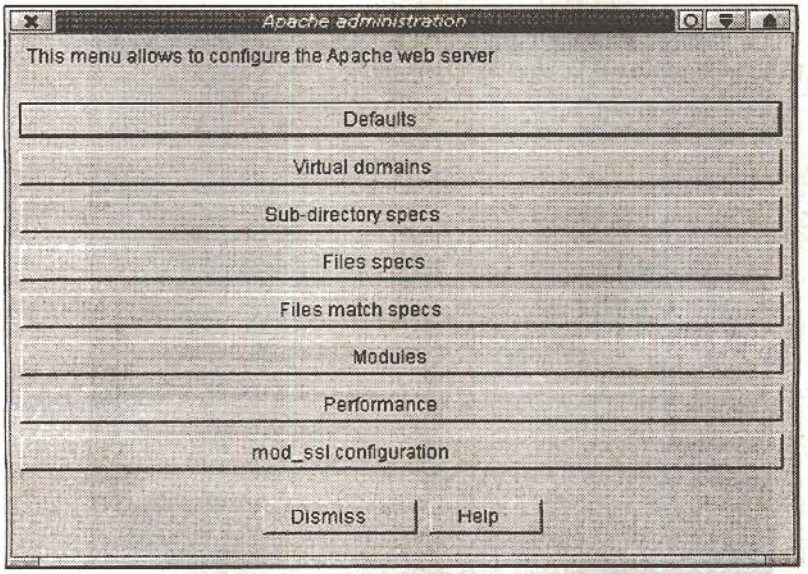


Рис. 4.2. Конфигурирование Apache с помощью netconf

С помощью **netconf** вы легко можете установить основные параметры Apache (см. рис. 4.3), определить виртуальные хосты, установить параметры подкаталогов, определить спецификацию каталогов и модулей, а также установить параметры модуля **mod_ssl** (см. рис. 4.4).

Примечание.

Модуль mod_ssl обеспечивает поддержку защищенных SSL-соединений. Рассмотрение данного модуля выходит за рамки этой книги. Настройка модуля mod_ssl подробно рассмотрена в моей книге «Linux-сервер своими руками».

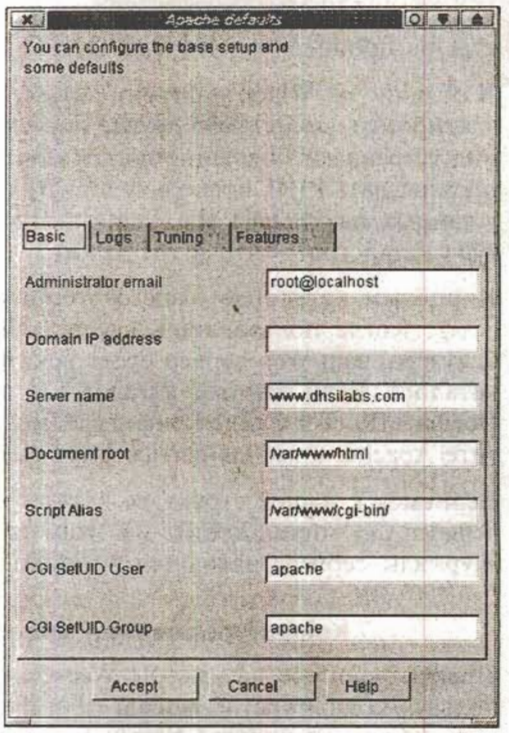


Рис. 4.3. *Основные параметры Apache*

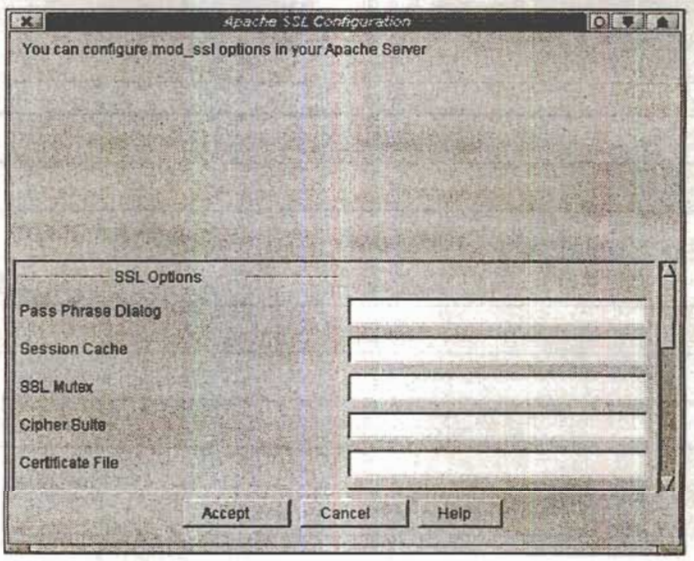


Рис. 4.4. *Конфигурирование модуля mod_ssl*

4.3.2. Установка MySQL

Установив Apache, перейдем к установке **MySQL**. Благодаря своей простоте сервер MySQL приобрел широкое распространение в сети Интернет. При создании Интернет-сервера вы просто не сможете обойтись без сервера MySQL: большинство провайдеров предоставляет хостинг вместе с сервером баз данных, большинство дизайнеров выбирают MySQL, потому что практически невозможно построить серьезный Интернет-проект без использования баз данных.

Я здесь не буду рассматривать технические характеристики сервера MySQL и не буду сравнивать его с другими серверами баз данных, такими как InterBase Server, IBM DB/2, Oracle. MySQL идеально подходит практически для любого Интернет-проекта. Естественно, если вы создаете распределенную систему обработки информации, вам лучше использовать InterBase Server или Oracle, поскольку эти СУБД являются более масштабируемыми, чем MySQL.

Итак, приступим к настройке сервера. Прежде всего, нужно установить пакеты, необходимые для работы MySQL.

Я использую MySQL версии 3.23, поэтому я установил такие пакеты (номера версий у вас могут быть другими, поэтому я заменил их иксами (x)):

1. MySQL_GPL-3.x.x
2. MySQL_GPL-client-3.x.x
3. MySQL_GPL-shared-libs-3.x
4. MySQL_GPL-bench-3.x.x
5. MySQL_GPL-resolveip-3.x.x

Проще всего воспользоваться командой **rpm -ih MySQL*** для установки сервера. После установки пакета нужно создать базу данных **mysql**. Скорее всего, она уже создана, но, чтобы окончательно убедиться в этом, введите команду:

```
mysql_install_db
```

Если основная база данных существует, программа сообщит вам об этом. Теперь нужно установить пароль для пользователя root. По умолчанию пользователь root не имеет пароля. Я не буду долго объяснять, как важен правильно заданный пароль для безопасности системы, не говоря уже об его отсутствии.

База данных **mysql** является системной базой данных и содержит следующие таблицы:

- таблицу **db**;
- таблицу **host**;
- таблицу **user**.

Сейчас нас интересует таблица **user**. Она содержит пароли всех пользователей, которые имеют право работать с сервером. На данном (начальном) этапе в базу данных внесен только один пользователь — root. Для изменения пароля запустите сервер командой:

```
safe_mysqld &
```

Эта команда запустит сервер в режиме демона и освободит консоль. Если все пакеты были установлены правильно, вы увидите сообщение:

```
mysql: ready for connections
```

Однако может быть и другое сообщение, свидетельствующее об удачном запуске. Затем введите команду:

```
mysql -u root mysql
```

Данная команда запускает клиент MySQL. При этом используется имя пользователя root, даже если вы работаете под другой учетной записью. Последний параметр определяет базу данных — **mysql**.

Измените пароль суперпользователя с помощью команды:

```
UPDATE user SET Password=PASSWORD('new_password') WHERE user='root';
```

Как вы заметили, это обычный SQL-запрос, обновляющий поле **Password** таблицы **user** для пользователя root. Теперь нужно, чтобы MySQL принял изменения. Для этого выполните еще один запрос SQL:

```
FLUSH PRIVILEGES;
```

Для принятия изменений можно также использовать программу **mysqladmin** с параметром **reload**. Вызвать программу можно так:

```
mysqladmin -p reload
```

Параметр **-p** вам обязательно нужно использовать, так как вы только что установили пароль для пользователя root. Выйти из клиента mysql вы можете, введя команду **quit**.

Установите права доступа к сценарию `/etc/rc.d/init.d/mysql`:

```
chmod +x /etc/rc.d/init.d/mysql
```

Теперь можете перезапустить сервер командой:

```
/etc/rc.d/init.d/mysql restart
```

Если вы забудете пароль, вы его уже не восстановите. Единственный выход из этого положения — удалить каталог `/var/lib/mysql/mysql` и создать базу mysql заново командой **mysql_install_db**.

Теперь вы уже не можете зарегистрироваться на сервере без пароля. Если вы введете команду **mysql -u root mysql**, то получите следующее сообщение:

```
ERROR: Access denied for user: 'root@localhost' (Using password: NO)
```

Для регистрации на сервере теперь нужно использовать команду **mysql -u root -p**. Параметр **-p** запросит пароль при регистрации.

Последнее, что вам осталось сделать — это добавить сервер MySQL в автозапуск. С этой целью перейдите в каталог `/etc/rc.d/rc3.d/` и создайте символическую ссылку на файл `/etc/rc.d/init.d/mysql`:

```
ln -s S14mysql /etc/rc.d/init.d/mysql
```

Префикс **S14** определяет очередность запуска сервера **mysql**. В данном случае он запустится после сервисов **network** (S10) и **portmap** (S11). У вас эти значения могут быть другими.

В своей работе демон **mysqld** использует файл журнала `/var/log/mysql.log`. Именно в него заносятся все транзакции, а также все команды, которые ввел пользователь. После установки сервера нужно внести пользователей, которые имеют право работать с сервером баз данных. Введите следующий запрос:

```
GRANT ALL PRIVILEGES ON *.* TO admin@localhost IDENTIFIED BY
'password' WITH GRANT OPTION;
```

Введенный вами запрос создаст пользователя admin, который будет иметь право выполнять любые операции со всеми базами данных. Данный пользователь будет имеет право подключаться к серверу с компьютера `localhost`, используя пароль password.

Маска *.* определяет, к каким базам данных и таблицам имеет право подключаться тот или иной пользователь. Первая звездочка определяет базу, а вторая — таблицу. Если вам нужно, чтобы пользователь `admin` имел право подключаться с любого хоста, используйте знак процента вместо имени хоста. В этом случае запрос будет выглядеть так:

```
GRANT ALL PRIVILEGES ON *.* TO admin@"%" IDENTIFIED BY ' password'
WITH GRANT OPTION;
```

Вместо всех полномочий вы можете определить, какие действия может выполнять с базой тот или иной пользователь. Если вы являетесь хостинг-провайдером и предоставляете доступ пользователю к его базе данных, то вы можете использовать следующий запрос:

```
GRANT CREATE,DROP,SELECT,INSERT,UPDATE,DELETE,INDEX ON user.*
TO user@% IDENTIFIED BY 'user_password';
```

Перед выполнением данного запроса необходимо создать базу данных **user**. Данный запрос позволяет пользователю user выполнять все операции с его базой данных.

Полный список полномочий представлен в табл. 4.3.

Полномочия пользователей сервера MySQL *Таблица 4.3*

Полномочия	Описание
SELECT, INSERT, UPDATE, DELETE	Одноименные операции с данными: пользователь имеет право просматривать, добавлять, модифицировать, удалять данные в таблицах базы данных
INDEX	Пользователь имеет право производить операции с индексами таблиц
REFERENCES	Пользователь имеет право работать со ссылками в базах данных и таблицах
CREATE, DROP	Создание и удаление таблиц и баз данных
GRANT, ALTER	Операции с полномочиями
RELOAD, SHUTDOWN, PROCESS	Пользователь имеет право перезагружать, останавливать сервер и просматривать все процессы (подключения)

Если запрос GRANT у вас не работает, то вы можете внести пользователя непосредственно в таблицу **user** базы данных **mysql**. Структура таблицы **user** выглядит следующим образом:

```
Host User Password Select_priv Insert_priv Update_priv Delete_priv
Create_priv Drop_priv Reload_priv Shutdown_priv Process_priv File_priv
```

Поля **Host**, **User**, **Password** — это, соответственно, узел, из которого пользователь может получить доступ, имя пользователя, пароль пользователя. Все остальные поля задают полномочия для пользователя. Если выполнение какой-нибудь операции разрешено пользователю, соответствующее поле должно быть равным «Y». В противном случае установите значение «N».

Например, нам нужно создать пользователя `admin`, который должен иметь все полномочия. Это можно сделать с помощью такого запроса SQL:

```
INSERT INTO user(Host,User,Password,Select_priv,Insert_priv,
Update_priv, Delete_priv, Create_priv, Drop_priv, Reload_priv,
Shutdown_priv, Process_priv, File_priv) VALUES ('localhost',
'admin', password('4td56ls12'),'Y','Y','Y','Y','Y','Y','Y','Y',
'Y','Y');
```

При вводе запроса обратите внимание на регистр названий полей. Сервер MySQL *различает прописные и строчные буквы!* С помощью вышеприведенного запроса был создан пользователь `admin`, который имеет право регистрироваться на сервере только из узла `localhost`. Если вам

нужно разрешить регистрацию из любого узла сети, используйте знак процента, однако это не совсем корректно с точки зрения безопасности. Пользователь admin обладает всеми возможными привилегиями. Пароль пользователя — 4td56ls12.

Для создания обыкновенного пользователя используйте следующий запрос:

```
INSERT INTO user(Host, User, Password, Select_priv,
Insert_priv, Update_priv, Delete_priv) VALUES('%', 'user',
password('123456'),'Y','Y','Y','Y');
```

4.3.3. Установка PHP и настройка связки Apache + PHP + MySQL

Сейчас мы произведем не только установку PHP, которая не вызывает особых проблем, а и настройку связки Apache+PHP+MySQL. Настроить данную связку, которая очень полезна при Web-программировании, можно двумя способами. Первый из них — это использовать программы, которые входят в состав дистрибутива и, как правило, устанавливаются из пакетов RPM. Второй способ заключается в загрузке последних версий Apache, MySQL и PHP и в самостоятельной их сборке из исходных текстов. Первый способ я могу порекомендовать начинающим пользователям, так как он более прост. Второй способ из-за своей сложности не рассматривается в этой книге.

Могу сразу обрадовать пользователей дистрибутивов Red Hat 7.2 и Mandrake 8.1 (или более поздних версий): вам не нужно настраивать первую часть связки — все настраивается во время установки системы. Нам также понадобится пакет **php-mysql**, обеспечивающий поддержку сервера MySQL языком PHP. В состав дистрибутива Red Hat 7.2 входит сервер Apache 1.3.20-16, а при установке системы устанавливаются библиотека **gd** и интерпретатор **php** версии 4.0.6, а также модуль для сервера Apache (пакет mod_php). Поэтому вы можете сразу приступить к тестированию связки Apache + PHP (см. файл test.php ниже).

Примечание.

В состав дистрибутива Red Hat 9.0 входят:

- сервер Apache версии 2.0 (пакет называется httpd-2.0);
- сервер MySQL версии 3.23;
- PHP версии 4.22.

Загрузить последнюю версию PHP 5 можно по адресу http://www.php.net/downloads.php.

Убедитесь, что сервер Apache установлен и корректно функционирует:

```
lynx http://localhost
```

Текстовый браузер **lynx** должен отобразить стартовую страницу Apache. После успешной проверки работы сервера остановите его командой:

```
/etc/init.d/httpd stop
```

Проверьте наличие библиотеки **gd** — она необходима для работы с графикой в PHP:

```
rpm -qa | grep gd
```

Если библиотека **gd** не установлена, установите ее командой, предварительно указав ту версию библиотеки, которая у вас имеется (я использую версию 1.8.4):

```
rpm -ihv gd-1.8.4-4.i386.rpm
```

Установите пакет **php**, если вы его еще не установили:

```
rpm -Uh php*
```

После этого установите модуль Apache, обеспечивающий поддержку PHP:

```
rpm -Uh mod_php*
```

Данный модуль должен входить в состав дистрибутива и обычно находится на первом инсталляционном диске. Затем в файле `httpd.conf` найдите и раскомментируйте следующую строчку. После этого файлы с расширением .php будут правильно обрабатываться сервером:

```
AddType application/x-httpd-php4 .php
```

и/или

```
AddType application/x-httpd-php5 .php
```

Теперь можно проверять правильность настройки двух компонент связки: Apache и PHP.

Напомню, что в большинстве современных серверных дистрибутивах сервер Apache уже установлен вместе с модулем **mod_php**, поэтому выполнять вышеизложенные шаги совсем необязательно, а можно только проверить корректность работы модуля **mod_php**.

Для проверки работы модуля **mod_php** создайте тестовый файл `test.php` с таким содержимым:

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

Этот файл сохраните в каталоге **DocumentRoot** сервера Apache. Обычно это каталог `/var/www/html`. После этого запустите любой браузер и введите следующий адрес:

```
http://localhost/test.php
```

При этом на экране вы должны увидеть различную информацию о PHP, сервере Apache и других компонентах и библиотеках, например, о библиотеке **gd** (см. рис. 4.5).

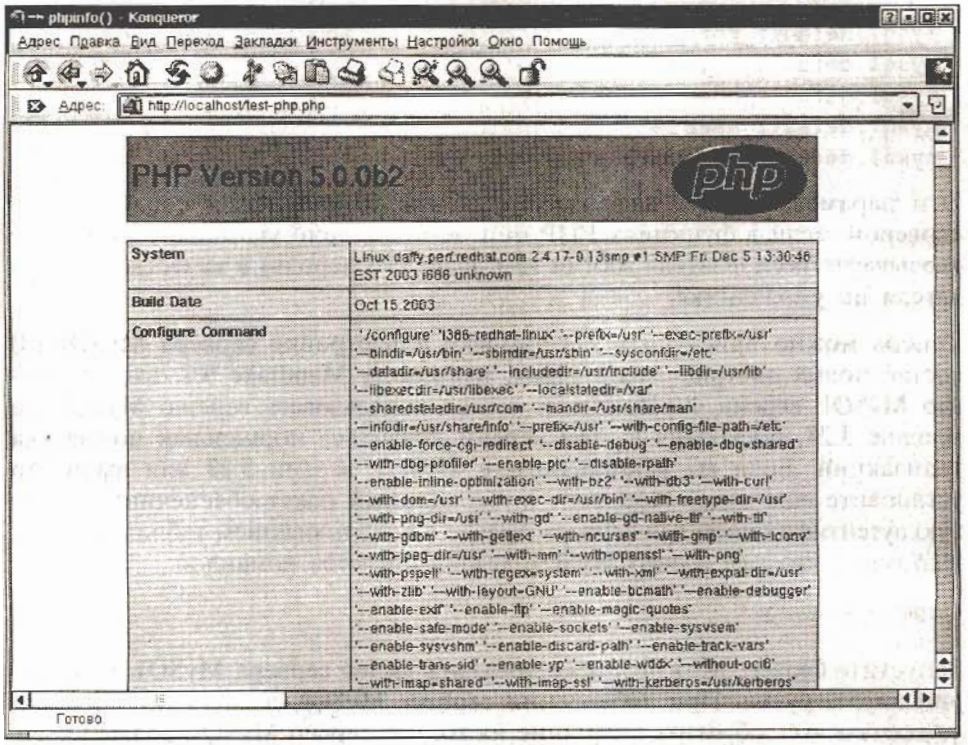


Рис. 4.5. *Функция phpinfo()*

Функция **phpinfo()** очень информативна: внимательно изучив информацию, которую она предоставляет, вы много узнаете о своем Web-сервере. К тому же эту информацию можно использовать при Web-программировании.

Теперь немного настроим PHP. С помощью функции **phpinfo()** узнайте, где расположен инициализационный файл PHP. Обычно он называется `php.ini` и находится в каталоге `/etc`. Откройте этот файл в любом тек-

стовом редакторе и раскомментируйте следующую строку, убедившись, что в вашей системе есть файл `mysql.so` (он устанавливается при установке php-mysql):

```
extension=mysql.so
```

Точнее, проверьте, раскомментирована ли данная строка. При установке пакета **php-mysql** данная строка должна быть автоматически раскомментирована.

После этого перейдите в секцию MySQL файла `php.ini` и установите параметры сервера MySQL по умолчанию:

```
mysql.default_port =
mysql.default_socket =
mysql.default_host =localhost
mysql.default_user =
mysql.default_password =
```

Эти параметры будут использоваться при установлении соединения с сервером, если в функциях PHP они не будут явно указаны. Никогда не указывайте пользователя root (а тем более его пароль) в качестве пользователя по умолчанию!

Теперь можно приступить к установке и настройке сервера MySQL. В состав новых дистрибутивов Linux, например, Mandrake 9.2, входит сервер MySQL версии 4.0.15. Имеет смысл использовать версию MySQL не младше 3.23, поскольку в этой версии появилась нормальная поддержка транзакций. Если вы устанавливаете MySQL версии 3.23 или выше, то установите еще пакет mod_auth_mysql. Данный пакет обеспечивает базовую аутентификацию для сервера Apache с использованием таблиц MySQL. Напомню, что сам сервер MySQL устанавливается командой:

```
rpm -ih mysql*
```

Запустите сервер. Можно также добавить вызов сервера MySQL в сценарии автозагрузки. При добавлении сервера MySQL в сценарии загрузки (`/etc/rc.d/`) обратите внимание на то, что сервер MySQL должен быть запущен ПЕРЕД сервером Apache. Представьте такую ситуацию: сначала запускается сервер Apache и сразу же получает запрос от клиента к базе данных MySQL, а сервер баз данных еще не запущен. Даже если запускать сервер MySQL сразу после Apache, то все равно ему понадобится еще некоторое дополнительное время для инициализации. Не забудьте установить пароль для пользователя root сервера MySQL:

```
mysql -u root -e "update user set password=password('new_password')
where user='root'" mysql
```

Перезагрузите сервер:

```
mysqladmin reload
```

4.3.4. Тестирование созданной конфигурации

Теперь можно проверить работу всей связки Apache + PHP + MySQL. С этой целью создайте небольшой тестовый файл `mysql_test.php` в каталоге `/var/www/html` (см. листинг 4.13).

Листинг 4.13. Файл mysql_test.php

```
<?
// Используется имя пользователя root и пароль passwd
if(!mysql_connect("localhost","root","password"))
{
echo "Не могу соединиться с сервером\n";
echo mysql_error();
exit;
}
echo "Работает!"
?>
```

Как вы уже успели догадаться, если в окне браузера вы увидите слово «Работает!», значит вы все сделали правильно.

4.4. Установка Apache + PHP + MySQL под Windows

4.4.1. Установка и настройка Apache

Для установки Apache for Windows посетите сайт `http://www.apache.org` и загрузите версию Apache для Windows. Установка сервера проблем не вызывает: просто запустите программу установки и выберите тип установки Typical. Еще от вас потребуется указать каталог, в который будет производиться установка. По умолчанию будет установлен в каталог «C:\Program Files\Apache Group\Apache». В остальных случаях просто нажимайте кнопку «Next». В общем обычный процесс установки.

По окончании установки Apache необходимо настроить. По умолчанию корневым каталогом для HTML-документов будет `C:\Program Files\Apache Group\Apache\htdocs`. Сами понимаете, такое расположение не очень удобно, поэтому создайте каталог `C:\www`, в котором будут находиться наши HTML-файлы. В этом же каталоге создайте два подкаталога — `cgi` и `html`. В первом будут находится CGI-приложения, а во втором — HTML-файлы.

Итак, откройте в Блокноте файл `C:\Program Files\Apache Group\Apache\conf\httpd.conf` и отредактируйте следующие директивы:

- **ServerName** — установите имя сервера. При этом обращаю ваше внимание, что директива ServerName по умолчанию закомментирована, то есть игнорируется сервером. Поэтому необходимо раскомментировать эту директиву, убрав перед ней значок #.
- **DocumentRoot** — установите значение `C:/www/html` (используйте прямую наклонную черту – «/», а не обратную, как в Windows):

  ```
  DocumentRoot c:/www/html
  ```

  Таким образом, вы укажете каталог, в котором будут размещаться ваши HTML-файлы.
- Установите директивы **ScriptAlias** так:

  ```
  ScriptAlias /cgi/ "c:/www/cgi/"
  ScriptAlias /cgi-bin/ "c:/www/cgi/"
  ```

  Благодаря этому вы создадите два псевдонима для каталога `c:/www/cgi`. Таким образом, если будет указан путь `http://localhost/cgi` или `http://localhost/cgi-bin`, то на самом деле обращение будет к каталогу `c:/www/cgi`.
- Найдите секцию `<Directory "C:/Program Files/Apache Group/Apache/htdocs">` и замените ее на следующую:

  ```
  <Directory "c:/www">
      Options Indexes Includes
      AllowOverride All
      Allow from all
  </Directory>
  ```

  Таким образом, вы зададите ряд настроек по умолчанию для каталога `c:\www` и всех его подкаталогов. А именно: автоматическую генерацию списка содержимого каталога при просмотре его в браузере, поддержку SSI, а также возможность использовать файлы `.access` для индивидуальной настройки каталогов.
- Убедитесь, что у вас есть директива **AddHandler**:

  ```
  AddHandler cgi-script .bat .exe .cgi
  ```

  Эта директива указывает Apache на то, что файл с расширениями `.bat` `.exe` и `.cgi` следует воспринимать как CGI-модули.
- Установите параметры SSI:

  ```
  Addtype text/html .shtml
  AddHandler server-parsed .shtml .html .htm
  ```

  Благодаря этому файлы с указанными расширениями будут обрабатываться SSI.
- Не забудьте сохранить отредактированный файл.

4.4.2. Тестирование Apache под Windows

Теперь можно запустить Apache. Для этого в **Блокноте** создайте файл `apache.bat` и поместите его в каталог `c:\windows\command` (листинг 4.14).

Листинг 4.14. Файл apache.bat

```
@echo off
C:
cd "\Program Files\Apache Group\Apache"
apache.exe
```

Откройте Сеанс MS-DOS и в командной строке введите команду **apache**. Вы должны увидеть:

```
Apache/1.3.14 (Win32) running...
```

Теперь можно проверить корректность работы сервера. Запустите Internet Explorer или любой другой браузер и введите: `http://localhost`. Вы должны увидеть стартовую страницу Apache. Теперь проверим CGI и SSI. Для этого создайте файл `test.bat` (см. листинг 4.15) и поместите его в каталог `c:\www\cgi`:

Листинг 4.15. Тестовый файл test.bat

```
@echo off
echo Content-type: text/html
echo.
echo.
echo Hello!
```

В окне браузера введите `http://localhost/cgi/test.bat` — вы увидите слово `Hello!` Для проверки SSI нужно создать файл `test.html` и поместить его в каталог `c:\www\html`:

```
SSI Test<br>
<!--#include virtual="/index.html" -->
```

Примечание.

SSI (Server Side Include) — включение со стороны сервера, позволяет включать в Web-страницу код HTML, находящийся в других файлах.

В окне браузера при вызове файла `test.html` вы увидите SSI Test, а затем — стартовую страницу Apache. Если на каком-нибудь этапе произошла ошибка, попробуйте заново сконфигурировать CGI и SSI.

4.4.3. Установка PHP и настройка Apache для работы с PHP

После установки Apache возьмемся за установку PHP 5 (или PHP 4 — как вам будет угодно). Версию PHP 5 для Windows можно скачать по адресу `http://www.php.net` (последняя версия PHP доступна по адресу `http://www.php.net/downloads.php`).

Запустите программу установки, которая по умолчанию установит PHP 5 в каталог `C:\Program Files\PHP 5\`. Однако в процессе установки вы сможете выбрать другой каталог.

Сразу после установки откройте файл `php.ini` (он будет в каталоге, куда вы установили PHP) и отредактируйте следующие параметры:

```
[mail function]
; For Win32 only.
SMTP = localhost

; For Win32 only.
sendmail_from = me@localhost.com

; For Unix only.  You may supply arguments as well (default:
'sendmail -t -i').
;sendmail_path =
```

Параметр SMTP задает имя SMTP-сервера. По умолчанию в качестве SMTP-сервера используется локальный компьютер — это хорошо для Linux, но в Windows вам нужно указать SMTP-сервер провайдера, потому что вряд ли у вас будет установлен почтовик под Windows. Если вы не сделаете этого, функция **mail()**, отправляющая сообщения, работать не будет.

В Linux/Unix можно также указать путь к программе **sendmail**, если она установлена в каталог, отличный от каталога `/usr/sbin`.

После этого откройте файл `httpd.conf` (файл конфигурации Apache) и найдите в нем строку:

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

Убедитесь, что данная строка не закомментирована, то есть перед ней не стоит значок #. Если он стоит — уберите его. Таким образом, всем файлам с расширением **.php** будет поставлен в соответствие *MIME-тип* **application/x-httpd-php**.

Сразу же после этой строки добавьте строки:

```
ScriptAlias /_php_/ "c:/Program Files/PHP5/"
Action application/x-httpd-php "/_php_/php.exe"
```

В этих двух строках мы сначала создаем псевдоним _php_ для каталога `c:/Program Files/PHP5/`, а затем связываем все файлы типа `application/x-httpd-php` с обработчиком `php.exe`.

Теперь можно протестировать PHP. Для этого создайте файл `test.php`:

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

Данный файл нужно поместить в каталог `c:\www\html`. В браузере введите `http://localhost/test.php`. В итоге вы должны увидеть результат работы функции **phpinfo()**.

4.4.4. Установка MySQL

Сервер MySQL можно загрузить по адресу `http://www.mysql.com`. Программа установки установит сервер в каталог `C:\Progam Files\MySQL\`. Сейчас немного модифицируем наш файл server.bat так, чтобы он запускал сразу два сервера:

```
@echo off
"C:\Program Files\MySQL\bin\mysqld"
start /m "C:\Program Files\Apache Group\Apache\apache"
```

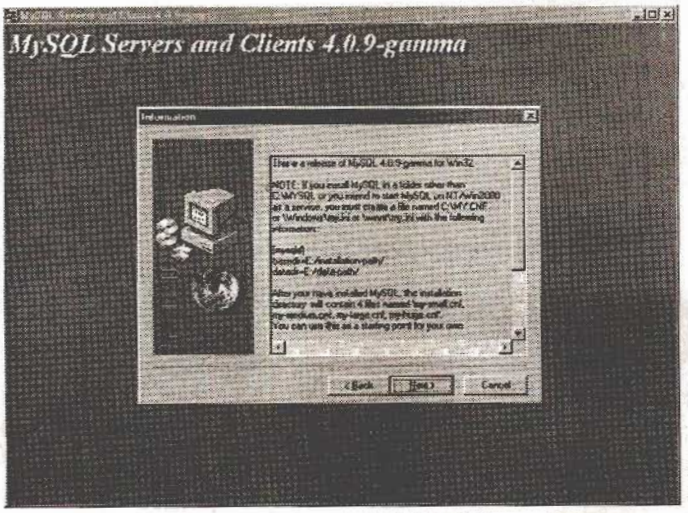


Рис. 4.6. *Установка сервера MySQL для Windows*

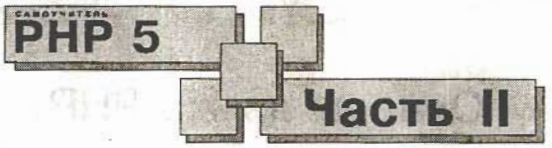
Если вы работаете в Windows NT или 2000, `server.bat` должен выглядеть так, как это показано в следующем листинге:

```
@echo off
start C:\Progra~1\MySQL\bin\mysqld-nt -standalone
C:\Progra~1\Apache~1\Apache\apache -k start
```

Теперь для запуска Apache и MySQL нужно ввести команду **server**. Создадим файл `shutdown.bat`, который будет останавливать оба сервера:

```
@echo off
C:\Progra~1\Apache~1\Apache\apache -k shutdown
C:\Progra~1\MySQL\bin\mysqladmin -u root shutdown
```

Для тестирования Apache + MySQL + PHP можно использовать тот же самый файл, который мы использовали для тестирования конфигурации под Linux.



Основные понятия, функции и механизмы PHP

Глава 5. Основы языка PHP

Глава 6. Передача параметров сценариям

Глава 7. Массивы и списки

Глава 8. Функции пользователя. Создание и использование собственных функций

Глава 9. Функции для работы со строками

Глава 10. Полезные стандартные функции

Глава 11. Работа с файлами и каталогами

Глава 5

Основы языка PHP

5.1. Структура программ в PHP. Первая программа

5.1.1. Стандартный вид PHP-сценария

Общие понятия и описание PHP

Для начала познакомимся с самим PHP. Язык программирования PHP специально предназначен для работы в Интернет, сочетающий достоинства языков C и Perl. Синтаксис языка очень сильно напоминает C — если вы программируете на C, то скоро вы это заметите.

В официальной документации язык PHP подается как server-side HTML-embedded scripting language, то есть, как встраиваемый в HTML скриптовый язык с обработкой на сервере. Это позволяет сразу же иметь ввиду следующее:

- Обработка PHP-кода производится на стороне сервера еще до того, как Web-страница будет передана браузеру. Это, кстати, отличает язык PHP от языка JavaScript.
- PHP-код может быть непосредственно встроен в HTML-код страницы. Этим он отличается от Perl и отличается в лучшую сторону.

Мы знаем, что языки программирования бывают двух видов: интерпретаторы и компиляторы. Чем же является PHP? Чтобы выяснить это, нам нужно разобраться в терминологии. Программа, переводящая код, написанный на одном языке программирования, на другой называется транслятором. Например, программа p2c, которая переводит Pascal-код в C-код, является транслятором. Компилятор — это тоже транслятор. Он переводит код, написанный на языке высокого уровня, в машинный код. Язык высокого уровня понятен человеку и близок к человеческому (Pascal, C, PHP), а машинный код (0 и 1) понятен компьютеру — с двоичными данными компьютеру проще работать. В результате работы компилятора, как правило, создается двоичный исполняемый файл (в Windows/DOS — с расширением .exe или .com), который можно выполнять без помощи компилятора.

Интерпретатор — это совершенно другой тип языка программирования, не имеющий ничего общего с компилятором. Интерпретатор ничего не переводит, а только исполняет код. Он анализирует код программы и исполняет каждую ее строчку. Например, предположим, что интерпретатор нашел строку «A = A + 7». Он анализирует и проверяет ее корректность, а затем ищет переменную A. Если переменная была объявлена до строки «A = A + 7», то интерпретатор увеличивает ее значение на 7 — он выполняет код. Была бы вместо строки «A = A + 7» строка `print "Hello"`, наш интерпретатор вывел бы на экран строку «Hello». Выполнив код, интерпретатор не создает исполняемый файл, поэтому для следующего запуска вашего кода опять нужен интерпретатор.

По производительности интерпретаторы значительно уступают компилятору, поскольку машинный код выполняется намного быстрее. Зато интерпретаторы позволяют полностью контролировать программу во время ее исполнения.

PHP не является ни компилятором, ни интерпретатором. PHP представляет собою что-то среднее, между компилятором и интерпретатором. Рассмотрим, как PHP обрабатывает сценарии. На вход PHP подается сценарий. Он переводит (транслирует) его в специальный *байт-код* (внутреннее представление). Затем PHP выполняет *байт-код (а не код самой программы!)*, при этом он не создает исполняемый файл. Байт-код значительно компактнее обыкновенного кода программы, поэтому его легче (и быстрее) интерпретировать (выполнять). Как видите, PHP больше является интерпретатором, нежели компилятором. Для чего нужна была такая двойная работа — сначала переводить во внутреннее представление, а потом — выполнять? Неужели не проще сразу выполнить *каждую строку программы*?

Вдумайтесь в это выражение *«каждую строку программы»*. Представим, что у нас есть цикл:

```
for (I=0;I<10;I++)
{
оператор_1;
…
оператор_100;
}
```

Наш цикл будет «крутиться» не бесконечно, а всего лишь десять раз. За каждый из этих десять проходов интерпретатор должен *проанализировать и выполнить* СТО строк кода. Итого ему нужно проанализировать и выполнить (я не зря дважды упоминаю эту фразу) 10×100 = 1000 строк кода. Если перевести один раз весь цикл в байт-код, то анализировать ему придется в 10 раз меньше! А это значит, что ваши сценарии будут выполняться в 10 раз быстрее.

Выходит, что PHP является транслирующим интерпретатором или интерпретирующим транслятором — как кому больше нравится. Стоит отметить, что PHP версии 3 был «чистым» интерпретатором, и только с появлением четвертой версии сценарии стали выполняться значительно быстрее. Пятая версия, как и четвертая, является интерпретирующим транслятором.

Однако, чтобы не делали разработчики PHP, откомпилированные двоичные исполняемые файлы работают значительно быстрее — в десятки, а может и в сотни раз. Но производительность PHP вполне достаточна для создания Интернет-приложений. Сейчас объясню почему.

Пусть вышеупомянутый цикл, переписанный на языке C, выполняется за 0.1 секунды. На языке PHP он будет выполняться в 10 раз дольше — за 1 секунду. Однако передача данных пользователю — результатов работы цикла — будет длиться 3 секунды. Получаем, соответственно, в сумме 3.1 и 4 секунды. Согласитесь, разница между 3.1 и 4 секунды незначительна — как пользователь Интернета вы даже ее не почувствуете. Даже если у вас быстрый выделенный канал, разница все равно не будет чувствоваться — может сказаться нагрузка на сервер или другие факторы, даже не зависящие от PHP.

Использование интерпретатора (а значит и PHP) имеет свои неоспоримые преимущества:

- Вам не нужно заботиться об освобождении выделенной памяти, вам не нужно закрывать файл по окончании работы с ним — за вас все сделает интерпретатор, поскольку ваша программа выполняется под его бдительным контролем.
- Вам не нужно думать о типах переменных, вам также не нужно объявлять переменную до его первого использования.
- Отладка программы и обнаружение ошибок существенно упрощаются — интерпретатор непременно сообщит вам, если что-то пошло не так.
- Вы можете написать программу, которая будет писать другую программу (да, это так), а потом включит в себя код только что написанной программы и выполнит его. Об этой возможности мы поговорим отдельно, когда наступит время.

Пишем первую программу (скрипт) на PHP

Согласно традиции, знакомство с любым языком программирования начинается с программы Hello, World! Эта программа ничего особого не делает, она только выводит на экран надпись «Hello World!». Мы сейчас немного отступим от традиции и напишем программу, которая будет здороваться не со всем миром (все равно этого никто не услышит), а только с нами. Текст программы приведен в листинге 5.1.

Листинг 5.1. Первая программа (скрипт) на PHP

```
<?
echo "<html><body><h1>";
echo "Hello, $my_name";
echo "</h1></body></html>";
?>
```

Прежде, чем запустить программу, ее нужно правильно «установить» на сервере. Для этого сохраните сценарий под именем `first.php` и скопируйте его в каталог DocumentRoot вашего сервера. Если вы работаете под Linux, таким каталогом будет `/var/www/html` (в старых версиях — `/home/httpd/html`). Если вы работаете под Windows и следовали моим рекомендациям (см. гл. 4), каталогом DocumentRoot будет `c:\www\html`.

В отличие от обыкновенных CGI-программ, сценарии PHP являются для сервера обыкновенными документами и их не нужно помещать в каталог `cgi-bin`. Также не нужно делать их исполняемыми (в случае, если вы работаете под Linux).

Выполнив вышеуказанные рекомендации по установке, в строке браузера введите `http://localhost/first.php?my_name=Denis`. В результате в окне браузера вы увидите `Hello, Denis` (см. рис. 5.1).

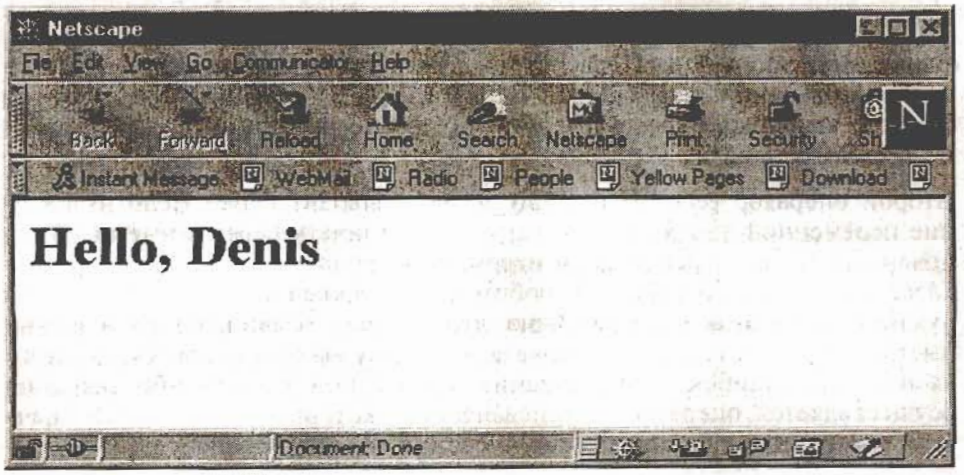


Рис. 5.1. *Первая PHP-программа*

Если теперь страницу, отображенную в браузере, просмотреть в виде HTML-кода, то можно увидеть следующее:

```
<html><body><h1>
Hello, Denis
</h1></body></html>
```

Как видите PHP-код выполнился, подставил нужное значение переменной `$my_name` и выдал готовую HTML-страницу.

Это мы PHP-сценарий запустили через CGI-интерфейс. При этом мы передали переменной `$my_name` значение Denis. Однако любую PHP-программу можно запустить так: **php file.php**. При этом CGI-переменные будут недоступны — ведь вы запустили программу из командной строки. Демонстрационные PHP-программы, не требующие CGI, в этой книге я буду запускать именно так.

Примечание.

Вводить текст программы лучше в текстовом редакторе, поддерживающем подсветку синтаксиса PHP. Если вы работаете под Linux, могу порекомендовать три редактора: K Write (команда kwrite), Kate (kate) и Quanta (quanta). Под Windows также не составляет труда найти такой редактор (в Интернете). Если вы не найдете редактор с подсветкой синтаксиса PHP, можно использовать редактор с подсветкой синтаксиса языка C.

Теперь рассмотрим саму программу. Код PHP заключается в специальные теги **<?** и **?>**. После тега начала сценария (<?) следует первый оператор — **echo**. Оператор **echo** осуществляет вывод информации. Его с полной уверенностью можно назвать самым главным оператором — ведь без него был бы невозможен вывод информации в браузер и пользователь не увидел бы результата работы сценария.

Выводимая оператором **echo** строка заключается в кавычки. Первый оператор выводит теги HTML, BODY и H1. В принципе, можно было бы и не выводить первые два тега, но нужно привыкнуть к дисциплине с самого начала.

Второй оператор **echo "Hello, $my_name";** выводит слово Hello и значение переменной `$my_name`. Чтобы как-то отличать переменные от обыкновенных строк, перед именем переменной ставится знак доллара. И это касается не только строк, в любом случае перед именем переменной нужно ставить знак доллара. Знаю, что это непривычно, но вы довольно быстро привыкните к этому. Правда, поначалу вы будете допускать много «денежных» ошибок. Присваивание переменной какого-либо значения осуществляется оператором присваивания, который обозначается значком *«=» (равно)*. Примеры использования переменных:

```
$i=0;
$d=7;
$i=5 + $d// $i=12
```

Еще один пример:

```
$word1 = "Hello";
$word2 = "world";

echo "$word1, $word2";
echo "$word1, word2";
```

Первый оператор выведет в браузер строку «Hello, world», а второй — «Hello, word2». В этом примере мы определили две переменные — `$word1` и `$word2`. Так как мы забыли дописать символ доллара во втором операторе, PHP воспринял «word2» как обыкновенную строку, а не как переменную.

Обратите внимание, что в нашем первом примере (`first.php`) мы не объявили переменную `$my_name`. Ее объявил автоматически сам PHP — ведь это параметр, переданный сценарию браузером. В этом заключается еще одно преимущество PHP перед другими языками программирования.

Вы когда-нибудь программировали на C или Pascal? Обращение к параметру, переданному программе, осуществляется по номеру, например, прочитать значение параметра номер 1 можно, вызвав функцию **ParamStr(1)**, а не по имени. Ощущаете разницу? То есть, если мы программируем на Pascal или C, нам обязательно нужно знать номер параметра. Например, нам нужно передать программе два параметра — файл-источник и файл-назначение. Программа должна скопировать данные из источника в назначение. Если мы перепутаем порядок файлов, может произойти непоправимое — файл-источник будет перезаписан файлом-назначение. В PHP нам нет никакой разницы, как будут следовать параметры:

`src=file1&dest=file2` или `dest=file2&src=file1`
(так передаются браузером параметры нашему сценарию).

Очень удобно! И это не говоря уже о том, что PHP сам выполняет разбор переданной строки, объявляет и инициализирует переменные.

Третий оператор нашей программы в особых комментариях не нуждается, как и тег окончания PHP-программы (?>).

Модифицируем немного наш текст программы `first.php`, чтобы он выглядел так:

```
<?
echo "<html><body><h1>";
echo "Hello, $my_name";
echo "<br> $var";
echo "</h1></body></html>";
?>
```

Если мы не передадим программе параметр `$var`, то есть, как обычно, запустим программу только с параметром `$my_name`, то в окне браузера мы увидим сообщение о том, что переменная `$var` не определена. Для того чтобы избавиться от этой ошибки, непосредственно перед оператором `echo "<br> $var";` добавьте строку:

```
$var=" ";
```

Переменная будет объявлена и инициализирована. В PHP, как и в C++, вы можете объявлять переменные в любом месте сценария, но до места первого использования переменной. Как вы уже знаете, любой другой интерпретатор, например, Perl, в подобной ситуации сгенерировал бы сообщение 500: «Ошибка сценария». Интерпретатор PHP решает проблему просто и элегантно, избавляя тем самым Web-программиста от просмотра многотомных журналов Web-сервера.

5.1.2. PHP-сценарии в теле HTML-документов

Мы только что рассмотрели структуру PHP-программы, и можно было бы перейти к рассмотрению переменных. Однако PHP-сценарии можно записывать и по-другому. Чтобы вы лишний раз не листали книгу, напомню текст сценария `first.php`:

```
<?
echo "<html><body><h1>";
echo "Hello, $my_name";
echo "</h1></body></html>";
?>
```

Этот же сценарий можно записать так:

```
<html><body><h1>
<? echo "Hello, $my_name"; ?>
</h1></body></html>
```

Удивлены? Обыкновенный HTML-код корректно обрабатывается интерпретатором, поэтому у вас не будет ошибок при выполнении этого сценария.

Давайте разберемся, как правильно включать HTML-код в состав вашего сценария. Если PHP встречает скобки <? ?>, то он считает, что они содержат PHP-код и пытается выполнить его. Все, что находится за пределами скобок, считается простым текстом и передается браузеру безо всяких изменений. Вот небольшой пример:

```
<html><body>
<? if ($REMOTE_ADDR=="127.0.0.1") {?>
<h1>Добро пожаловать домой!</h1>
<? } else { ?>
```

```
<h1>Привет, гость!</h1>
<? } ?>
</html></body>
```

Если IP-адрес удаленного компьютера равен 127.0.0.1, мы выводим HTML-код "<h1>Добро пожаловать домой!</h1>" без использования оператора **echo**. В противном случае, мы выводим другое сообщение – "<h1>Привет, гость!</h1>". Таким образом, теги <h1> и </h1> воспринимаются и выводятся как обычный текст, а не как htmtl-теги заголовка.

Примечание.

В приведенном примере был использован условный оператор if, который позволяет выполнять определенные условия в зависимости от истинности или ложности условия. После условия указывается действие, которое следует выполнять в случае его истинности, а после слова else — в случае ложности. Подробнее условный оператор будет рассмотрен далее.

При включении HTML-кода в ваши сценарии будьте внимательны, потому что легко запутаться. Вообще я рекомендую выводить HTML-код только с помощью **echo**.

5.1.3. Комментарии в PHP-сценариях

Очень немаловажный элемент любой программы — это комментарии. Комментарии в PHP могут быть трех типов:

```
# Это комментарий

// Это тоже комментарий

/*
А это
длинный многострочный
комментарий
*/
```

Я предпочитаю использовать комментарии второго и третьего типа. Хотя какой из них использовать — дело вкуса.

5.2. Переменные и типы данных

Практически во всех языках программирования есть такое понятие, как переменная. Почему я употребил выражение «Практически во всех»? Существуют и такие языки, в которых нет переменных, как таковых. К счастью, PHP не относится к таким языкам — он самый обыкновенный язык, с точки зрения наличия переменных.

Переменная — это область оперативной памяти, доступ к которым осуществляется по имени. Все данные, с которыми работает программа, хранятся в виде переменных.

Правила задания переменных

Как уже отмечалось выше, в PHP обязательно перед именем переменной нужно ставить знак доллара — так интерпретатору легче выделить переменные из «серой массы» текста программы.

В официальной документации PHP сказано, что имена переменных могут состоять не только из латинских букв, но из любых символов, код которых больше 127. Это означает, что имена переменных могут содержать русские буквы. Однако я настоятельно советую вам не использовать русскоязычные имена — ведь в разных кодировках буквы кириллицы имеют разные коды.

Определим правила задания переменных в PHP:

- Имя переменной должно начинаться со знака доллара $.
- Имя переменной не должно содержать никаких других символов, кроме символов латинского алфавита, цифр и знака подчеркивания. Если вы будете использовать это правило, вы существенно облегчите себе жизнь.
- Имена переменных в PHP, как и в C, чувствительны к регистру символов, то есть переменные `$a` и `$A` — это совершенно разные переменные.
- Объявлять переменную можно в любом месте программы, но до места первого ее использования.

При объявлении переменных можно не скупиться — объявляйте столько, сколько вам нужно. Привычные методы оптимизации, которые применялись 20 лет назад (например, чрезмерная экономия памяти — чем меньше переменных использует программа, тем лучше), во времена первых компьютеров, не сработают: ваша программа не будет работать быстрее, если вы будете в ущерб собственному комфорту использовать одну переменную вместо 5. К тому же можно наделать ошибок, которые очень трудно найти.

Типы данных

Переменные в PHP могут содержать все, что угодно — любую информацию. Исключение составляют только константы, которые могут содержать только число или строку.

Такого понятия как указатель (это такая переменная, которая содержит не конкретное значение, а указывает на область памяти, где находится это значение), хорошо знакомого нам из языка C, вообще не существует. При присваивании переменная копируется «один в один», какую бы

структуру она не имела. Предположим, что у нас есть две переменные – массив и строка. Если мы первой переменной присвоим значение второй, то наш массив будет навсегда потерян, а переменная, которая содержала массив, будет содержать строку — это и означает, что копируется не только значение переменной, как в других языках, но и структура. Правда, начиная с 4-й версии, в PHP появились жесткие и мягкие ссылки, но о них мы поговорим позже.

Рассмотрим следующее объявление переменных:

```
$a = 10;
$A = "Hello";
```

Как вы видите, при объявлении переменных мы не указали их тип. Выбор типа осуществляется самим интерпретатором. Однако иногда PHP может ошибиться и сопоставить переменной неправильный тип. Поэтому нужно задать тип явно. О том, как это сделать, мы поговорим чуть позже, а пока рассмотрим основные типы переменных (табл. 5.1).

Типы переменных в PHP *Таблица 5.1*

Тип переменной	Описание
integer	Целое число. Диапазон значений: от –2 147 483 648 до –2 147 483 647 (32 бита). Данный тип соответствует типу long int в языке C (longint – в Pascal)
double (или float)	Вещественное число очень большой точности
string	Строка. Это очень важный тип данных, поскольку в большинстве случаев сценарии занимаются обработкой каких-либо строк, поэтому строкам посвящена целая глава
array	Массив. О массивах мы также поговорим в отдельной главе
object	Язык PHP поддерживает принципы объектно-ориентированного программирования, хотя и самые простые. Программист может создавать свои объекты и использовать их в программе. Доступ к отдельным элементам объекта осуществляется с помощью оператора ->
bool	Логический тип данных. Переменные этого типа могут принимать только одно из двух значений: true (истина) или false (ложь)

Функции определения и задания типа переменных

Язык PHP предоставляет много средств для определения типа переменных. Вы можете использовать целых семь функций для определения типа:

- **is_int($x) или is_integer($x)** — возвращает `true`, если переданная переменная — целое число.
- **is_double($x) или is_float($x)** — возвращает `true`, если переданная переменная — вещественное число.
- **is_string($x)** — возвращает `true`, если переданная переменная — строка.
- **is_array($x)** — возвращает `true`, если переданная переменная — массив.

- **is_object($x)** — возвращает `true`, если переменная является объектом.
- **is_bool($x)** — возвращает `true`, если переменная объявлена, как логическая.
- **gettype($x)** — возвращает строки, соответствующие типу переменной: `integer`, `double`, `string`, `object`, `array`, `bool` или `unknown type`, если невозможно определить тип (когда тип переменной не встраивается в PHP, а добавляется с помощью модулей, расширяющих возможности языка).

Если PHP неправильно определил тип переменной, можно указать его явно. Для этого используется функция **settype($x, $type)**, где `$type` — это одна из строк, возвращаемых функцией **gettype()**. Например:

```
settype($x,"double");
```

Функция возвращает значение false, если невозможно привести тип переменной `$x` к указанному типу `$type`.

Логические переменные и их особенности в PHP

Наверное, вы заметили, что большинство функций для определения типа возвращают логическое значение `true` или `false` (все функции, кроме **gettype()**). Сейчас самое время поговорить о логических переменных и связанных с ними «подводными камнями».

В PHP истиной являются любое не равное нулю число, любая непустая строка, значение `true`. Значение `false`, пустая строка, нуль — это ложь. Для того, чтобы понять это проведем ряд экспериментов с логическими переменными. При этом мы немного забежим вперед и будем использовать условный оператор **If**. Однако, если вы знакомы с каким-либо алгоритмическим языком (Си, Паскаль и т.п), то без труда поймете смысл оператора. Тем более, что текст примеров сопровождается подробными комментариями. В противном случае вы можете вернуться к этому пункту позднее.

Итак, создайте сценарий logical.php:

```
<?
echo false;
echo true;
?>
```

Первый оператор выводит пустую строку, то есть ничего не выводит, а второй — выведет 1. Выше было сказано, что ненулевое значение принимается, как истина.

В использовании логических переменных в PHP имеется еще одна особенность. Если в операторах сравнения (= =, !=, <,>) один тип является

логическим, то и второй также воспринимается, как логический. Модифицируем наш сценарий так:

```
<?
$x=10;
if ($x==1) echo " Переменная равна 1\n";
if ($x==true) echo "Переменная равна True\n";
?>
```

В первой строке переменной `$x` было присвоено значение 10. Затем `$x` сравнивается с 1 и если `$x` равно 1, то выводится строка «Переменная равна 1». Затем значение переменной `$x` сравнивается со значением `True`. И если `$x` равно `true`, то выводится строка «Переменная равна True». Исходя из приведенного выше утверждения, должна быть выведенной только вторая строка. Так оно и происходит.

Рассмотрим еще один пример:

```
<?
$x = 100;
$y = true;
echo "x= $x\n";
echo "y= $y\n";
if ($x==$y) echo "X=Y";
?>
```

Вот здесь уж совсем смешная ситуация: сначала программа сообщает, что X = 100, Y = 1, а затем, что X = Y.

5.3. Константы

Константы содержат постоянные значения. В отличие от переменной, вы не можете изменить значение константы, которое было ей присвоено при ее объявлении. Константы удобно использовать для хранения значений, которые не должны изменять во время работы программы, например, имя разработчика программы.

Задание констант. Функция define()

Для определения константы используется функция **define**. Эта функция имеет следующий формат:

define ($name, $value, $case_sen)

где **$name** — имя константы;
$value — значение константы;

$case_sen — необязательный параметр логического типа, указывающий следует ли учитывать регистр букв (true) или нет (false).

Пример:

```
define("pi",3.14,true);
echo pi;
```

Если параметр `$case_sen` (именно так называется третий параметр) равен true, то интерпретатор будет учитывать регистр символом при работе с именем константы. Обратите внимание, что после объявления константы ее можно использовать без предваряющего символа доллара.

При объявлении констант нужно помнить два важных момента:

- Имя константы должно быть заключено в кавычки, а не в апострофы.
- Нельзя объявлять две константы с одним и тем же именем.

Проверка существования констант. Функция defined()

Для проверки существования константы можно использовать функцию **defined()**. Данная функция возвращает `true`, если константа объявлена, и `false` — если не объявлена.

Пример:

```
define("pi",3.14,true);
if ((defined("pi")==true) echo "Константа объявлена";
```

Стандартные константы

При программировании на PHP вы можете использовать следующие стандартные константы (см. табл. 5.2).

Стандартные константы *Таблица 5.2*

Имя	Значение
__FILE__ (два знака подчеркивания до FILE и два – после)	Содержит имя программы, которая выполняется в данный момент
__LINE__	Номер строки, которую сейчас обрабатывает интерпретатор
PHP_OS	Имя и версия операционной системы, под которой запущен PHP
PHP_VERSION	Версия PHP
TRUE, true, FALSE, false	Уже знакомые нам логические константы

5.4. Операции с переменными

Мы уже рассмотрели две важнейшие операции с переменными — определение и установку типа переменной. Поэтому нам осталось рассмотреть всего три операции:

- Присвоение значения.
- Проверка существования переменной.
- Уничтожение переменной.

Вы можете спросить, а как же арифметические и другие операции? Все остальные операции специфичны для конкретного типа переменной. Например, операция умножения совершенно бессмысленна для двух строк или числа и строки.

Тем не менее, PHP является очень гибким языком и выполняет преобразования числа в строку (и наоборот) незаметно для нас. Когда я говорил, что операция умножения бессмысленна для двух строк, я был не совсем прав. Выполните этот сценарий, и вы убедитесь в этом:

```
<?
$x = "10"; // строка
$y = "2"; // и это строка
echo $x*$y;
?>
```

Сценарий, вместо сообщения об ошибке, выведет на экран число 20. Как видите, если строка допускает преобразование типа, то PHP работает со строками, как с обыкновенными числами.

Присвоение значений. Оператор присваивания

Оператор присваивания позволяет придать переменной некоторое значение. Нет такого языка программирования, в котором не было бы оператора присваивания. Поэтому я вас сильно не удивлю, если скажу, что в PHP он тоже есть:

```
$x = 4;
$y = $x;
```

Вот, что вам нужно знать об этом операторе и правилах присваивания:

- Переменной можно присвоить: какое-либо значение; значение, возвращаемое функцией; значение другой переменной; значение выражения ($x + $y * 2 − 7); ссылку на другую переменную.
- В PHP нет указателей, поэтому, если в переменную $x вы поместите файл размером 500 Кб, а потом присвоите ее переменной $y, то

будет создана точная копия переменной $x, которая тоже будет занимать 500 Кб. Итого у вас в памяти будет почти один мегабайт информации. Это нужно учитывать при создании так называемых временных переменных. Желательно после окончания работы с такой переменной освободить память, то есть уничтожить переменную.

- Интерпретатор сам выполняет преобразование типов, но иногда привести тип переменной к другому просто невозможно. Например, как можно преобразовать строку «1212ю3te» в число?
- Как я уже говорил, при присваивании создается точная копия переменной, копируется также и тип переменной. Это означает, что если вы массиву присвоите число, весь массив будет потерян.

Работа оператора присваивания состоит из двух последовательных этапов:

1. Вычисление выражения, стоящего в правой части.
2. Присвоение переменной, стоящей в левой части, значения, полученного в пункте 1.

Таким образом, из-за того, что процесс присвоения состоит из двух раздельных этапов, в левой и правой части оператора присваивания может находиться одна и та же переменная. Например: `$x = $x+4`.

Кроме того, так же как и в языке Си, в PHP существует возможность использования операции присваивания в следующем виде: **переменная операция=значение**.

Например:

```
$x +=10; // эквивалентно $x = $x + 10;
$x *=10; // эквивалентно $x = $x * 10;
```

Проверка существования переменной

Проверка существования переменной — это очень удобная возможность языка программирования. Вы можете спросить: «А зачем она нужна? Что я не помню, объявлял я переменную или нет?» Я сейчас поясню. Помните, я говорил, что переменные может объявлять сам PHP — при разборе параметров. Так вот, благодаря возможности проверки существования переменной вы можете проверить, передан ли сценарию определенный параметр или нет, а только потом начинать с ним работать. Рассмотрим два сценария.

Сценарий 1:

```
<?
$x = $m_name;
?>
```

Сценарий 2:

```
<?
if (isset($m_name))
 {
  $x = $m_name;
 }
?>
```

В первом случае мы сразу начинаем работать с переменной. Если по каким-либо причинам интерпретатор не передал сценарию параметр m_name, мы получим сообщение о том, что переменная не определена. Во втором сценарии мы с помощью функции isset($m_name) проверяем существование переменной $m_name, а только потом начинаем с ней работать. В результате ваш сценарий не прекратит свою работу из-за банальной ошибки пользователя, если тот забудет ввести свое имя.

Конечно, если проверять все переданные параметры, наша программа станет намного «толще» (особенно, если параметров много). Однако мы можем поступить проще. Предположим, что нам нужно написать анкету на PHP, содержащую имя и адрес электронной почты. Алгоритм нашего сценария будет такой:

- Если пользователь не передал параметры, значит мы выводим форму анкеты.
- Если параметры переданы (о чем свидетельствует наличие переменной кнопки submit), значит можно начать обработку параметров.

Выходит, что нам достаточно проверить только один параметр:

```
<html><head><title>Анкета</title></head><body>
<?
if(!isset($go))
 {
 // переменная go (кнопка) не установлена, выводим форму:
 echo "<form action=script.php method=post>";
/* Выводим поля анкеты, а затем кнопку go */
echo "<input type=submit name=go value=Send></form>";

 }
else {
/* Параметры введены пользователем, начинаем обработку */
}
```

Проверив существование кнопки go, мы начинаем обработку параметров. В выводимой форме я использовал метод POST. Для PHP-сценариев совершенно нет никакой разницы, каким методом им будут переданы параметры, поэтому из соображений безопасности лучше использовать метод POST (никто из-за спины не увидит, что ввел пользователь), да и выглядит он более эстетично (ничего лишнего в строке адреса не отображается).

Удаление переменных

Чтобы не засорять память, можно удалить ненужные нам переменные. Это можно делать с помощью функции **unset()**. Эта функция удаляет указанную в скобках переменную из памяти, и программа продолжает выполняться как-будто эта переменная вообще не была инициализирована.

```
<?
$a = читаем_большой_файл;
обрабатываем_файл;
unset($a); // освобождаем память
?>
```

Эта функция особенно полезна и даже необходима при обработке больших объемов данных, чтобы они не оставались в памяти компьютера и не замедляли его работу.

5.5. Выражения и операции

5.5.1. Понятие «выражение» в PHP

Выражения являются теми «кирпичиками», из которых состоят PHP-программы. Практически все, что вы пишете в программе, является выражением. При этом под выражением понимается то, что имеет значение. Можно сказать и по-другому: все, что имеет значение, является выражением. Самое простое выражение — это константа, стоящая в правой части оператора присваивания:

```
$x = 100;
```

Разберемся, что есть что. `$x` — это переменная, `=` — это оператор присваивания, а 100 — это и есть выражение. Его значение — 100.

Выражением может служить и переменная, если ей сопоставлено определенно значение. Например:

```
$x = 4;
$y = $x;
```

В первом операторе выражением является константа 4. Во втором — переменная `$x`, так как ранее ей было присвоено значение 4. Но это еще не все, `$y = $x` тоже является выражением! Ведь не вызывает сомнения то, что его значение 4. Это, в свою очередь, позволяет использовать следующие операторы: `$y = $x = 5` или `$y = ($x = 5)`.

Таким образом, в середине выражения можем использовать оператор присваивания:

```
$y = 1;
$x = 100*($y = 2) * $y;
```

Очевидно, что переменной `$x` будет присвоено значение 400 (100*2 = 200 *8 2 = 400).

5.5.2. Операции в PHP

Арифметические операции

Как и в любом другом языке, мы можем использовать арифметические операции:

- **X + Y** — сложение;
- **X − Y** — вычитание;
- **X * Y** — умножение;
- **X / Y** — деление;
- **X % Y** — остаток от деления X на Y.

Также возможно использование скобок. Приоритет одних арифметических операций над другими и изменение приоритетов при использовании скобок в арифметических выражениях соответствуют обычным математическим правилам.

К арифметическим операциям можно также отнести операции **инкремента** и **декремента**:

- операция инкремента **$x++** увеличивает значение на 1;
- операция декремента **$x--** уменьшает значение на 1.

Битовые операции

Данные операции предназначены для установки или снятия групп битов целочисленной переменной. Ведь любое число — это просто последовательность бит. Целые числа в PHP — 32-разрядные, поэтому для представления одного числа используются 32 бита:

```
0000 0000 0000 0000 0000 0000 0000 0000 - это ноль
0000 0000 0000 0000 0000 0000 0000 0001 - это 1
0000 0000 0000 0000 0000 0000 0000 0010 - это 2
0000 0000 0000 0000 0000 0000 0000 0011 - это 3
0000 0000 0000 0000 0000 0000 0000 0100 - это 4
0000 0000 0000 0000 0000 0000 0000 0101 - это 5
...
0000 0000 0000 0000 0000 0000 0000 1111 - это 15
...
```

Битовые операции:

- x & y — будут установлены те биты, которые установлены в x и y одновременно:

  `x = 1111 (15) y = 1100 (12) Результат = 1100 (12).`
- x | b — будут установлены только те биты, которые были установлены или в x или в y:

  `x = 0100 (4) y = 0101 (5) Результат = 0101 (5).`
- ~x — инвертирование битов:

  `x = 1001 Результат = 0110.`
- x << y — поразрядный сдвиг битов x влево на y разрядов:

  `x = 0010 y = 0001 (1 разряд) Результат = 0100.`
- x >> y — поразрядный сдвиг битов x вправо на y разрядов:

  `x = 0010 y = 0001 (1 разряд) Результат = 0001.`

Логические выражения

Логические выражения — это выражения, результатом которых может быть или истина, или ложь. Примеры логических выражений:

```
$x = true;          // $x = истина
$x = $y < 0;        // $x = true, если $y < 0
$x = $y == 0;       // $x = true, если $y = 0
$x = $y>=0&&$y<=10  // $x = true, если $y < =0 и $y<=10, то
                    // есть $y в диапазоне от 0 до 10.
```

В логических выражениях можно использовать операторы сравнения и логические операции. **Операторы сравнения**:

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

В PHP нельзя сравнивать массивы и объекты, разрешается сравнивать только скалярные переменные.

Логические операции:

- !x возвращает `true`, если x — ложь, и наоборот (отрицание — NOT);
- x && y возвращает `true`, если x и y — истинны (логическая операция И — AND);
- x || y возвращает `true`, если истинно хотя бы одно из значений или все они истинны (логическая операция ИЛИ — OR).

Операции операторов инкремента (++) и декремента (--) не работают с логическими переменными, то есть не являются логическими операциями. Можно бы было, конечно, предположить, что если `false` — это 0, следовательно, инкремент `false` — это `true`, то есть 1.

```
<?
$x = false;
echo $x;
$x++;
echo $x;
?>
```

Однако приведенный сценарий ничего не выведет (точнее он выведет две пустые строки). Это означает, что операция инкремента для логических значений не работает. То же самое можно сказать и об операции декремента.

Сколько будет 2+2*2? Правильно, 6. Поскольку операция умножения выполняется перед операцией сложения, следовательно, приоритет операции умножения выше. Приоритет арифметических и логических операций представлен в табл. 5.3.

Операции с более высоким уровнем приоритета выполняются в первую очередь. Если вы в чем-то сомневаетесь, используйте скобки.

Приоритет операций в PHP *Таблица 5.3*

Приоритет	Оператор	Порядок выполнения
13	(постфикс)++ (постфикс)--	слева направо
12	++(префикс) --(префикс)	справа налево
11	* / %	слева направо
10	+ -	
9	<< >>	
8	< <= > >=	
7	== !=	
6	&	
5	^	
4	|	
3	&&	
2	||	
1	= += -= *= /= %= >>= <<== &= ^= |=	справа налево

5.5.3. Строки и строковые выражения

Строки в апострофах и строки в кавычках

Рассмотрим небольшой пример:

```
$s = "Hello";
echo "$s";
echo '$s';
```

Оба оператора **echo** выведут строки. Первый оператор **echo** выведет строку `Hello`, а второй — `$s`. Между строками в кавычках и в апострофах существует большая разница. Если строка заключена в апострофы, то все символы трактуются как есть. Исключения составляют последовательности \' и \\. Первая представляет собой обыкновенный апостроф, а вторая — обратную наклонную черту.

Строки в кавычках позволяют, кроме всего прочего, вывести значение переменной, поэтому никогда не используйте строку в апострофах для этой цели. Предположим, что у нас есть переменная `$a = 10`. Проанализируем вывод двух операторов **echo**:

```
echo "$a"; // строка в кавычках
echo '$a'; // строка в апострофах
```

Первый оператор выведет 10 — значение переменной `$a`. Второй напечатает то, что заключено между апострофами — `$a`. Улавливаете разницу?

Кроме того, строки в кавычках могут содержать следующие специальные символы:

- `\n` — символ новой строки;
- `\r` — символы возврата каретки;
- `\t` — символ табуляции;
- `\$` — знак доллара;
- `\"` — кавычка;
- `\\` — обратный слэш;
- `\xNN` — символ с кодом NN (в шестнадцатеричной системе).

Строковые операции

Строковых операций в PHP всего две:

- s1.s2 — конкатенация (слияние) двух строк;
- s1[n] — обращение к символу в строке.

Все остальные действия над строками выполняются стандартными функциями.

Присвоить значение строке очень просто:

```
s = "hello";
s = "Это большая
многострочная строковая
константа";
```

Многострочные строковые переменные

Начиная с четвертой версии PHP, появились так называемые Here-документы. Это еще один способ записи многострочных констант. Длина строки в PHP ограничена только объемом свободной оперативной памяти, поэтому в переменную можно записать даже небольшой файл. Записать большое текстовое сообщение можно так:

```
$s =<<<LABEL
Текст
...
LABEL;
```

При использовании этого способа записи многострочных строковых переменных, нужно помнить три правила:

- LABEL может быть любым идентификатором.
- После LABEL и до конца строки не должно быть непробельных символов.
- После второго (завершающего) LABEL должна стоять точка с запятой.

Предположим, что нам нужно написать небольшую программу, выводящую строку «Hello world». Нет ничего проще? А я так не думаю:

```
<?
$a = "Hell";
echo "$ao world";
?>
```

В этом случае интерпретатор сообщит, что переменная `$ao` не найдена. Как выйти из этой ситуации? Можно использовать операцию слияния двух строк:

```
echo $a."o world";
```

А можно взять имя переменной в фигурные скобки:

```
echo "${a}o world"
```

или

```
echo "{$a}o world"
```

Однако лучше использовать первый вариант, поскольку иногда фигурные скобки могут трактоваться как спецсимволы.

Строки в обратных кавычках (вызов внешней программы)

Вы хотите запустить внешнюю программу и присвоить переменной результат ее работы? Нет ничего проще! Для этой цели используйте строки в обратных кавычках (они находятся под тильдой):

```
$directory = 'dir';
echo "<h1>Содержимое каталога:</h1><BR><PRE> $directory";
```

5.5.4. Оператор эквивалентности (===)

Иногда обычный оператор сравнения == работает не совсем корректно, точнее не так, как нам хочется, в некоторых ситуациях даже получаются мини-анекдоты. Рассмотрим следующий пример:

```
<?
$x = 0; // число, равно 0
$s = ""; //пустая строка
echo gettype($x)."\n";
echo gettype($s)."\n";
if ($x==$s) echo "x =s";
?>
```

Переменные `$x` и `$s` не могут быть равны даже с логической точки зрения, программа сначала выводит сообщение о том, что переменная `$x` является целым числом (integer), а переменная `$s` — строкой (string). А после этого программа сообщает, что переменные равны. Все дело том, что PHP трактует одну из переменных (какую именно — не знаю, поскольку и 0, и " " — это `false`, вполне может быть, что интерпретатор считает логическими обе переменные), как логическую переменную, а если хотя бы одна переменная логическая, то вторая тоже считается логической. Поэтому мы получаем `false==false`, следовательно, программа выведет строку `"x=s"`.

Чтобы избежать подобной ситуации, нужно использовать оператор эквивалентности ===. Если модифицировать нашу программу так:

```
<?
$x = 0;// число, равно 0
$s = ""; //пустая строка
echo gettype($x)."\n";
echo gettype($s)."\n";
if ($x===$s) echo "x =s";
?>
```

то сейчас строка "x=s" выведена не будет. Проведем еще один эксперимент; *с использованием массивов. Понимаю, что о массивах в PHP вы еще ни*чего не знаете, но сейчас просто необходимо забежать немного наперед.

```
<?
$A = array('1'=>'0001'); // используются апострофы, а не кавычки
$B = array('2'=>'0010');

if ($A == $B) echo "A= =B";
if ($A === $B) echo "A= = =B";
?>
```

Будет выведена первая строка, но не вторая. В версиях PHP 4.0.x и ниже массивы $A и $B преобразовывались в строки "array", которые и сравнивались. Вот из-за этого оператор == неправильно работал. Оператор === с самого начала лишен этого недостатка. Сейчас у меня установлена версия 4.1.2 (не самая новая), но в ней оператор == работает корректно.

Для оператора === должен быть обратный оператор !===, но его нет. Если вы напишете:

```
if ($A !=== $B) echo "A<>B";
```

вы получите сообщение об ошибке. Но можно обойти это неудобство так:

```
if (!($A === $B)) echo "A<>B";
```

5.6. Ссылки

Как уже отмечалось ранее, в PHP нет такого понятия, как указатель. Хорошо это или плохо — сложно сказать. То, что является достоинством C — указатели — одновременно является его недостатком — уж слишком там все запутано, ошибки, связанные с указателями, очень трудно найти. Лично мне больше нравится идеология указателей в языке Pascal: вы можете нормально работать, как с указателями, так и без них. В любом случае, большее, на что нам приходится рассчитывать в PHP — это ссылочные переменные или просто ссылки. Ссылки бывают двух типов: символические и жесткие. Последний тип ссылок появился в четвертой версии интерпретатора.

Жесткие ссылки

Жесткая ссылка — это просто псевдоним имени другой переменной. Ссылки на ссылку на переменную (многоуровневые ссылки), как в языке Perl, не поддерживаются. Для создания жесткой ссылки используется оператор **&**:

```
<?
$x = 77;
$link = &$x;

$link = 66;
echo $link;
?>
```

Нетрудно догадаться, что выведет программа — 66. Для разрыва связи между переменной и ссылкой используется функция **UnSet**:

```
UnSet($link);
```

Саму переменную нельзя удалить до тех пор, пока на нее ссылается хотя бы одна ссылка.

Символические ссылки

Символическая ссылка — это обыкновенная переменная, содержащая имя другой переменной. Для доступа к значению ссылки используется двойной знак доллара **$$**:

```
<?
$x = 77;
$link = "x";        // символическая ссылка на переменную $x
echo $$link;        // выводит 77
echo $link;         // выводит x
$$link = 66;        // аналогично $x = 66
echo $x;            // выведет 66
?>
```

Я не рекомендую использовать символические ссылки, их нужно использовать в самых крайних случаях, которых будет немного. Использование символических ссылок может и без того запутать запутанную программу. Лучше использовать жесткие ссылки: хотя бы исходя из того, что для них нужен один оператор $. А вообще старайтесь не использовать ссылки — так ваша жизнь станет чуть проще.

5.7. Основные конструкции языка

Конструкции языка PHP очень похожи на конструкции языка C, поэтому, если вы знаете язык C, очень быстро освоите PHP. Нужно заметить, что синтаксис PHP очень лаконичен, а мы знаем, что «краткость — сестра таланта». PHP еще раз позволяет нам убедиться в этом.

5.7.1. Условный оператор

Проблему выбора можно без зазрений совести отнести к глобальным проблемам. Как часто вам приходится выбирать? Так почему бы не заставить компьютер сделать выбор за нас? Для этого и предназначен условный оператор.

Конструкция **if-else** (условный оператор) является, наверное, самой распространенной конструкцией в программах. Какая же серьезная программа не использует эту конструкцию?

Синтаксис конструкции if-else таков:

```
if (логическое выражение)
 оператор_1;
else
 оператор_2;
```

Первый оператор будет выполнен, если логическое выражение истинно, второй — если логическое выражение ложно. Можно также использовать сокращенную форму конструкции, без второго оператора:

```
if (логическое выражение)
 оператор_1;
```

В этом случае мы можем обойтись без «else». Если вам нужно выполнить не один оператор, а целую группу, тогда эти операторы следует поместить в блок операторов { }:

```
if (логическое выражение)
 {
 оператор_1;
 ...
 оператор_i;
 }
else
 {
  оператор_i+1;
  ...
  оператор_n;
 }
```

Кажется, в этой книги я еще ни разу не обращал внимания на разделитель операторов — это символ точка с запятой ";". Совсем очевидно, что мы можем записать несколько операторов в одну строку:

```
if (логическое выражение)  {  оператор_1;  ...  оператор_i;  }
```

В данном случае после фигурной скобки можно ставить точку с запятой, а можно и не ставить. Но если вы думаете использовать полную конструкцию (то есть с «else»), тогда точка с запятой не ставится. Если же вместо блока операторов используется только один оператор (см. общий синтаксис), тогда точка с запятой после него (перед else) обязательна:

```
if ($a>5) echo "a>5";
else echo "a<=5";
```

Существует еще один вариант конструкции **if-else**. Вот его синтаксис:

```
if (логическое выражение 1)
 оператор_1;
elseif ( логическое выражение 2 )
 оператор_2;
else
 оператор_3;
```

По-моему данная конструкция не очень удобна, поэтому я советую использовать ее классический вариант.

Для полноты описания данной конструкции нужно сказать, что вы можете использовать еще один — третий альтернативный вариант (на этот раз уже точно последний) конструкции:

```
if (логическое выражение 1):
 оператор_1;
elseif ( логическое выражение 2 ):
 оператор_2;
else:
 оператор_3;
endif
```

Если пропустить точки с запятой, то вы получите сообщение об ошибке.

5.7.2. Циклы

На втором месте по частоте использования, после условного оператора, находятся циклы. Цикл позволяет повторить определенное (и даже не определенное — когда работа цикла зависит от условия) количество раз какие-нибудь операторы. Данные операторы называются телом цикла — они будут «крутиться» в цикле. Проход цикла называется итерацией.

Как и C, PHP поддерживает три вида циклов:

- Цикл с предусловием (while).
- Цикл с постусловием (do-while).
- Цикл со счетчиком (for).

Цикл с предусловием While

Принцип работы цикла с предусловием:

- Вычисляется значение логического выражения.
- Если значение истинно, выполняется тело цикла, в противном случае — переходим на следующий за циклом оператор.

Синтаксис цикла с предусловием выглядит следующим образом:

```
while ( логическое выражение )
 инструкция;
```

В данном случае телом цикла является «инструкция». Обычно тело цикла состоит из большего числа операторов. Простейший пример цикла:

```
<?
$i=0;
while($i++ < 10) echo $i;
?>
```

Данная программа выведет строку 12345678910. Обратите внимание на последовательность выполнения операций условия $i++ < 10. Сначала проверяется условие, а только потом увеличивается значение переменной. Если же мы поставим операцию инкремента перед переменной (++$i<10), то сначала будет выполнено увеличение переменной, а только потом — сравнение. В результате мы получили бы строку 123456789.

Этот же цикл можно было бы записать по-другому:

```
$i = 0;
while ($i<10)
 {
 $i ++;// увеличение счетчика
 echo $i;
 }
```

Если мы увеличим счетчик после выполнения оператора **echo**, мы получим строку 0123456789. Но в любом случае, у нас будет 10 итераций. Итерация — это проход цикла, то есть выполнение инструкций тела цикла.

Цикл с постусловием do while

Цикл с постусловием отличается от цикла с предусловием тем, что сначала выполняется тело цикла, а только потом уж проверяется условие. Таким образом, тело цикла хотя бы один раз, но будет обязательно выполнено.

Синтаксис цикла do while такой:

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

Попробуем с помощью цикла с постусловием получить строку 12345678910. Поскольку в этом цикле сначала выполняется тело, а по-

том проверяется условие, то нам нужно установить начальное значение счетчика в единицу:

```
<?
$i = 1;
do echo $i; while ($i++ <10);
?>
```

Напомню, что мы можем увеличивать значение счетчика в теле цикла (так даже понятнее).

Цикл со счетчиком For

Цикл со счетчиком используется для выполнения тела цикла определенное количество раз.

Например, в цикле **while** удобно ждать определенного события, например, когда возвращаемое какой-то функцией значение будет истинно (а такая ситуация может и не произойти). Цикл **for** удобно использовать, например, для заполнения массива. Синтаксис цикла **for**:

```
for (команды_инициализации; условие; команды_после_итерации) { тело цикла }
```

Оператор **for** начинает свою работу с выполнения команд инициализации. Данные команды выполняются всего лишь один раз. После этого проверяется условие: если оно истинно, выполняется тело цикла. После того, как будет выполнен последний оператор тела, выполняются команды «после итерации». Затем снова проверяется условие, в случае, если оно истинно, выполняется тело цикла и поститерационные команды, и т.д.

Выведем привычную нам строку:

```
<?
for ($i=0; $i<10; $i++) echo $i;
?>
```

Данный сценарий выведет строку 0123456789. Нам же нужна строка 12345678910, поэтому нужно обеспечить увеличение счетчика при проверке логического выражения:

```
<?
for ($i=0; $i++<10;) echo $i;
?>
```

В этом случае нам не нужны команды, выполняющиеся после итерации. Если вам нужно указать несколько команд, вы можете разделять их запятыми:

```
for ($i=0, $j=0; $<10; $i++, $j++) echo $i;
```

Как вы уже успели заметить, цикл со счетчиком является вариацией цикла с предусловием, так как сначала в нем проверяется условие, а только потом выполняется тело.

Альтернативный синтаксис

Циклы **while** и **for** имеют альтернативный синтаксис. Для цикла **while** он таков:

```
while (логическое_выражение):
 тело цикла
endwhile;
```

А для цикла **for** — такой:

```
for (команды_инициализации; условие; команды_после_итерации):
   тело цикла
endfor;
```

Обратите внимание на двоеточие — его наличие обязательно! Альтернативный синтаксис также предусмотрен для оператора **if-else**:

```
if (логическое_выражение):
 команды;
elseif(логическое_выражение2):
    команды;
else:
   команды;
endif;
```

Использовать данный синтаксис или нет — ваше личное дело. Что же касается меня, то я предпочитаю использовать «классику» — обыкновенный синтаксис.

Использовать альтернативный синтаксис имеет смысл, если вы предпочитаете включать в PHP-сценарии HTML-код, а не выводить его с помощью **echo**. В этом случае альтернативный синтаксис облегчает использование операторов **if-else**, **while** и **for**:

```
<?if ($REMOTE_ADDR):?>
   <h1>Welcome home!</h1>
<?else:?>
   <h1>Hello</h1>
<?endif; ?>
```

Вложенные циклы

Каждый цикл может содержать в своем теле несколько циклов. Каждый из этих циклов называется вложенным циклом. Вкладываться может неограниченное число циклов, причем все они могут быть разного типа — **while**, **do** или **for**. Рассмотрим следующий код:

```
for ($I=0; $I<10; $I++) {
   echo "$I\n";
   for ($J=0; $J<5; $J++) echo "$J ";
   echo "\n";
}
```

Вложенный цикл выводит строку 0 1 2 3 4. Данная строка будет выведена 10 раз — это число итераций внешнего цикла for.

Операторы break и continue

При использовании циклом можно использовать операторы **break** и **continue**. Первый прерывает работу всего цикла, а второй — только текущей итерации. Рассмотрим следующие примеры:

```
<?
$i = 0;
while ($i++ < 10)
 {
 if ($i==3) break;
 echo "Итерация $i\n";
 }

?>
```

В этом примере после второй итерации работа цикла будет прервана и программа выведет строки:

```
Итерация 1
Итерация 2
```

Если же вместо оператора **break** использовать оператор **continue**, будет прервана работа только третьей итерации и мы получим вывод:

```
Итерация 1
Итерация 2
Итерация 4
Итерация 5
Итерация 6
Итерация 7
Итерация 8
Итерация 9
Итерация 10
```

Предположим, что у нас есть вложенные циклы, например,

```
for (...)
 {
     for (...)
     {
        for (...)
     }
 }
```

Нам нужно прервать работу определенного цикла, например, первого. Тогда как оператор **break** находится в теле третьего. В этом случае мож-

но передать функции **break** параметр — номер цикла, например, break(1). Нумерация циклов начинается с единицы.

Цикл перебора массивов foreach

В четвертой версии PHP появился еще один вид цикла — **foreach**. Данный цикл предназначен специально для перебора массивов.

Синтаксис данного цикла таков:

```
foreach(массив as $ключ=>$значение)
операторы;
```

Операторы, содержащиеся в теле цикла, будут выполнены для каждого элемента массива. Переменная `$ключ` будет содержать имя ключа элемента, а переменная `$значение` — значение элемента.

Следующий пример отобразит значение массива $GLOBALS, содержащего глобальные переменные (будут выведены как имена переменных, так и их значения):

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

?>
```

Вывод программы выглядит примерно, как на рис. 5.2.

Рис. 5.2. *Вывод массива $GLOBALS*

Примечание.

Массив — это упорядоченный набор данных. Каждый элемент массива имеет индекс или ключ. Ключ служит для однозначной идентификации элемента массива. В одном массиве не может быть двух элементов с одинаковыми ключами.

В PHP есть один очень полезный стандартный массив $GLOBALS. Ключом данного массива является имя переменной, а значением элемента массива — значение переменной. Об этом важном массиве мы еще не раз поговорим в этой книге.

5.7.3. Инструкция switch-case (конструкция выбора)

Нами осталась незамеченной очень важная конструкция — **switch-case**. Данная конструкция предназначена для выбора действий, в зависимости от значения указанного выражения. Конструкция **switch-case** чем-то напоминает оператор **if-else**, который, по сути, является ее аналогом. Конструкцию выбора нужно использовать, если предполагаемых вариантов много, скажем, больше 5, и для каждого варианта нужно выполнить специфические действия. В этом случае использовать оператор выбора **if-else** просто неудобно.

Синтаксис **switch-case** таков:

```
switch (выражение)
{
case значение1 :  команды_1; [break;]
...
case значениеN :  команды_N; [break;]
[default: команды_по_умолчанию; [break;]]
}
```

Альтернативный синтаксис конструкции выглядит так:

```
switch (выражение):
   case значение1: команды1; [break;]
   …
   case значениеN :  команды_N; [break;]
   [default: команды_по_умолчанию; [break;]]

endswitch;
```

Работает эта конструкция следующим образом:

1. Вычисляется значение выражения.
2. Просматривается набор значений. Пусть значение_i равно значению выражения, вычисленного на первом шаге. Если не указан оператор **break**, то будут выполнены команды i, i+1, i+2, ... ,N. В противном случае (есть **break**) будет выполнена только команда с номером i.

3. Если ни одно значение из набора не совпало со значением выражения, тогда выполняется блок **default**, если он указан.

Небольшой пример:

```
<?
$age=21;
switch ($age)
{
case 20 : echo "Вам двадцать лет\n";
case 21 : echo "Вам двадцать один год\n";
case 22 : echo "Вам двадцать два года\n";
default: echo "Вам $age лет\n";
}
?>
```

Если переменная `$age` равна 21, тогда будут выведены строки:

```
Вам двадцать один год
Вам двадцать два года
Вам 21 лет
```

Ясное дело, что нам нужно, чтобы программа вывела только одну строку, поэтому нужно использовать операторы **break**:

```
<?
$age=21;
switch ($age)
{
case 20 : echo "Вам двадцать лет\n"; break;
case 21 : echo "Вам двадцать один год\n"; break;
case 22 : echo "Вам двадцать два года\n"; break;
default: echo "Вам $age лет\n"; break;
}
?>
```

Если кто-то знает язык Pascal, то в нем была подобная конструкция **case**, позволяющая указывать диапазоны значений:

```
case (age)
10..20 : writeln('Вам от 10 до 20 лет');
...
end;
```

В PHP такую возможность можно реализовать так (диапазон 20...22):

```
<?
$age=21;
switch ($age)
```

```
{
case 20 :
case 21 :
case 22 : echo "Вам от 20 до 22\n"; break;
}
?>
```

5.8. Собираем один скрипт из нескольких отдельных скриптов

5.8.1. Инструкции require

Данные инструкции позволяют собрать большую программу на несколько отдельных файлов. Инструкция **require** позволяет включить код до выполнения нашего сценария.

Общий синтаксис инструкции такой:

```
require имя_файла;
```

Теперь разберемся подробнее, что же делает **require**. Интерпретатор анализирует сценарий и, найдя **require**, просто заменяет ее содержимым указанного файла. Это может быть HTML-файл или сценарий на PHP. В последнем случае содержимое файла должно быть обрамлено тегами <? и ?> — как в случае с обыкновенным сценарием. Вставка содержимого файла, происходит ДО выполнения сценария — прочитав следующий пункт, вы заметите разницу.

Очень удобно использовать эту инструкцию для включения HTML-заголовков. Например:

```
Файл header.html:
<html><head>
<title>My Company's Official Web Page</title>
</head>
<body>
```

Сценарий:

```
require header.html
/* тело документа */
```

5.8.2. Инструкции include

В отличие от **require** директива **include** позволяет включить код в сценарий во время выполнения сценария. Использование этой директивы выглядит следующим образом:

```
include имя_файла;
```

Возможно, вам нужно динамически менять заголовок в зависимости от ситуации:

```
for ($i=0; $i<4; $i++) {
inlcude "header{$i}.html"
};
```

Создайте четыре файла: `header0.html`, `header1.html`, `header2.html` и `header3.html`. Пусть каждый из них содержит соответствующую имени файла цифру — от 0 до 3. При выполнении сценария вы увидите вывод 0123. Данный вывод означает, что содержимое файлов вставляется в сценарий во время его выполнения. Замените **include** на **require**, и вы почувствуете разницу.

Обратите внимание на фигурные скобки. При таком контексте использования **include** они обязательны. Почему? Интерпретатор просто заменяет **include** содержимым файла. Представьте, что наш файл содержит не один, а несколько операторов. Получится следующее:

```
for ($i=0; $i<4; $i++)
 operator1;
 operator2;
 operator3;
```

Как вы видите, мы получаем совсем не то, на что рассчитывали. Мы хотели, чтобы все три оператора выполнились четыре раза, а на самом деле, четыре раза выполняется только первый оператор, а остальные два будут выполнены после окончания работы цикла. Отсюда следует небольшой вывод: если вы забудете заключить **include** в фигурные скобки (в цикле), то ваша программа будет работать некорректно или вообще возникнет ошибка. Всегда заключайте инструкцию **include** в фигурные скобки!

5.8.3. Особенности использования include и require

Что же происходит на самом деле при использовании **include**? Мы знаем, что PHP преобразует наш сценарий в собственное внутреннее представление. Первую строку, вторую, третью... И так, пока он не дойдет до инструкции **include**. Затем PHP прекращает транслировать наш сценарий и переключается на указанный в **include** файл. Вполне объяснимо: интерпретатор не знает, что будет в этом файле, поэтому не может отложить его трансляцию «до лучших времен». Если у вас большая программа, и она подключает дополнительные файлы с помощью **include**, вряд ли она будет работать быстро, поскольку из-за особенностей **include** быстродействие резко снижается.

С **require** таких проблем нет: файл включается до выполнения сценария, то есть на момент трансляции он уже включен в наш сценарий и PHP нет необходимости ждать, пока оттранслируется указанный файл.

Что же использовать? Если вам нужно подключить какой-то дополнительный модуль (например, класс **HtmlMimeMail**, который будет рассматриваться в этой книге), используйте **require**. Если вы точно знаете, что вам нужно включить какой-то файл не один раз, используйте **include**, однако старайтесь по возможности использовать **require** везде, где только можно.

5.8.4. Инструкции однократного включения include_once и require_once

Ваша программа разрослась до такой степени, что вы не можете вспомнить, какой файл вы включали, а какой — нет? Такого быть не может! Неужели не видно, что и где включается?

Такая ситуация встречается довольно часто при разработке действительно больших сценариев. Предположим, что у нас есть сценарий `main.php`, в который включены файлы `header.php`, `code.php`, `footer.php`. В файл `code.php` тоже включается несколько файлов. В одном из этих файлов находится инструкция:

```
require "header.php";
```

Получается, что файл `header.php` включен дважды. Что ж бывает — вы забыли или кто-то из ваших коллег даже не подозревал, что файл `header.php` уже включен. Такое включение может стать причиной серьезной ошибки, которую очень трудно найти.

Разработчики PHP предусмотрели выход из этой ситуации. Используя инструкции однократного включения **include_once** и **require_once**, вы можете быть уверены: ни один файл не будет включен дважды. Работают эти инструкции точно так же, как и **include** и **require**, но с одним отличием: перед включением файла интерпретатор проверяет, включался ли этот файл или нет. Если да, то файл не будет включен в состав сценария. Конечно, использование этих инструкций требует дополнительных усилий от вашего интерпретатора, поскольку он должен помнить имена всех включенных файлов, но это лучше, чем помнить их самому.

5.9. Обработка ошибок

Интерпретатор PHP позволяет программисту определить, какие сообщения об ошибках нужно выводить, а какие — нет. Пока вы отлаживаете программу, я рекомендую выводить все сообщения об ошибках и все предупреждения, а потом, когда программа нормально работает, выводить только сообщения об ошибках.

Параметр **error_reporting** в файле конфигурации PHP — `/etc/php.ini` — позволяет определить степень обработки и протоколирования ошибок. Возможные значения:

E_ALL — все ошибки и предупреждения;

E_ERROR — фатальные ошибки во время выполнения программы (например, невозможно открыть файл);

E_WARNING — предупрежения времени выполнения (нефатальные ошибки);

E_PARSE — синтаксические ошибки;

E_NOTICE — замечания времени выполнения (например, сообщение о том, что переменная неинициализирована).

Кроме вышеперечисленных возможны и другие значения, но они не столь важны. Я рекомендую включить оповещения обо всех ошибках и предупреждениях:

```
error_reporting = E_ALL
```

Когда программа отлажена, можно отключить замечания времени выполнения. Для отключения какого-нибудь вида ошибок используется тильда:

```
error_reporting = E_ALL & ~E_NOTICE
```

Если вы не можете отредактировать файл `php.ini` (например, у вас нет на это прав), можно использовать вызов функции **error_reporting()**, например,

```
error_reporting(E_ALL);
```

Следует учитывать, что действие директивы **error_reporting** в файле конфигурации распространяется на все ваши сценарии, а вызов функции **error_reporting()** — только на текущий сценарий.

Существует еще один способ избежать вывода сообщения об ошибке — это использование оператора @. Если этот оператор поставить перед каким-либо выражением, то все ошибки, которые в этом выражении возникнут, будут проигнорированы.

Например, если вы не хотите, чтобы пользователь видел сообщение об ошибке какой-нибудь стандартной функции, нужно указать оператор @ перед именем функции, например:

```
@mysql_query("create table tbl (no int, info text)");
```

Глава 6

Передача параметров сценариям

6.1. Формы в HTML-документах

6.1.1. Общие положения

В этом небольшом разделе мы рассмотрим создание HTML-форм. Возможно вы уже изучали этот материал, когда осваивали HTML. В этом случае вы можете сразу перейти к п. 6.2, в котором мы непосредственно будем рассматривать передачу параметров с помощью форм. Однако при изучении языка HTML формы обычно изучаются поверхностно, так как без знания CGI они ни к чему. Поэтому я все-таки рекомендую восстановить в памяти механизм создания форм. Чтобы вам не обращаться к другим книгам, я специально выделил небольшой раздел, посвященный данному вопросу.

Итак, формы предназначены для отправки информации CGI-приложению, то есть нашему сценарию. Рассмотрим простую форму ввода имени:

```
<form action="http://localhost/1.php" method="GET">
   Имя: <input type=text name=user_name><br>
   <input type=submit value="Передать">
</form>
```

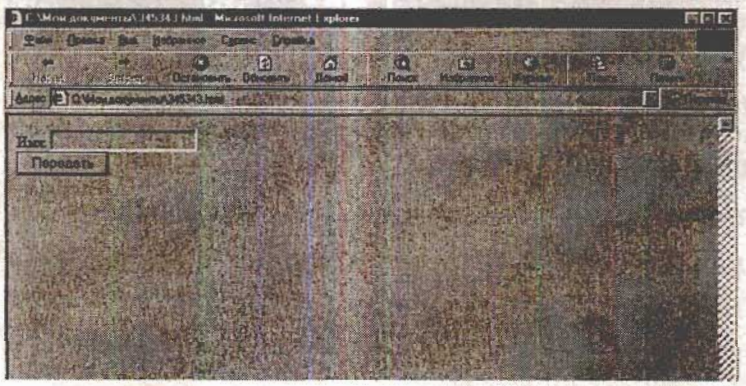


Рис. 6.1. Форма

Данная форма содержит всего два элемента — поле для ввода имени и кнопку с надписью «Передать». Что же происходит, когда пользователь вводит имя и нажимает на кнопку «Передать»? Браузер передаст сценарию `1.php` параметр `user_name`. В качестве значения параметра будет введенное пользователем имя. Так как используется метод GET, то в строке адреса вы увидите:

```
http://localhost/1.php?user_name=Denis
```

6.1.2. Элементы форм

Форма в HTML-документе реализуется тегом-контейнером FORM, в котором задаются все управляющие элементы — поля ввода, кнопки и т.д. Если управляющие элементы указаны вне содержимого тега FORM, то они не создают форму, а используются для построения пользовательского интерфейса на WEB-странице, то есть для привнесения в нее различных кнопок, флажков, полей ввода. Обработка таких элементов производится индивидуально в рамках самого HTML-документа с помощью включенных в него скриптов. А могут вообще никак не обрабатываться. Например, управляющий элемент (тег)TEXTAREA часто используется для создания окна с полосой прокрутки внутри документа для вывода большого текста, который играет второстепенную роль. Обычно так отображаются тексты лицензионных соглашений, тексты больших комментариев или правила пользования данным WEB-ресурсом.

Имена элементам формы присваиваются через их атрибут NAME.

Каждый элемент формы имеет начальное, используемое по умолчанию, и конечное значения, которые являются символьными строками. Начальные значения элементов не меняются, благодаря чему может осуществляться сброс значений, указанных пользователем. Результатом этого действия будет установка всех управляющих элементов формы в своих первоначальных используемых по умолчанию значениях.

В HTML 4.01 определены следующие типы управляющих элементов:

- **Кнопки** — задаются с помощью элементов BUTTON и INPUT. Различают:
  - кнопки отправки — при нажатии на них осуществляют отправку формы серверу;
  - кнопки сброса — при их нажатии устанавливают управляющие элементы в первоначальные значения;
  - прочие кнопки — кнопки, для которых не указано действие, выполняемое по умолчанию при их нажатии.
- **Зависимые переключатели (переключатели с зависимой фиксацией)** — задаются элементом INPUT и представляют собой переключатели

«вкл\выкл». Если несколько зависимых переключателей имеют одинаковые имена, то они являются взаимоисключающими. Это значит, что если одна из них ставится в положение «вкл», то все остальные автоматически — в положение «выкл». Именно это и является преимуществом их использования.

- **Независимые переключатели (переключатели с независимой фиксацией)** — задаются элементом INPUT и представляют собой переключатели «вкл\выкл», но в отличие от зависимых, независимые переключатели могут принимать и изменять свое значение независимо от остальных переключателей. Даже если последние имеют такое же имя.
- **Меню** — реализуется с помощью элементов SELECT, OPTGROUP и OPTION. Меню предоставляют пользователю список возможных вариантов выбора.
- **Ввод текста** — реализуется элементами INPUT, если вводится одна строка, и элементами TEXTAREA — если несколько строк. В обоих случаях введенный текст становится текущим значением управляющего элемента.
- **Выбор файлов** — позволяет вместе с формой отправлять выбранные файлы, реализуется HTML-элементом INPUT.
- **Скрытые управляющие элементы** — создаются управляющим элементом INPUT.

Как видите, очень много элементов задаются с помощью одного и того же тега INPUT. Этот тег действительно является универсальным.

6.1.3. Тег FORM – контейнер форм

Как уже говорилось, форма в HTML-документе реализуется тегом-контейнером FORM. Этот тег своими атрибутами указывает адрес сценария, которому будет послана форма, способ пересылки и характеристику данных, содержащихся в форме. Начальный и конечный теги FORM задают границы формы, поэтому их указание является обязательным. Пример приводить еще раз не буду, он был приведен в п. 6.1.1. Однако атрибуты тега FORM подробно рассмотреть все-таки следует:

- **action** — единственный обязательный атрибут. В качестве его значения указывается URL-адрес запрашиваемой CGI-программы, которая будет обрабатывать данные, содержащиеся в форме. Допустимо использовать запись MAILTO:URL, благодаря которой форма будет послана по электронной почте. Если атрибут ACTION все-таки не указан, то содержимое формы будет отправлено на URL-адрес, с которого загружалась данная WEB-страница.

- **method** — определяет метод HTTP, используемый для пересылки данных формы от браузера к серверу. Атрибут METHOD может принимать два значения: get или post.
- **enctype** — необязательный атрибут. Указывает тип содержимого формы, используемый для определения формата кодирования при ее пересылке. В HTML определены два возможных значения для атрибута ENCTYPE:
  - APPLICATION/X-WWW-FORM-URLENCODED (используется по умолчанию).
  - MULTIPART/FORM-DATA.

6.1.4. Тег INPUT и способы его использования

Элемент INPUT является наиболее употребительным тегом, т.к. с его помощью реализуются основные функции формы. Он позволяет создавать внутри формы поля ввода строки текста, имени файла, пароля и др.

Обратите внимание на особенность INPUT — у него нет конечного тега. Атрибуты и особенности использования INPUT зависят от способа его использования. Эти способы мы сейчас и рассмотрим.

Однострочные поля ввода

Наиболее часто используются поля ввода — ведь даже кнопка является полем ввода. Мы уже знакомы с полем ввода текстовой информации, поэтому обзор полей ввода начнем именно с текстового поля. Формат тега INPUT для создания поля ввода текстовой строки:

```
<input type=text
   name=имя_параметра
   [value=значение]
   [size=размер]
   [maxlen=длина]
>
```

Данный тег создает поле ввода с максимально допустимой длиной текста **maxlen** и размером в **size** знакомест. Если задан атрибут **value**, то в поле будет изначально отображаться указанная строка. В квадратных скобках помечены необязательные атрибуты. Как выглядит рассматриваемое поле, можно посмотреть на рис. 6.1.

Поле ввода пароля

Возможно, вы создаете какой-нибудь серьезный проект, требующий авторизации, то есть проверки имени пользователя и пароля. Конечно, имя пользователя можно ввести с помощью обыкновенного текстового поля.

А вот как быть с паролем? Ведь он не должен отображаться на экране? В этом случае вам поможет поле ввода пароля:

```
<input type=password
    name=имя_параметра
    [value=значение]
    [size=размер]
    [maxlen=длина]
>
```

Принцип работы данного тега точно такой же, как и текстового. Разница заключается в том, что вводимая информация в поле не отображается, а заменяется «звездочками».

Следует иметь в виду, что не рекомендуется устанавливать значение **value** (значение по умолчанию) из соображений безопасности. Да, в окне браузера данное значение не отображается, но стоит просмотреть исходный HTML-код, пароль будет виден «невооруженным глазом».

Скрытое текстовое поле

Для передачи служебной информации (о которой пользователь даже не должен подозревать) используются скрытые поля. С помощью таких полей, например, могут передаваться параметры настройки.

```
<input type=hidden
   name=имя
   value=значение
>
```

Такие поля передаются серверу, но на Web-странице не отображаются.

Независимые переключатели

Очень часто пользователю, заполняющему форму у себя в браузере, необходимо дать возможность указать свои настройки с помощью выбора определенных значений. При этом приводятся сами эти значения, а рядом с ними помещается небольшое квадратное поле, в котором можно установить, либо убрать галочку. При этом значение, соответственно, будет либо выбрано, либо нет.

Реализовать это можно опять же с помощью тега INPUT. Для этого только необходимо в качестве значения атрибута **type** указать **checkbox**:

```
<input type=checkbox
   name=имя
   value=значение
   [checked]
>
```

Если переключатель был включен на момент нажатия кнопки отправки данных, то сценарию будет передан параметр **имя=значение**. Если же флажок выключен, то сценарию вообще ничего не будет передано — как будто нашего переключателя вообще нет.

Переключатель может быть по умолчанию либо включен, либо выключен. Чтобы переключатель был по умолчанию включен, необходимо для него указать атрибут **checked**.

Переключатель **checkbox** называется независимым, так как его состояние не зависит от состояния других переключателей **checkbox**. Таким образом, в одной форме может быть одновременно выбрано несколько переключателей (рис. 6.2).

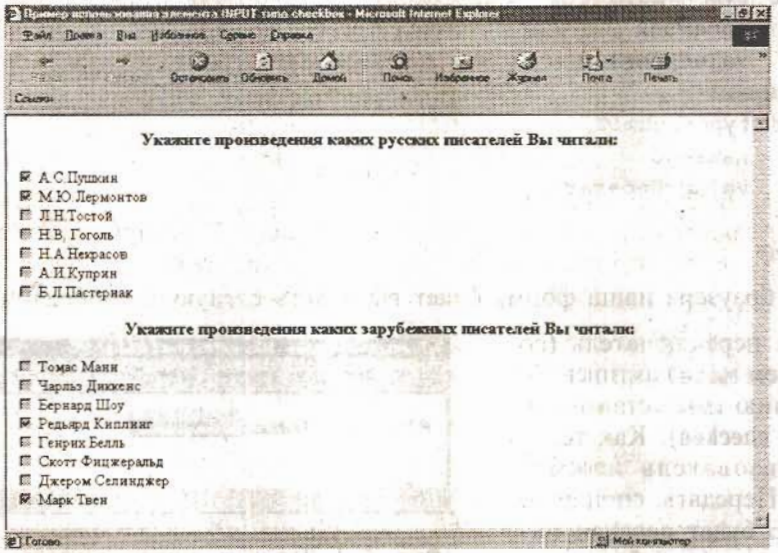


Рис. 6.2. *Использование независимых переключателей* ***checkbox***

Однако в HTML предусмотрен и такой переключатель, который зависит от други. Его мы и рассмотрим в следующем пункте.

Зависимые переключатели

Зависимый переключатель, так же как и независимый переключатель, может быть либо включен, либо выключен. При этом переключатель **radio** является зависимым переключателем, поскольку на форме может быть только один включенный переключатель типа **radio**. Точнее, если в форме присутствует несколько одноименных зависимых переключателей, то включен из них может быть только один. При выборе одного переклю-

чателя все одноименные зависимые переключатели автоматически выключаются. В качестве имени переключателей воспринимается значение атрибута **name**. Может быть только один активный переключатель. Рассмотрим такой пример, приведенный в листинге 6.1.

Листинг 6.1. Форма с зависимыми переключателями

```
<form action="http://localhost/sex.php" method="post">
<input type=radio
       name=sex
       value=male
       checked
>    Мужчина
<input type=radio
       name=sex
       value=female
>   Женщина
<input type=submit
       name=go
       value=Передать
>
</form>
```

В окне браузера наша форма будет выглядеть следующим образом

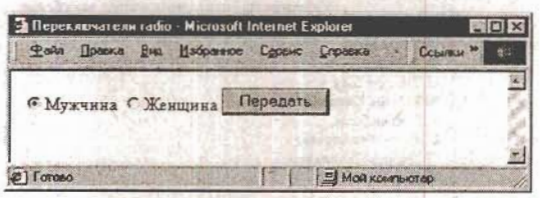


Рис. 6.3. Зависимые переключатели radio

Первый переключатель (со значением `Male`) активен по умолчанию (мы установили атрибут **checked**). Как только пользователь нажмет кнопку Передать, сценарию `sex.php` будет передан параметр `sex` (атрибут name обоих переключателей) со значением `Male`. Если же пользователь выберет другой вариант (`female`), сценарию будет передано соответствующее значение параметра sex.

Кнопка отправки формы

Последними элементами управления типа INPUT являются кнопки. Кнопка отправки служит для отправки сценарию введенных в форму значений. Синтаксис тега INPUT при этом таков:

```
<input type=submit
   [name=go]
   value=Передать
>
```

Атрибут **value** определяет текст, который будет написан на кнопке отправки. Атрибут **name** определяет имя кнопки и является необязательным. Если значение этого атрибута не указывать, то скрипту будут переданы введенные в форму значения и все. Если же атрибут **name** для кнопки будет указан, то дополнительно к данным формы будет отправлена пара **имя=значение** от самой кнопки. Я советую обязательно указывать этот атрибут — он нам очень пригодится при программировании на PHP.

Посмотреть, как выглядит кнопка отправки, можно на рис. 6.1.

Кнопка сброса

Кроме кнопки **submit** есть еще кнопка **reset**, которая сбрасывает форму — устанавливает для всех элементов формы значения по умолчанию. Желательно, чтобы на вашей форме была такая кнопка, особенно, если это большая форма. Наличие данной кнопки облегчает очистку формы, если были введены неправильные параметры.

```
<input type=reset
   value=Сброс
>
```

Кнопка отправки с индивидуальным рисунком

Вместо кнопки submit можно использовать рисунок для отправки данных. При щелчке на этом рисунке произойдет то же, что и при нажатии на кнопку **submit**. Однако, кроме этого, сценарию будут переданы координаты места произведения щелчка на рисунке. Координаты будут переданы в формате: имя.x=коор_X, имя.y=коор_У.

```
<input type=image
name=имя
src=рисунок
>
```

6.1.5. Ввод многострочного текста. Тег TEXTAREA

Мы переходим к рассмотрению многострочного поля ввода. В HTML эта возможность реализуется с помощью тега TEXTAREA. Поле, создаваемое этим тегом, позволяет вводить и отправлять не одну строку, а сразу несколько. Формат тега TEXTAREA таков:

```
<textarea
   name=имя
   [cols=ширина в символах]
   [rows=высота в символах]
   wrap=тип_переноса
>Текст по умолчанию </textarea>
```

Несколько замечаний относительно использования атрибутов: необязательные параметры **cols** и **rows** желательно указывать. Первый из них задает количество символов в строке, а второй — количество строк в области. Атрибут **wrap** определяет, как будет выглядеть текст в поле ввода:

- **Virtual** — справа от текстового поля выводится полоска прокрутки. Вводимый пользователем текст выглядит разбитым на строки, а символ новой строки вставляется при нажатии клавиши Enter.
- **Physical** — этот тип зависит от браузера и в разных браузерах может вести себя по-разному.
- **None** — текст выглядит в поле в том виде, в котором пользователь его вводит. Если текст не умещается в одну строку, появляется горизонтальная полоска прокрутки.

Стоит заметить, что наиболее удобным является тип **Virtual**. Внешний вид поля можно увидеть на рис. 6.4.

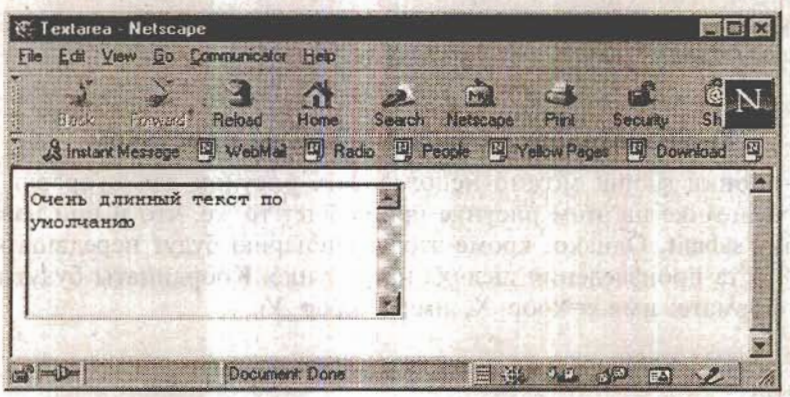


Рис. 6.4. *Поле TEXTAREA*

6.1.6. Списки выбора. Тег SELECT

Списки с единственным выбором

Довольно часто нам нужно представить какие-нибудь данные в виде списка и предусмотреть возможность выбора в этом списке. В HTML это реализуется в виде списков выбора, создавать которые можно с помощью тега **SELECT**. Список выбора позволяет выбрать один вариант из множества. Можно было бы использовать зависимые переключатели **radio**, но это не очень рационально. Например, если нам нужно, чтобы пользователь выбрал месяц, то один список будет занимать намного меньше места в окне браузера, чем целых двенадцать переключателей **radio**. Вот пример списка выбора, реализованного с помощью тега **SELECT**:

```
<select name=month size=1>
   <option value=01 >Январь</option>
   <option value=02>Февраль</option>
   <option value=03>Март</option>
   <option value=04>Апрель</option>
   <option value=05 selected >Май</option>
   <option value=06>Июнь</option>
   <option value=07>Июль</option>
   <option value=08>Август</option>
   <option value=09>Сентябрь</option>
   <option value=10>Октябрь</option>
   <option value=11>Ноябрь</option>
   <option value=12>Декабрь</option>
</select>
```

Как видно, варианты выбора задаются с помощью тегов **option**. При отправке формы сценарию будет отослано значение **value** выбранного элемента списка.

Атрибут **name** определяет имя параметра, который будет передан сценарию. Если атрибут **size** равен 1, то список будет выпадающим. В противном случае список будет занимать **n** строк и будет «оснащен» полосой прокрутки. Значение, выбранное в списке по умолчанию, можно указать с помощью атрибута **selected** для соответствующего тега **option**. В приведенном примере месяц по умолчанию — Май (рис. 6.5).

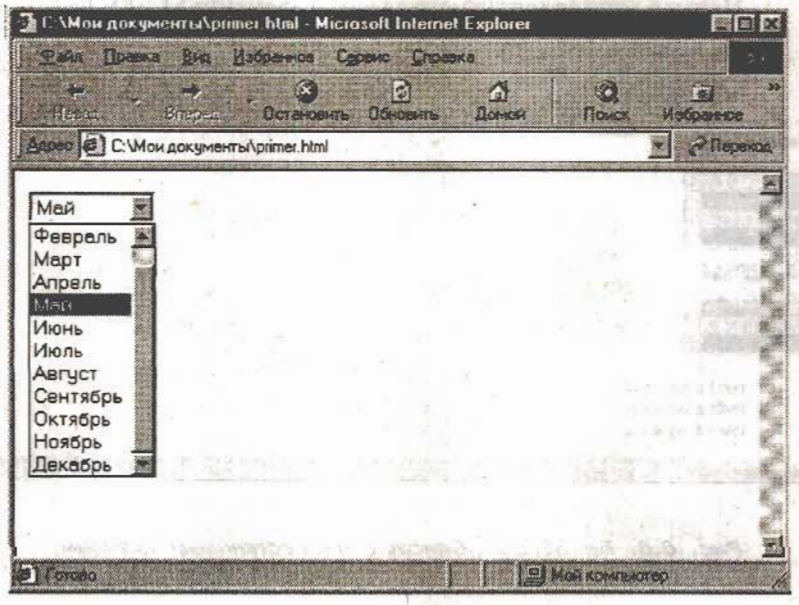


Рис. 6.5. *Тег SELECT. Список с единственным выбором*

Атрибут **value** является необязательным. Если его не указать, то будет передана строка, заключенная в тег option. В нашем случае сценарию будет передан параметр `month=Сентябрь` (если **value** не указан).

Списки множественного выбора

С помощью тега SELECT можно создавать и списки множественного выбора. В таких списках можно выбрать не одно, а сразу несколько значений. Чтобы сделать список с множественным выбором, необходимо для тега SELECT указать атрибут **multiple**.

Если пользователь выберет несколько вариантов, то сценарию будут переданы параметры в форме:

```
имя=значение&имя=значение...&имя=значение
```

О том, как обработать такие параметры с помощью PHP, вы узнаете ниже, в п. 6.2. Пример использования списка множественного выбора показан на рис. 6.6.

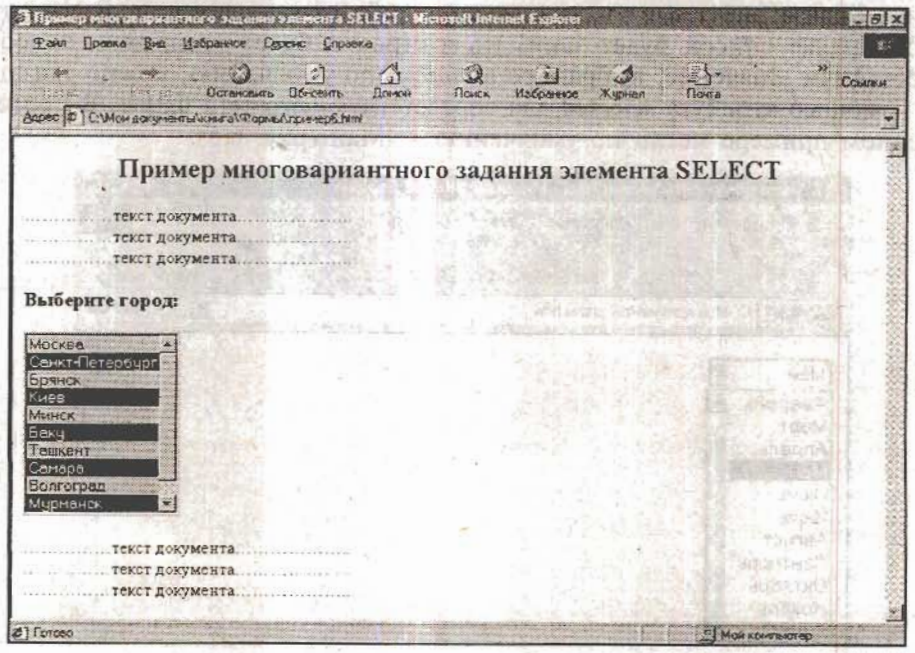


Рис. 6.6. Тег SELECT. Список с множественным выбором

6.1.7. Загрузка файлов

Тег INPUT позволяет реализовывать еще одну возможность в формах. Он позволяет создавать поле выбора файла для отправки. При этом формат тега таков:

```
<input type=file
   name=имя
   [value=имя файла]
>
```

Более подробно о загрузке файлов мы поговорим в главе «Работа с WWW».

6.2. Передача параметров с помощью формы

В этой главе мы поговорим о передаче параметров с помощью форм данных. Создадим HTML-документ с формой, использующей все типы полей, описанные в п. 6.1. А затем напишем сценарий, который будет обрабатывать переданные параметры. Код HTML-документа приведен в листинге 6.2.

Листинг 6.2. Все типы полей в одном HTML-документе

```
<html>
   <head>
   <meta http-equiv="Content-Type" content="text/html; charset=koi8-r">
   <title>Форма данных</title>
   </head>
   <body>
   <form action="http://localhost/form.php" method="POST">
      Текстовое поле txt <input type="text" name="txt"><br>
      Поле ввода пароля pswd <input type="password" name="pswd"><br>
      Скрытое поле hid <input type="hidden" name="hid" value="Hidden
      Field"><br>
      <hr>
      <input type="checkbox" name="var1" value="Вариант 1"
checked>Вариант 1 (по умолчанию)
      <input type="checkbox" name="var2" value="Вариант 2">Вариант 2
      <hr>
      <input type="radio" name="sex" value="male" checked>Мужчина
      <input type="radio" name="sex" value="female" >Женщина
      <hr>
      <textarea name="t_area" cols="20" rows="5" wrap="virtual">
```

```
      Текст по умолчанию </textarea>
      <hr>
      Список без возможности множественного выбора:<br>
      <select name="month" size=1>
        <option value=01 selected>Январь</option>
        <option value=02>Февраль</option>
        <option value=03>Март</option>
        <option value=04>Апрель</option>
        <option value=05>Май</option>
        <option value=06>Июнь</option>
        <option value=07>Июль</option>
        <option value=08>Август</option>
        <option value=09>Сентябрь</option>
        <option value=10>Октябрь</option>
        <option value=11>Ноябрь</option>
        <option value=12>Декабрь</option>
      </select>
      <br>Список с возможностью множественного выбора:<br>
      <select name="month2[]" size=4 multiple>
        <option value=01 selected>Январь</option>
        <option value=02>Февраль</option>
        <option value=03>Март</option>
        <option value=04>Апрель</option>
        <option value=05>Май</option>
        <option value=06>Июнь</option>
        <option value=07>Июль</option>
        <option value=08>Август</option>
        <option value=09>Сентябрь</option>
        <option value=10>Октябрь</option>
        <option value=11>Ноябрь</option>
        <option value=12>Декабрь</option>
      </select>
      <hr>
      <input type="submit" name="go" value="Send">
      <input type="reset" value="Clear form">
    </form>
    </body>
  </html>
```

В окне браузера наша форма будет выглядеть так, как на рис. 6.7.

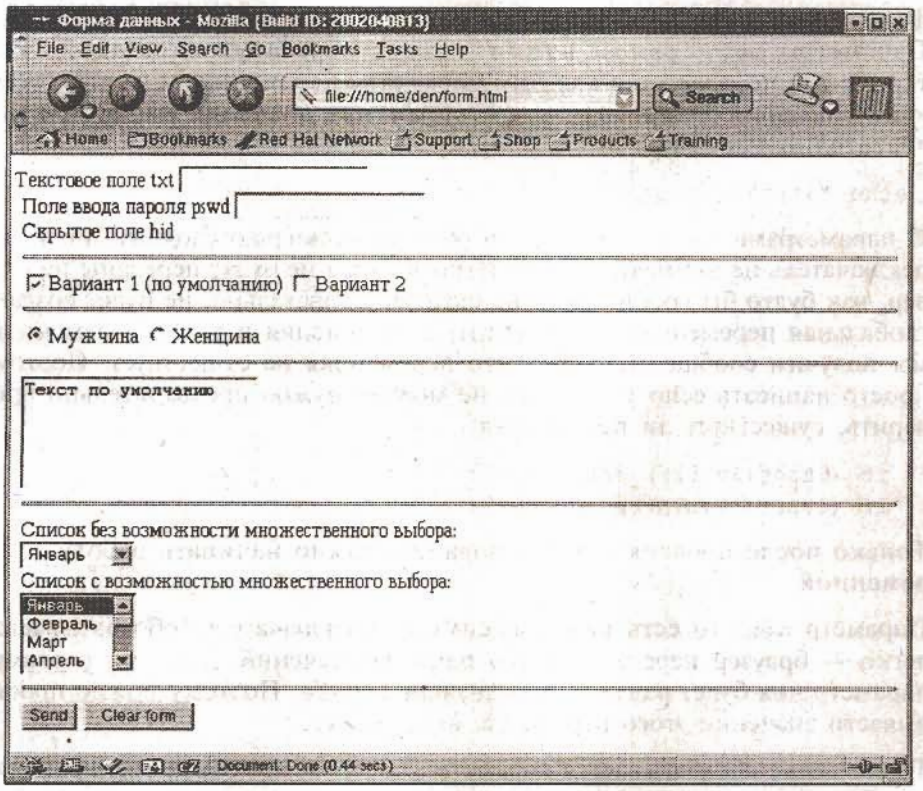


Рис. 6.7. *Форма в окне браузера*

Когда пользователь нажмет кнопку go, браузер передаст сценарию следующие параметры:

1. `txt` — значение текстового поля;
2. `pswd` — значения поля ввода пароля;
3. `hid` — значение скрытого поля;
4. параметры `var1` и `var2` будут переданы только, если соответствующие им переключатели активны (может быть передано одно из значений, оба значения или же ни одно из значений);
5. `sex` — значение группы зависимых переключателей (будет передано одно из значений — `male` или `female`);
6. `t_area` — содержимое текстовой области;
7. `month` — значение обыкновенного списка;
8. `month[]` — значение множественного списка.

Теперь нам нужно разобраться, как обработать все эти параметры. Параметры txt, pswd, hid и t_area обрабатываются легко. Интерпретатор создает для каждого параметра переменную с соответствующим именем. Например, для отображения значения параметра txt достаточно написать:

```
echo "$txt";
```

С параметрами var1 и var2 дело обстоит несколько сложнее. Если переключатель не активен, то параметры вообще не будут переданы на сервер, как будто бы вообще их и не было. Следовательно, не будет создана глобальная переменная и при попытке обращения к такой переменной, мы получим сообщение о том, что переменная не существует. Поэтому просто написать echo $var1; мы не можем: нужно предварительно проверить, существует ли переменная:

```
if (isset($var1)) echo "var1: $var1";
if (isset($var2)) echo "var2: $var2";
```

Только после проверки существования можно начинать работу с переменной.

Параметр sex, то есть наш зависимый переключатель, обрабатывается легко — браузер передает только одно из значений. Другими словами, параметр sex будет равен или male, или female. Поэтому можно просто вывести значение этого параметра: echo $sex;

Точно также обрабатывается параметр $month — это обыкновенный список без возможности множественного выбора, поэтому по сути его можно представить, как набор из 12 зависимых переключателей.

Сложнее всего обработать список с множественным выбором. Ведь в этом случае параметры передаются так:

```
name=value1&name=value2&name=value3...
```

При стандартном подходе переменная $name будет содержать только последнее значение (valueN). Однако разработчики PHP позаботились об этом: множественный список можно представить в виде массива, а обработать его элементы можно с помощью цикла **foreach**. Нам даже не нужно знать количество переданных элементов списка. Предварительно лишь нужно сообщить PHP, что мы будем передавать массив:

```
<select name="month2[]" size=4 multiple>
```

Квадратные скобки — это признак массива. Обрабатывается массив так:

```
foreach($month2 as $key=>$value)
 echo "$key = $value <br>";
```

Окончательный листинг сценария form.php представлен в листинге 6.3.

Листинг 6.3. Сценарий form.php

```
<?

/* Проверяем, нажал ли пользователь кнопку go */
if (!isset($go))
 {
    echo "Не указаны параметры\n";
    exit(1);
 }
else
 {
    /* Начинаем обработку параметров */
    echo "<html><body>";
    echo "<b>Текстовые поля</b><br>";
    echo "txt: $txt pswd: $pswd hid: $hid <br>";

    echo "<b>Checkbox</b><br>";
    if (isset($var1)) echo "var1: $var1";
    if (isset($var2)) echo "var2: $var2";

    echo "<br><b>Radio</b><br>";
    echo "sex: $sex";

    echo "<br><b>Область ввода текста</b><br>";
    echo "$t_area";

    echo "<br><b>Список month:</b> $month <br>";
    echo "<br><b>Список month2:</b><br>";
    foreach($month2 as $key=>$value)
     echo "$key = $value <br>";
   }

?>
```

Предположим, что в текстовое поле txt мы ввели имя Denis, в поле пароля — my_password, выбрали только вариант 2, установили параметр sex в значении male, заполнили текстовую область, в первом списке выбрали Январь, а во втором — первые три месяца. В результате мы получим такой вывод сценария:

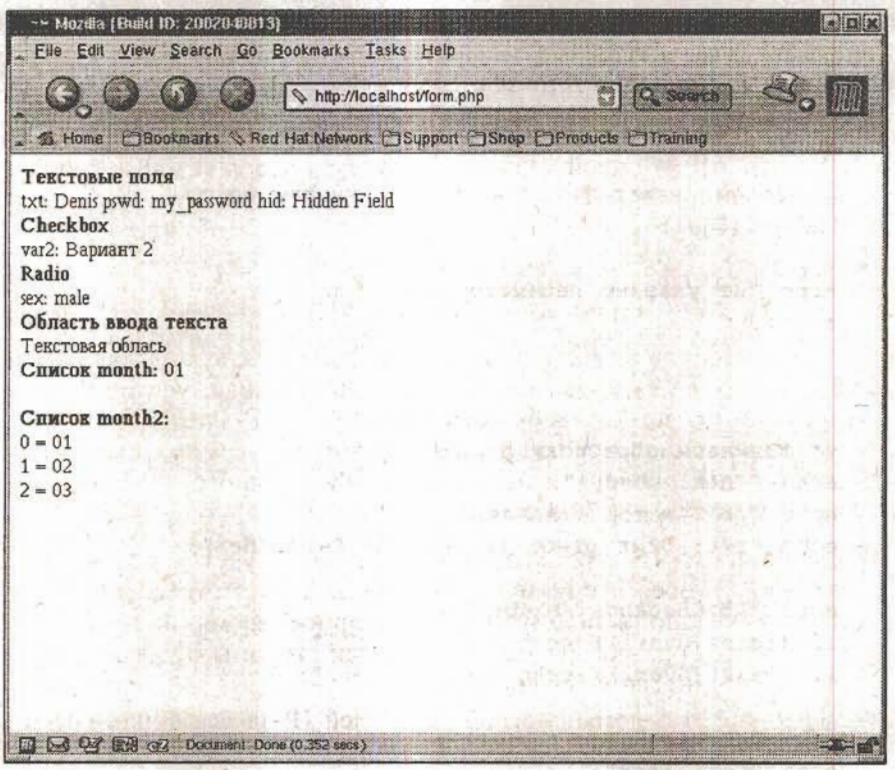


Рис. 6.8. *Результаты работы сценария form.php*

Вывод сценария в особых комментариях не нуждается, кроме трансляции списков. Как вы видите, передается не строка (Январь), а заданное значение value (01). Если не указывать значение **value**, то значением параметра станет строка, заключенная в тег **option**. Также обратите внимание на трансляцию множественного списка: выводится номер элемента (0, 1, 2) и значения (01, 02, 03) элементов массива.

6.3. Механизм Cookies

6.3.1. Принцип работы механизма Cookies

Мне не хочется забивать вам голову техническими подробностями, поэтому я кратко скажу, что такое Cookies и, нет, не с чем их едят, а как с ними нужно работать.

Представим, что перед нами поставлена такая задача: нужно написать индивидуальный счетчик посещения для каждого клиента сайта нашей

компании. То есть, чтобы пользователь знал, сколько раз именно он был на нашем сайте, а не видел общее количество посещений.

Можно решить данную задачу двумя способами. Первый способ заключается в том, чтобы вести учет по IP-адресу. Для этого нужна база данных, состоящая всего из одной таблицы, структура которой примерно такова:

```
IP-адрес          Посещения
111.111.111.111     23
111.123.215.211     100
...
```

Когда пользователь заходит на наш сайт, нам нужно определить его IP, найти в базе данных информацию о его посещениях, увеличить счетчик на единицу и вывести его в браузер. Определять IP-адрес клиента мы уже сумеем (переменная $REMOTE_ADDR), поэтому нам несложно написать такую программу. Однако при использовании данного способа рано или поздно «всплывут» две проблемы:

- Для каждого IP-адреса нужно будет вести учет в одной таблице, которая, скорее всего, будет большой, а из этого следует, что мы нерационально используем процессорное время и дисковое пространство. Гораздо правильнее с нашей стороны будет использовать это пространство с большей пользой.
- У большинства домашних пользователей IP-адреса динамические. То есть, сегодня у него адрес 111.111.111.111, завтра — 111.111.111.118. И попробуй потом определить, кто есть кто. Можно ввести регистрацию с использованием имени пользователя и пароля, но это еще больше увеличит нашу базу данных и создаст неудобства для клиентов.

Ясное дело, что данный способ нам не поможет. Остается использовать второй способ, который намного легче в реализации. Мы устанавливаем Cookie — переменную, которая будет **храниться на диске у пользователя**. Эта переменная и будет хранить информацию о посещениях. Она будет считываться сценарием при обращении пользователя к серверу. Выгода очевидна. Во-первых, нам не нужно хранить ненужную информацию о IP-адресах, во-вторых, нам не страшны динамические IP-адреса, поскольку данные о своих посещениях хранятся у клиента.

Теперь нам ясно, для чего используются Cookies — для хранения небольшой по объему информации у клиента, например, его логина (но ни в коем случае не пароля!), настроек сайта (цветовая гамма, язык, и т.д.) и другой информации.

Почему небезопасно хранить пароль в Cookies? Получив от сервера Cookie-переменную, браузер сохраняет ее на диске в открытом виде, то есть, не кодируя ее. Любой, кому не лень, может просмотреть значения этой переменной. Кроме того в некоторых браузерах Cookies вообще сохраняются в

обыкновенном текстовом файле, а некоторые — даже позволяют просмотреть их. Например, в браузере Mozilla для просмотра Cookies выполните команду меню **Edit → Preferences→ Privacy → Security → Cookies** (см. рис. 6.9).

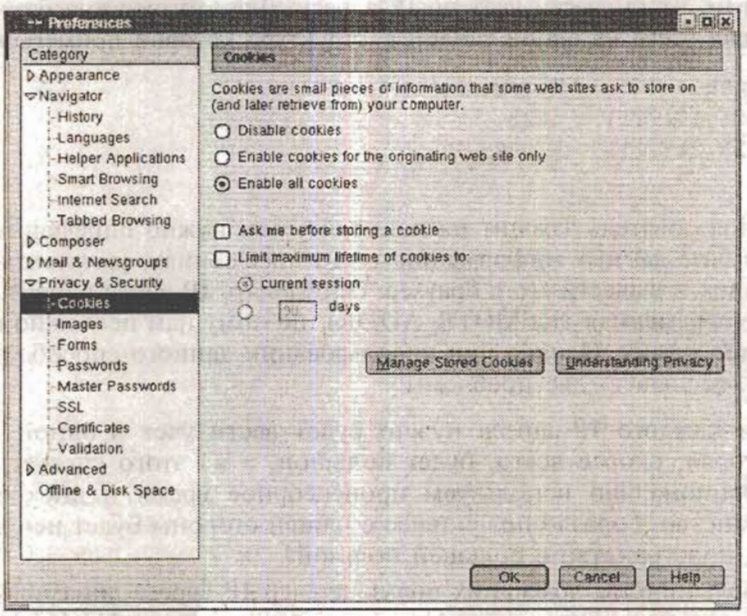


Рис. 6.9. *Управление параметрами Cookies*

В этом разделе вы можете включить (Enable) или отключить (Disable) прием Cookies, установить максимальный срок хранения Cookie (Limit maximum lifetime):

- Current session (Текущая сессия) — переменная будет удалена по окончании сессии работы с сайтом (например, когда пользователь закроет окно сайта).
- Установить определенное количество дней, по истечении которых все принятые Cookies будут удалены.

Очень не рекомендую устанавливать режим запрос на прием Cookies (Ask me before storing a cookie) — многие сайты устанавливают Cookies, и этот режим очень скоро вам надоест.

Нажмите кнопку Manage Stored Cookies. Появится окно с информацией обо всех установленных Cookies (см. рис. 6.10).

Щелкните на имени установленной Cookie-переменной, чтобы получить информацию о ней:

- **Name** — имя переменной;

- **Information** — информация, которую содержит Cookie;
- **Host** — узел, установивший переменную;
- **Secure Server** — признак того, что Cookie должна передаваться через защищенное (HTTPS) соединение;
- **Expires** — время жизни Cookie. Значение at end of session означает, что Cookie будет «жить» до конца сессии.

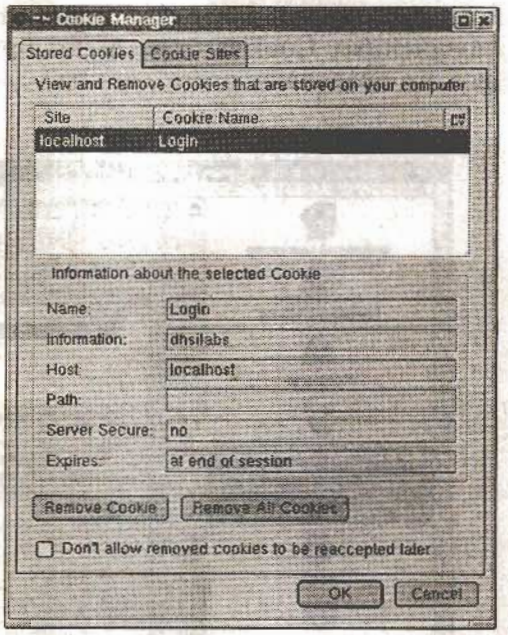


Рис. 6.10. *Установленные Cookies*

Кнопка **Remove Cookie** удаляет текущую переменную, а кнопка **Remove All Cookies** удаляет все Cookie-переменные.

Вы часто покупаете товары в Интернет-магазинах? Тогда рекомендую время от времени (скажем, раз в месяц) удалять все Cookies. Зачем? Вы увидите, что цены станут ниже. Дело в том, что некоторые Интернет-магазины (особенно, «буржуйские») грешат на доверии клиентов. Пару раз они могут продать вам товар по заниженным ценам, затем вы, зная, что это самый дешевый в Интернете магазин, будете покупать товары только у них. А они как раз и воспользуются Cookies — будут сознательно завышать цены на все товары, когда вы в очередной раз будете делать покупку, уверяя себя, что экономите деньги. При этом с помощью Cookies Интернет-магазины будут определять, кто вы такой и что вы раньше покупали.

Однако не нужно совсем отключать Cookies. Некоторые сайты станут загружаться некорректно. Если вы боитесь, что через Cookies можно получить вирус, то не беспокойтесь: даже если кто-то передаст вам с помощью Cookies «вредный код», то система не сможет его запустить, так как Cookies хранятся в текстовом виде (а в Linux для Cookies не устанавливается право за запуск файла).

Перед тем, как начать работу с Cookies, позволю себе пару замечаний относительно выбора браузера. Для отладки ваших сценариев, если они используют Cookies, я рекомендую использовать браузер Mozilla, поскольку он позволяет эффективно управлять Cookies. Если этот браузер вас почему-то не устраивает, используйте Konqueror,

который также позволяет просмотреть подробную информацию о полученных Cookies (рис. 6.11). Браузеры Internet Explorer версий 4 и 5 не позволяют этого сделать.

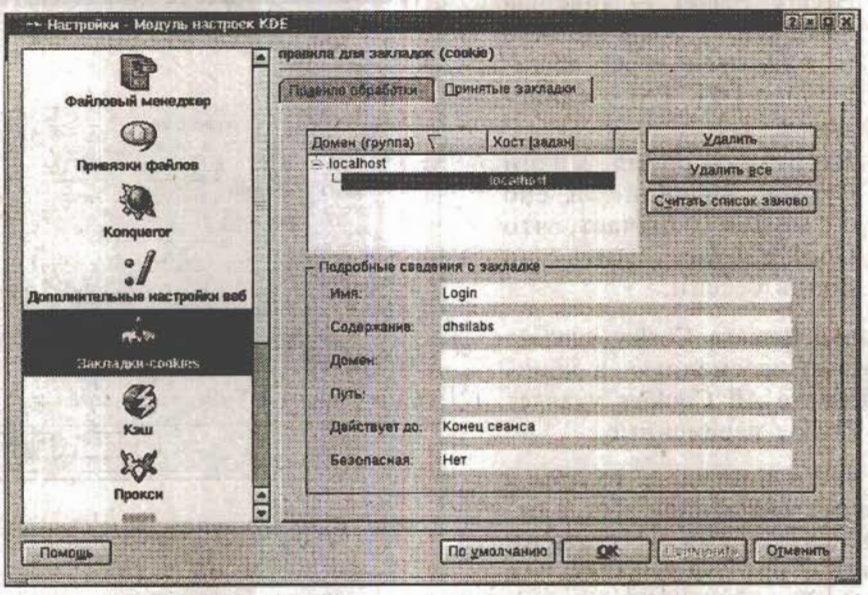


Рис. 6.11. Окно настройки браузера Konqueror

6.3.2. Программирование Cookies

Установка Cookies

Вот теперь можно приступить к программированию Cookies. Для установки Cookies используется функция **SetCookie**. В общем случае для этой функции можно указать целых шесть параметров, но обязательный из них является только один — **name**:

- **name** — задает имя (строка), закрепленное за Cookie.
- **value** — определяет значение переменной (строка).
- **expire** — время жизни переменной (целое число). Если этот параметр не указан, Cookie будет «жить» до конца сессии, то есть до

закрытия браузера. Если время указано, то, когда оно наступит, Cookie самоуничтожится.

- **path** — путь к Cookie (строка).
- **domain** — домен (строка). В качестве значения устанавливается имя хоста с которого установили.
- **secure** — передача через HTTPS-соединение.

Обычно используются только три первые параметра. В листинге 6.4 приведены несколько примеров установки Cookies.

Листинг 6.4. Примеры установки Cookies

```
<?

// До конца сессии
SetCookie("Test","Value");

// Время жизни - один час после установки
SetCookie("My_cookie","One hour",time()+3600);

?>
```

При использовании Cookies нужно иметь в виду, что Cookies должны быть установленными до первого вывода сценария (будь то оператор **echo** или вывод какой-нибудь функции). Поэтому желательно устанавливать Cookies в самом начале сценария. Дело в том, что Cookies устанавливаются с помощью определенного заголовка сервера, а если вы выводите что-нибудь, то это означает, что начинается тело документа. В результате Cookies не будут установлены. Вот что вы получите, если установка Cookies невозможна:

```
Warning: Cannot add header information - headers already sent
by (output started at /var/www/html/cookie.php:2) in /var/www/
html/cookie.php on line 4
```

Чтение значений Cookies

Получить доступ к Cookies достаточно просто. Все они хранятся в переменной окружения HTTP_COOKIE. Если мы установили несколько Cookies, то в переменной HTTP_COOKIE они будут перечислены через «;» (через точку с запятой). Например: `mycookie1=value1;mycookie2=value2`.

Для чтения значения Cookies можно обращаться к ним, как к обыкновенным переменным. Например: `echo $Test;` Только не забудьте проверить существование переменной. В заключение приведу пример «индивидуального счетчика» посещений. Код сценария приведен в листинге 6.5.

Листинг 6.5. Индивидуальный счетчик с помощью Cookies

```
<?

// Если переменная не существует, объявляем ее
if (!isset($Ind_Counter)) $Ind_Counter = 0;

// Увеличиваем счетчик
$Ind_Counter++;

/* Устанавливаем Cookie которая будет жить до переустановки
пользователем Windows, то есть очень долго. Если вам интересно,
то браузер
удалит ее 18/07/29 08:49 */

SetCookie("Ind_Counter",$Ind_Counter,0x6FFFFFF);

echo "Вы посетили этот сайт $Ind_Counter раз"

?>
```

Выше я говорил, что перед установкой Cookies нельзя использовать оператор **echo**. Только что я сам допустил подобную ошибку. Если оператора **echo** в вашей программе нет (по крайней мере до вызова **SetCookie()**), а Cookies невозможно установить, убедитесь в том, что символы маркера начала программы — <? — являются первыми в файле. У меня только что была такая ситуация:

```
начало файла
\n
<?

...
```

Естественно, что символ \n я не увидел, а PHP «подумал», что это фрагмент HTML-документа. В результате Cookies не были установлены, так как Cookies устанавливаются при формировании заголовка документа, а после первого вывода сценария формирование заголовка прекращается.

На этом данная глава заканчивается, но к Cookies мы еще будем не раз возвращаться.

Глава 7

Массивы и списки

7.1. Массивы и списки

Понятие массива

Массив — это упорядоченный набор данных. Каждый элемент массива имеет индекс или ключ. Индекс (ключ) служит для однозначной идентификации элемента внутри массива. В одном массиве не может быть двух элементов с одинаковыми индексами.

Работать без массива не очень удобно, думаю, вы и сами это понимаете. Существует ряд задач, где без использования массива просто не обойтись. Примером может послужить простой список друзей. Не будете же вы создавать 20 переменных, содержащих имена ваших друзей? Гораздо рациональнее создать один массив, содержащий 20 элементов.

Если вы программировали на C или Pascal, то знаете, что длина массива ограничена. При этом, если вы наперед не знаете, сколько элементов будет содержать ваш массив, значит вам нужно использовать динамические структуры (список, стек, очередь). В PHP все гораздо проще — длину массива не нужно задавать при объявлении массива, длина будет автоматически увеличена при добавлении нового элемента в массив. Все это становится возможным только благодаря тому, что PHP — интерпретатор, а не компилятор.

Очень важной особенностью PHP является то, что он, в отличие от других языков, позволяет создавать массивы любой сложности непосредственно в теле программы (сценария). Это бывает очень полезным, когда требуется создать массив, на основе некоторых, заранее неизвестных параметров.

Рассмотрим простейший способ инициализации массива:

```
$People[0]="Коля";
$People[1]="Витя";
$People[2]="Дима";
$People[3]="Марк";
```

PHP узнает о том, что создается массив по квадратным скобкам. В квадратных скобках указывается индекс (ключ) отдельного элемента массива. В приведенном примере мы объявили массив `$People`, состоящий из четырех элементов. Однако в дальнейшем нам никто не помешает инициализировать еще один (или сколько нужно) новый элемент массива. В PHP размер массива не задается явно, как в других языках программирования.

Обратиться к элементу массива можно по имени массива и индексу элемента внутри массива:

```
echo $People[1];
```

Зная количество элементов массива, мы можем вывести все элементы массива с помощью цикла:

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

Если мы не знаем точное количество элементов массива, можно использовать функцию **count()**. Эта стандартная функция определяет размер массива, то есть количество элементов в нем. Соответственно, с использованием функции **count()** вышеприведенный цикл выглядит следующим образом:

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

Понятие списка

Массивы, индексами которых являются числа, начинающиеся с нуля без пропусков, будем называть списками. С технической точки зрения разницы между массивами и списками нет.

Не знаю как вам, но мне удобнее создавать списки без указания индекса — пусть это сделает за меня PHP. Ведь я могу ненароком указать уже существующий индекс. Кроме того, такое автоматическое создание массивов удобно при записи текстового файла в массив для последующей обработки.

Вот пример автоматического создания массива:

```
$People[]="Коля";
$People[]="Витя";
$People[]="Дима";
```

При этом PHP начнет нумерацию с наименьшего незанятого значения индекса. В частности, если переменная **$People** только сейчас инициализируется, то нумерация будет начата с нуля. В дальнейшем, с каждым новым элементом массива индекс будет увеличиваться на единицу.

Как видите все просто и удобно — не нужно указывать индексы. Однако, это как раз тот случай, когда мы не знаем точного числа элементов в массиве.

Соответственно, при работе с такими массивами целесообразно использовать функцию **count()**, которая определяет количество элементов в массиве.

Рассмотрим небольшой пример — чтение текстового файла в массив. Забегая наперед, скажу, что функция **fopen()** используется для открытия файла, функция **feof()** истинна, если достигнут конец файла, а функция **fgets()** читает строку из файла.

При работе с файлами мы не знаем, сколько строк будет содержать конкретный файл. Программируя на C или Pascal, нам нужно было бы создать динамический список, что вызывает проблемы у начинающих программистов, не знакомых с принципами работы с динамической памятью. В PHP все просто — вы можете добавить сколько угодно элементов в массив, не указывая их номера. Новые элементы массива будут добавлены в конец массива.

```
// Открываем файл
$f = fopen("file.txt","rt") or die("Ошибка");
// Читаем построчно
while(!feof($f))
{
// Заносим новый элемент (строку, прочитанную из файла)
// в конец массива
$File[ ] = fgets($f,50);
   }
// Закрываем файл
fclose($f);
```

7.2. Ассоциативные массивы

Мы договорились, что массивы, индексами которых являются числа начинающиеся с нуля, будем называть списками. Однако в PHP индексом массива может быть и строка. Причем на эту строку не накладывается никаких ограничений: она может содержать пробелы, специальные символы, также нет ограничений на длину строки.

Массивы, индексами которых являются строки, называются ассоциативными. Индексы ассоциативных массивов называются ключами. Пример ассоциативного массива:

```
$People["Иванов"] = "Иван";
$People["Сидоров"] = "Николай";
$People["Петров"] = "Петр";
```

Доступ к элементам ассоциативных массивов осуществляется так же, как и к элементам обыкновенных массивов, и называется **доступом по ключу**:

```
echo $People["Иванов"];
```

7.3. Инструкции list()

Предположим, что у нас есть массив, состоящий из трех элементов:

```
$People[0] = "Иван";
$People[1] = "Николай";
$People[2] = "Петр";
```

Теперь допустим, что в какой-то момент нам необходимо передать значения всех трех элементов массива, соответственно, трем переменным `$Ivanov`, `$Sidorov`, `$Petrov`, в которые мы хотим записать имена Иванова, Сидорова и Петрова. Это можно сделать так:

```
$Ivanov = $People[0];
$Sidorov = $People[1];
$Petrov = $People[2];
```

Согласитесь, что если массив большой, то такой подход не очень удобен. Существует более рациональное решение — использовать инструкцию **list()**, которая специально для этого предназначена:

```
list($Ivanov,$Sidorov,$Petrov) = $People;
```

Если же нам нужны имена только Сидорова и Петрова, первый параметр можно пропустить:

```
list(,$Sidorov,$Petrov) = $People;
```

7.4. Инструкция array() и создание многомерных массивов

Инструкция **array()** используется специально для создания массивов. При этом она позволяет создавать пустые массивы, чего никак нельзя сделать рассмотренными ранее механизмами. Методы использования инструкции **array()** рассмотрим на следующих примерах:

```
$A = array(); // создает пустой массив
$A = array("Иванов"=>"Иван","Петров"=>"Петр"); // создает асso
                // циативный массив с двумя элементами
$A = array("Иванов","Петров");      // создает список с двумя
                // элементами. Индексы нумеруются с нуля.
```

Инструкцию **array()** удобно использовать для создания многомерных массивов. Предположим, что мы хотим создать массив, элементами которого будут ассоциативные массивы с ключами `name`, `sex` и `email`.

Создать такой массив можно с помощью следующих операторов:

```
$A["Ivanov"] = array("name"=>"Иванов И.", "sex"=>"M", "email"
=> "ivanov@mail.ru");
$A["Petrov"] = array("name"=>"Петров И.", "sex"=>"M", "email"
=> "petrov@mail.ru");
$A["Sidorova"] = array("name"=>"Сидорова. Н", "sex"=>"F",
"email" => "sidorova@mail.ru");
```

Это же самое можно записать в следующем виде:

```
$A = array (
   "Ivanov" = array("name"=>"Иванов И.", "sex"=>"M", "email" =>
"ivanov@mail.ru"),
   "Petrov" = array("name"=>"Петров И.", "sex"=>"M", "email" =>
"petrov@mail.ru"),
   "Sidorova" = array("name"=>"Сидорова. Н", "sex"=>"F", "email"
=> "sidorova@mail.ru")
   );
```

Добраться до элементов массива можно так:

```
echo $A["Ivanov"]["name"];  // выведет Иванов И.
echo $A["Ivanov"]["email"]; // выведет ivanov@ivanov.org
```

Многомерные массивы в PHP больше положи на записи (record) в языке Pascal или структуры в языке C, чем на многомерные массивы в обычном понимании.

7.5. Операции над массивами

Мы уже познакомились с основной операцией над массивами — доступ по ключу, поэтому рассматривать ее сейчас я не вижу смысла. Сейчас мы рассмотрим три операции с массивами:

- Удаление.
- Слияние.
- Перебор.

7.5.1. Удаление массивов

Для удаления массива, как и любой другой переменной, используется уже знакомая нам функция **UnSet()**:

```
UnSet($People);
```

Обычно удаление может потребоваться для очень большого массива, чтобы освободить используемую им память.

7.5.2. Слияние массивов

Стандартное слияние

Слияние (конкатенация) — это операция создания массива, состоящего из элементов нескольких других массивов. Слияние массивов — это очень опасная операция, поэтому старайтесь избегать ее. Результат слияния подчиняется своей логике, забыв о которой можно потерять данные. Слияние реализуется при помощи оператора +.

Итак, пусть у нас есть два массива:

```
$A = array("1"=>"первый","2"=>"второй");
$B = array("3"=>"третий","4"=>"четвертый");
```

Теперь, давайте сольем эти два массива в один массив $C:

```
$C = $A + $B;
```

Сразу нужно сказать, что оператор «+» для массивов не коммутативен, а это значит, что $A + $B не равно $B + $A. В результате слияния $A + $B мы получим:

```
"1"=>"первый","2"=>"второй","3"=>"третий","4"=>"четвертый"
// именно в таком порядке
```

А в результате $B + $A, получим такой массив:

```
"3"=>"третий","4"=>"четвертый","1"=>"первый","2"=>"второй"
```

Особенно внимательным нужно быть при слиянии списков. Предположим, у нас опять есть два массива:

```
$A = array(10,11,12);
$B = array(15,17,18);
```

В результате слияния $C = $A + $B мы получим следующий массив:

```
10,11,12
```

Неожиданный результат неправда ли. Скорее всего вы ожидали получить массив array (10, 11, 12, 15, 17, 18). Однако в PHP на этот счет свои правила. При слиянии массивов, в которых содержатся элементы с одинаковыми индексами (ключами), в результирующем массиве останется элемент первого массива, причем на том же самом месте. А в приведенном примере элементы массива $A имели индексы 1, 2, 3 и элементы массива $B имели индексы 1, 2, 3. А поскольку массив $A при сложении стоит первым, то его элементы и являются результирующими. Можете убедиться в этом еще раз следующим образом:

```
$A = array(10,11,12);
$B = array(15,17,18);
$A = $A + $B;    // или $A += $B;
```

В этом примере элементы массива $A не изменятся. Изменить элементы массива можно только при помощи прямого перебора (см. следующий раздел).

Слияние с помощью функции array_merge()

Функция **array_merge()** предназначена для слияния двух массивов. Причем при слиянии массивов эта функция устраняет все недостатки оператора +. Она соединяет массивы, переданные ей в качестве аргументов, в один большой массив. Если в соединяемых массивах встречаются одинаковые ключи (ключ=>значение), то в результирующий массив помещается пара **ключ=>значение** из массива, который стоит правее в списке аргументов функции **array_merge()**. Однако это правило не затрагивает списки, то есть массивы с числовыми индексами. В этом случае элементы с повторяющимся числовым индексом помещаются в конец результирующего массива в любом случае.

Проиллюстрируем сказанное примером:

```
$A = array(10,11,12);
$B = array(15,17,18);
$C = $A + $B;          // $C = array(10,11,12);
$C = array_merge($A,$B);  // $C = array(10,11,12,15,17,18);
```

Таким образом, функцию **array_merge()** очень удобно использовать для слияния списков.

7.5.3. Перебор массива

Для перебора массивов в PHP можно использовать два метода:

- косвенный перебор;
- прямой перебор.

Косвенный перебор

Принцип косвенного перебора заключается в том, что сначала вычисляется очередной индекс (ключ), а затем по этому индексу косвенно находится значение элемента массива. Например:

```
$People[0]="Коля";
$People[1]="Витя";
$People[2]="Дима";

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

Это мы с вами перебрали список, индексами которого являются целые числа. Для этого мы использовали цикл **for**, в котором перебирали ин-

дексы, последовательно увеличивая их на 1, и по индексам получали элементы массива. Однако, что делать, если нам нужно перебрать не список, а ассоциативный массив? Ведь в ассоциативном массиве индексом (ключом) является не число, а строка:

```
$People["Иванов"] = "Иван";
$People["Сидоров"] = "Николай";
$People["Петров"] = "Петр";
```

Выполнить перебор такого массива поможет следующий цикл:

```
for(Reset($People); ($k = key($People)); Next($People))
  echo "$k = $People[$k] <br>";
```

Здесь функция **Reset()** устанавливает текущий элемент массива **$People[]** на первую позицию, как бы «перематывает» массив на начало. Функция **key()** возвращает ключ текущего элемента. Если нет больше элементов в массиве, то эта функция возвратит 0, и цикл прекратит свою работу. Функция **Next()** перемещает текущий элемент на одну позицию вперед (то есть переходит к следующему элементу массива).

В теле цикла можно также использовать функцию **current()**, которая возвращает значение текущего элемента. С использованием этой функции вышеприведенный цикл можно переписать так:

```
for(Reset($People); ($k = key($People)); Next($People))
  echo "$k =".current($People)."<br>";
```

До сих пор мы перебирали ассоциативный массив с начала в конец. Тем не менее, в PHP имеется возможность перебора такого массива и с конца. При этом вам лишь надо использовать функции **End()** и **Prev()**, вместо функций **Reset()** и **Next()**. Первая из них устанавливает текущий элемент в конец массива, а вторая перемещает текущий элемент на одну позицию назад (то есть переходит к предыдущему элементу):

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

Прямой перебор

Принцип прямого перебора заключается в том, чтобы при каждой итерации цикла одновременно получать и ключ, и значение текущего элемента массива. Для прямого перебора в версиях, начиная с PHP 4, используется цикл **foreach**, который мы уже неоднократно использовали в этой книге:

```
foreach ($People as $key => $value) echo "$key = $value";
```

Недостатки косвенного перебора

Косвенный перебор очень прост, а код, написанный с использованием косвенного перебора, очень читаем. Даже если вы вообще не знаете PHP, вы понимаете, что здесь и к чему. Однако косвенный перебор имеет существенные недостатки. Один из недостатков должен сразу броситься вам глаза: если в массиве встретится элемент с индексом 0, то после того, как он будет передан в цикл функцией **key()**, цикл прекратит свою работу.

И еще один недостаток: вы не можете использовать косвенный перебор во вложенных циклах, так как функция **Next()** второго цикла изменит положение текущего элемента в первом цикле.

Прямой перебор лишен этих недостатков.

7.6. Сортировка массивов

Для сортировки массивов в PHP предназначено много различных функций, но мы сейчас рассмотрим только основные из них:

asort() и **arsort()** — сортировка по значениям;
ksort() — сортировка по ключам;
sort() — сортировка списка;
array_reverce() — «переворачивание» массива;
shuffle() — перемешивание списка.

Сортировка массива по значению. Функции asort() и arsort()

Функция **asort()** выполняет сортировку массива по значениям элементов. При этом она располагает элементы массива таким образом, что их значения шли в алфавитном порядке (если это строки) или в возрастающем порядке (если это числа). Функция **arsort()** выполняет сортировку по значениям в обратном порядке.

Предположим, что у нас есть такой массив:

```
$FIO = array("Sidorov"=>"Nikolay","Ivanov=>"Alexandr","Petrov"=>"Petr");
```

После выполнения функции **asort($FIO)** наш массив будет преобразован следующим образом:

```
array("Ivanov" => "Alexandr", "Sidorov"=>"Nikolay","Petrov"=>"Petr");
```

Сортировка массивов по ключам. Функции ksort() и krsort()

Для сортировки по ключам используется функция **ksort()**. При этом сортировка осуществляется по таким же принципам, что и функцией **asort()**, только не по значению, а по ключам. Если передать ей наш массив `$FIO`, то мы получим такой результат:

```
array("Ivanov"=>"Alexandr","Petrov"=>"Petr","Sidorov"=>"Nikolay");
```

Сравните полученный результат с результатом функции **asort()**. Для сортировки ключей в обратном порядке используется функция **krsort()**.

Сортировка списка

Все четыре рассмотренные функции не разрывают пары ключ=>значение. Это очень важно. Например, у нас есть небольшой список:

```
$People = array("Ivanov"=>"555-6677","Sidorov"=>"345-4567","Petrov"=>"777-
5653");
```

После сортировки массива по ключам наш список будет выглядеть так:

```
$People = array("Ivanov"=>"555-6677","Petrov"=>"777-5653","Sidorov"=>"345-
4567");
```

Обратите внимание: пары **ключ=>значение** не разорваны, то есть у Петрова по-прежнему номер телефона 777-5653, а у Сидорова — 345-4567. Если бы пары были разорваны, номер телефона у Петрова был 345-4567, а у Сидорова — 777-5653, то есть, наоборот.

Если вам нужно отсортировать список, то удобнее всего использовать функцию **sort()**. Данная функция разрывает пары **ключ=>значение**, поэтому ее нужно использовать только для списков.

Пусть у нас есть список:

```
$List = array("Name","Address","Email");
```

Если вывести список в виде ключ=>значение, то мы получим следующий результат:

```
0 => Name
1 => Address
2 => Email
```

После сортировки **sort($List)** у нас получится следующий список:

```
0 => Address
1 => Email
2 => Name
```

Изначальная связь между парами **ключ=>значение** была разорвана. Возникнет как раз та ситуация, которую мы рассматривали на примере с номерами телефонов.

Переворачивание массива. Функция array_reverse()

Для переворачивания массива используется функция **array_reverse()**. Эта функция возвращает массив, элементы которого следуют в обратном порядке относительно исходного массива. Например, чтобы отсортировать массив в обратном порядке, можно использовать функцию **arsort()**,

а можно упорядочить массив в прямом порядке функцией **asort()** и перевернуть его функцией **array_reverse()**.

Примечание.

Если вам нужно отсортировать массив в обратном алфавитном порядке, я рекомендую использовать функцию **arsort()** — она работает быстрее, чем последовательность функций **asort()** и **array_reverse()**.

Перемешивание списков. Функция shuffle()

Функция **shuffle()**, так сказать, «перемешивает» список, то есть значения результирующего списка будут размещены в случайном порядке. Данная функция полностью изменяет массив, поэтому ее можно использовать только для списков.

Перед вызовом этой функции нужно инициализировать генератор случайных чисел, иначе при каждом новом вызове функции **shuffle()** результат перемешивания будет один и тот же (при перемешивании одного и того же массива). Для инициализации генератора случайных чисел используется функция **mt_srand()**:

```
mt_srand(time()*100000);
shuffle($Arr);
```

Примечание.

Для генерирования случайных чисел рекомендуется использовать функции mt_rand() и mt_srand() вместо стандартных rand() и srand(), поскольку первые обеспечивают лучшее качество генерируемых случайных чисел.

Пользовательская сортировка массивов. Функции uksort(), uasort() и usort()

Иногда нужно отсортировать массив не в алфавитном порядке, а по более сложному критерию. Для этого используются функции **uksort()**, **uasort()** и **usort()** — пользовательские функции сортировки массивов. Каждой функции нужно передать два параметра:

- массив, который нужно отсортировать;
- функцию сравнения, которая определит, какой из элементов массива меньше, а какой больше, согласно нашему критерию.

Функции сравнения нужно передать два параметра — два элемента массива ($el_1 и $el_2). В свою очередь, функция должна возвратить одно из трех значений:

-1, если `$el_1<$el_2`;
0, если `$el_1=$el_2`;
1, если `$el_1<$el_2`.

Предположим, что у нас есть список файлов и каталогов — `$FilesNDirs`. Нам нужно вывести сначала каталоги, а потом файлы. Сравнение будет производиться функцией `Cmp`, которую мы сами и напишем (см. листинг 7.1). Подробнее о создании и использовании пользовательских функций вы можете прочитать в гл. 8.

Листинг 7.1. Функция сравнения Cmp

```
function cmp($el_1,$el_2)
{
// Сначала каталог, а потом - файл
if (is_dir($el_1) && !is_dir($el_2)) return -1;
if (!is_dir($el_1) && is_dir($el_2)) return 1;

// Сравниваем по алфавиту
if($el_1<$el_2) return -1;
   elseif($el_1>$el_2) return 1;
         else return 0;
}
```

Теперь вызовем функцию **uasort()**:

```
uasort($FilesNDirs,"cmp");
```

Примечание.

В данном случае мы используем **uasort()** потому, что у нас простой список, а не ассоциативный массив. В последнем случае нужно было использовать функцию **uksort()**.

Функция **uasort()** аналогична функции **uksort()**, только она сортирует не ключи, как uksort, а значения. Обе функции — **uksort()** и **uasort()** — сохраняют пары **ключ=>значение**.

Функция **usort()** похожа на функцию **sort()**, но критерий сортировки задается пользовательской функцией, которая передается в качестве второго параметра. Еще одна важная особенность этой функции заключается в том, что она **не** сохраняет пару **ключ=>значение**, поэтому пригодна лишь для сортировки списков, но ее ни в коем случае не нужно использовать для сортировки ассоциативных массивов.

7.7. Получение части массива

Для получения части массива используется функция **array_slice()**. Ей нужно передать три параметра:

- **имя исходного массива,** из которого будут выбираться элементы;

- **смещение от начала массива**, то есть сколько элементов от начала массива надо пропустить перед началом выборки;
- **длина**, то есть количество элементов, которое нужно выбрать из исходного массива.

Данная функция возвращает часть указанного массива, начиная с указанного смещения от начала массива. Выборка элементов осуществляется подряд. Длина результирующего массива задается параметром **Длина**, который является необязательным. Если этот параметр не указан, то выборка элементов осуществляется до конца исходного массива.

Вот небольшие примеры:

```
$A = array(xox, ror, tot, pop, mom); // задаем исходный массив
$B = array_slice($A,2);              // получаем массив $B,
                                     // в котором содержатся
                                     // все элементы массива $A,
                                     // начиная с третьей позиции
                                     // (т.к. два надо пропустить).
                                     // Таким образом, массив $B
                                     // будет содержать элементы
                                     // tot, pop, mom.
$B = array_slice($A,0,2);            // Получаем массив $B, в котором
                                     // содержатся два элемента
                                     // массива $A, начиная с первого
                                     // (т.к. надо пропустить 0
                                     // элементов). Таким образом,
                                     // массив $B будет содержать
                                     // элементы  xox, ror.
```

Параметры **смещение** и **длина** могут быть отрицательными — в этом случае отсчет производится от конца массива:

```
$B = array_slice($A,2,-1);           // получаем массив $B,
                                     // состоящий из элементов tot,
                                     // pop.
```

Функция **array_splice()** похожа на функцию **array_slice()**. Ей нужно передать еще один дополнительный параметр — массив (назовем его `$R`). Как и **array_slice()**, функция **array_splice()** возвращает массив указанной длины, начиная со смещения `$offset`, но и заменяет переданные ей элементы на содержимое массива `$R`. В принципе, вместо массива `$R` можно указать обыкновенное строковое значение. Пример:

```
$A = array("Audi","Mercedes","Toyota","Honda","Renault");
array_splice($A,1,count($A), "BMW");
// $A = array("Audi","BMW");
```

7.8. Добавление и удаление элементов массива

Добавление в конец массива

Ранее мы с вами освоили простейший способ добавления новых элементов в массив, осуществляемый с помощью оператора []. Однако в последних версиях PHP (начиная с 4) предусмотрена функция **array_push()**, которая добавляет элементы в конец массива так же, как и оператор []. Например, следующие два оператора аналогичны:

```
$A[] = 5;
array_push($A,5);
```

Может возникнуть справедливый вопрос: «А зачем нужны два одинаковых способа делать одно и то же?». Ответ заключается в удобстве. Если необходимо добавить один элемент в массив (как в приведенном выше примере), то удобнее использовать оператор []. Использование же функции **array_push()** оправдывает себя тогда, когда нам нужно добавить в конец массива несколько новых элементов. Например:

```
$A[] = 5;
$A[] = 7;
$A[] = 9;
$A[] = 10;
$A[] = 15;
```

Вместо этого можно записать:

```
array_push($A, 5,7,9,10,15);
```

Примечание.

Функция **array_push()**, помимо всего прочего, возвращает новый размер массива. То есть, если помимо того, чтобы добавить новые элементы в массив, вам нужно узнать и его новую длину, то можно записать $D = array_push($A, 5, 7, 9, 10, 15). Таким образом, новая длина массива будет присвоена переменной $D.

Удаление последнего элемента списка

Функция **array_pop()** противоположна функции **array_push()**. Она «выталкивает» вершину списка (последний элемент списка). Другими словами, функция **array_pop()** возвращает значение последнего элемента массива и удаляет его из списка.

Функции **array_pop()** нужно передать всего один параметр — имя массива (списка):

```
$A = array(xox, ror, tot, pop, mom);
$T=array_pop($A);          // переменной $T присвоили
                           // значение mom, а массив $A
                           // лишился этого последнего
                           // элемента и теперь состоит из
                           // элементов xox, ror, tot, pop.
array_pop($A);             // массив $A еще раз лишился последнего
                           // элемента и теперь
                           // состоит из элементов xox, ror, tot.
```

Примечание.

С помощью функций **array_push()** и **array_pop()** мы можем строить структуры LIFO (Last In First Out — «последний вошел — первый вышел»), напоминающие стек. Структуры типа стек широко применяются в программировании. Классическое применение стека — обратная польская запись, которая используется для разбора арифметических выражений и вычисления их значения.

Добавление в начало списка. Удаление первого элемента списка

Функции **array_unshift()** и **array_shift()** похожи на функции **array_push()** и **array_pop()**, но только работают не с концом списка, а с его началом. Функция **array_unshift()** добавляет элементы в начало списка, а функция **array_shift()** возвращает значение первого элемента списка и удаляет его из списка (удаляет «голову» списка):

```
$A = array(3,4,5);
array_unshift($A, 1,2);   // получаем $A = array(1,2,3,4,5)
array_shift($A);          // получаем $A = array(2,3,4,5);
```

Примечание.

Функции **array_unshift()** и **array_shift()** полезно использовать, если вы хотите построить очередь типа FIFO (First In First Out — «первый вошел — первый вышел»). Очередь необходима для последовательной обработки элементов массива. Стек и очередь в языках C и Pascal являются основными динамическими структурами. Поскольку длина массивов в PHP ограничена свободными системными ресурсами, потребность в стеке и очереди исчезает. Функции для организации стека и очереди были написаны исключительно для удобства программиста — вдруг вы, программируя на C, привыкли работать именно со стеком или с очередью?

Функция array_unique()

Функция **array_unique()** предназначена для извлечения уникальных значений массива вместе с ключами этих значений. Вот пример применения этой функции:

```
$Arr = array("a"=>"AAA","b"="BBB","c","d","e"="BBB", "d");
$Unique_Arr = array_unique($Arr);
// $Unique_Arr = array("a"=>"AAA","b"="BBB","c","d");
```

7.9. Переменные и массивы. Взаимные переходы

Упаковка переменных в массив. Функция compact()

Функция **compact()** «упаковывает» переменные и их значения в массив. В результате получается ассоциативный массив, ключами которого являются имена переменных, а значениями элементов — значения переменных.

Рассмотрим сказанное на примере. Предположим, что у нас есть две переменные:

```
$Name = "Вася";
$Email = "vasya@vvv.org";
```

Функция compact("Name","Email") упакует переменные в один массив:

```
$A = compact("Name","Email");
//получаем  $A =array("Name"=>"Вася","Email"=>"vasya@vvv.org");
```

Параметры функции **compact()** описаны как **mixed**, потому что параметры могут быть не только строками, но и списками строк. Если так оно и есть, функция перебирает все элементы этого списка и упаковывает переменные, имена которых она встретила. Вот небольшой пример:

```
$a = "AAA";
$b = "BBB";
$c = "CCC";
$Arr = array("b","c");
$Massiv = compact("a",$Arr);

// $Massiv = array("a"=>"AAA","b"="BBB","c"="CCC");
```

Разупаковка (извлечение) переменных из массива. Функция extract()

Существует функция, противоположная функции **compact()** — это функция **extract()**. Она действует «с точностью до наоборот», то есть создает переменные с именами, соответствующими именам ключей и присваивает им значения элементов массива.

Распакуем наш «упакованный массив»:

```
extract($A);
```

В процессе распаковки может оказаться, что уже существует переменная с именем, как имя ключа в массиве $A. Для решения этой проблемы у функции **extract()** предусмотрен второй параметр, который позволяет определить поведение функции **extract()** в случае совпадения имен. Значение EXTR_OVERWRITE означает, что существующая переменная с именем ключа будет перезаписана, а EXTR_SKIP — пропущена:

```
$Name = "Вася";
$Email = "vasya@vvv.org";
$A = compact("Name","Email");
$Name = "Ivan";

extract($A,EXTR_OVERWRITE);    // $Name = "Ivan", так как мы
                               //указали EXTR_OVERWRITE
```

Значение EXTR_OVERWRITE используется по умолчанию и может не указываться. То есть достаточно указать **extract($A)** и все переменные будут перезаписаны. Кроме того, в PHP предусмотрено еще одно значение второго параметра функции **extract()** — это значение EXTR_PREFIX_ALL. В этом случае при разупаковке массива все имена создаваемых переменных будут предварены специальным префиксом, который нужно будет указать в качестве третьего параметра функции **extract()**.

Например, если мы в рассмотренном выше примере разупаковку произведем следующим образом:

```
extract($A,EXTR_PREFIX_ALL, "D_");
```

то на выходе будут созданы переменные $D_Name и $D_Email, то есть при создании переменных к имени ключа присоединится указанный префикс D_.

7.10. Создание диапазона чисел

Иногда нужно создать диапазон чисел, например, для нумерации записей в выводимой таблице. Создать диапазон чисел можно очень просто:

```
$List = array();
for($i=10; $i<=100; $i++) $List[]=$i;
```

Данный цикл создает список, заполненный целыми числами от 10 до 100 включительно. В PHP можно обойтись значительно проще — одним вызовом функции range(10, 100).

7.11. Небольшой практический пример

Попробуем написать сценарий, выводящий содержимое каталога. Причем сценарий должен не просто выводить содержимое каталога, а в алфавитном порядке и каталоги должны предшествовать файлам (см. листинг 7.2). Мы будем использовать ранее написанную и приведенную в листинге 7.1 функцию `cmp()`, которая сравнивает файлы и каталоги.

Листинг 7.2. Сценарий files.php

```
<?
// Получаем содержимое текущего каталога. Для нашего сценария
текущим будет каталог /var/www/html
// ($DocumentRoot)
exec("ls -1",$Files);

function cmp($el_1,$el_2)
{
if (is_dir($el_1) && !is_dir($el_2))  return -1;
if (!is_dir($el_1) && is_dir($el_2))  return 1;
if($el_1<$el_2) return -1;
   elseif($el_1>$el_2) return 1;
          else return 0;
}

uasort($Files,"cmp");

echo "<html><title>Сценарий files.php</title><body>";
echo "<h1>Содержимое каталога /</h1><p>";

foreach($Files as $f)
 if (is_dir($f))  echo "<br><img src=folder.png>$f";
 else echo "<br><img src=doc.png>$f";

echo "</body></html>";

?>
```

Разберемся, как работает сценарий. Сначала мы формируем список файлов и каталогов. Можно было бы воспользоваться стандартными функциями PHP, но намного проще (в Linux) вызвать команды **ls** с параметром **−1** (вывод в одну колонку). Функция **exec()** вызывает команду **ls** и записывает ее вывод в список `$Files`.

Потом мы вызываем функцию **uasort()**, которая сортирует наш список в нужном нам порядке — сначала она сортирует каталоги в алфавитном порядке, а затем — файлы. В результате мы получаем список, в котором сначала идут каталоги, отсортированные по алфавиту, а затем — файлы (тоже отсортированные по алфавиту).

В цикле **foreach** мы анализируем элемент нашего списка: если он является каталогом, мы выводим рядом с его именем картинку папки, а если файлом — картинку документа (см. рис. 7.1).

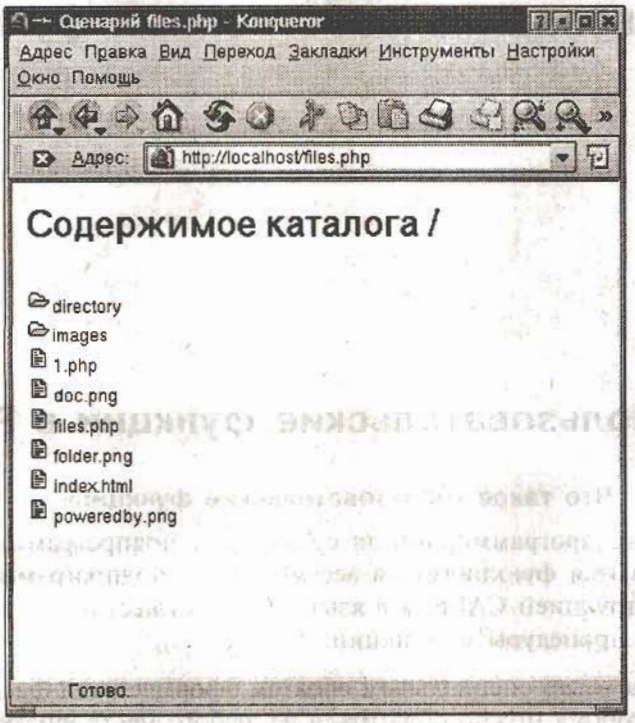


Рис. 7.1. *Результат работы сценария files.php*

Глава 8

Функции пользователя. Создание и использование собственных функций

8.1. Пользовательские функции в PHP

Что такое «пользовательские функции»

В любом языке программирования существуют подпрограммы. В языке C они называются функциями, в ассемблере — подпрограммами и вызываются инструкцией CALL, а в языке Pascal существует даже два вида подпрограмм: процедуры и функции.

Подпрограмма — это специальным образом оформленный фрагмент программы, к которому можно обратиться из любого места внутри программы. Подпрограммы существенно упрощают жизнь программистам: они улучшают читабельность исходного кода, сокращают исходный код (не нужно писать один и тот же фрагмент несколько раз). В языке PHP подпрограммы называются функциями.

Ранее мы рассматривали стандартные функции, предусмотренные самим языком PHP. Однако зачастую стандартных функций не хватает и возникает необходимость создания своей собственной функции, обладающей необходимыми качествами. Такие создаваемые самостоятельно функции и называются пользовательскими. О том, как их создавать и использовать, мы и узнаем в этой главе.

Особенности пользовательских функций в PHP

Как уже упоминалось, в языке PHP подпрограммы называются функциями. Вот особенности использования функций в PHP:

- Вы можете использовать параметры, установленные по умолчанию. Это, в частности, позволяет вызывать одну и ту же функцию с переменным числом параметров.
- Функция может возвращать любой тип.
- Область видимости переменных внутри функции, так же как и в других языках программирования, древовидная.
- Наличие инструкции **return**, как в языке C.
- Можно изменять переменные, переданные в качестве аргумента (как параметр-переменная в Паскале).

Язык PHP предоставляет программисту огромную свободу при создании собственных функций. При программировании на других языках мне очень не хватало параметров по умолчанию, которые позволяют написать функцию с различным числом параметров. В языке C++ появился механизм перезагрузки функций, но параметры по умолчанию — это более гибкое средство. Ведь в C++ для каждой перегруженной функции нам нужно заново писать ее тело. А в языке PHP этого делать не нужно. В этой главе мы обязательно рассмотрим параметры по умолчанию.

Однако у PHP-функций кроме преимуществ есть и недостатки, причем существенные. Первый недостаток связан с объявлением локальной функции. В PHP вы не можете объявить локальную функцию, как это можно делать в других языках. То есть, вы не можете создать функцию внутри другой функции таким образом, чтобы первая (вложенная) функция была видна только во второй функции. В PHP вложенная функция будет видна всей программе, то есть, не будет локальной. Рассмотрим следующий пример:

```
<?
function first() {
 echo "First";
  function Second()
  {
   echo "Second";
  }
}

First();
second();

?>
```

Сценарий напечатает `FirstSecond`. Это означает, что якобы локальная функция доступна во всей программе.

Примечание.

Обратите внимание на имена функций при объявлении (**first** и **Second**) и при вызове (**First** и **second**). Это не опечатка – PHP нечувствителен к регистру имен функций. Вы можете вызвать функцию **first()**, **FIRsT()** или как-то иначе.

Второй недостаток связан с областью видимости функций. Но сначала немного скажем о глобальных и локальных переменных. Глобальные переменные — это переменные, которые видны и доступны всей программе со всеми входящими в нее подпрограммами(функциями). Локальные переменные, определенные внутри подпрограмм, видны и доступны только внутри самой подпрограммы(функции), в которой они определены.

Так вот, в PHP все объявленные и используемые в функции переменные по умолчанию локальны для функции. То есть, по умолчанию вы не можете изменить значение глобальной переменной в теле функции как это можно сделать в других языках программирования. Если вы в функции будете использовать переменную с именем, идентичным имени глобальной переменной, то это никакого отношения к глобальной переменной иметь не будет. Просто в функции будет создана локальная переменная с таким же именем и за пределами функции она видна не будет.

Следующий пример подтверждает данный факт:

```
<?
$i = 10;

function change() {
$i = 5;
echo "i = $i<br>\n";
}
change();
echo "i = $i";
?>
```

Сценарий сначала напечатает 5, а потом — 10.

Тем не менее, в PHP существует возможность избавиться от этого недостатка. Для этого предусмотрено использование специальной инструкции **global**, которая позволяет функции работать с глобальными переменными. Подробнее об этом мы будем говорить в п. 8.6.

8.2. Создание пользовательских функций

Синтаксис задания функций в PHP

Функция может быть объявлена в любой части программы (в пределах операторов <? и ?>), но до места первого ее использования. Другими словами, не нужно никакого предварительного объявления, как в других языка программирования, в частности, в С. Дойдя до определения функции, транслятор проверит корректность определения и оттранслирует определение функции во внутреннее представление, но транслировать код он не станет. Данный подход является самым правильным — зачем транслировать код, который, возможно, никогда не будет использоваться?

Для объявления функций используется такой синтаксис:

```
function Имя (аргумент1[=значение1],...,аргумент1[=значение1])
{
тело функции
}
```

Объявление функции начинается служебным словом function, затем следует имя функции, после имени — список аргументов в скобках. Тело функции заключается в фигурные скобки и может содержать произвольное число операторов.

Требования к именам функций:

- Имя функции может содержать русские буквы, но давать функциям русские имена по понятным причинам не рекомендуется.
- Имя функции не должно содержать пробелов и пробельных символов.
- Имя должно быть уникальным с точностью до регистра, то есть функции F1() и f1() — это одна и та же функция.
- Функциям можно давать такие же имена, как и переменным, только без знака доллара в начале имени.

Так как тип возвращаемого значения может быть любым, его (тип) не указывают вовсе. Для передачи результата работы функции в основную программу используется инструкция **return**. Если функция ничего не возвращает, инструкцию **return** не указывают. Инструкция **return** может возвращать все, что угодно, в том числе и массивы.

Первая функция

Мы уже знаем достаточно о функциях, чтобы написать собственную функцию. Например, следующая функция возвращает массив из 10 случайных чисел (листинг 8.1):

Листинг 8.1. Функция, возвращающая массив из 10 случайных чисел

```
<?
function GetRand() {

$LocalArr = array();
mt_srand(time());

for ($i=0;$i<10;$i++)
$LocalArr [] = mt_rand(0,100);

return $LocalArr;
}

$A = GetRand();
foreach($A as $K=>$v) echo "$v ";

?>
```

В этой функции сначала мы создаем пустой локальный массив `$LocalArr`, затем настраиваем регенератор случайных чисел на новую последовательность (mt_srand), чтобы при каждом новом запуске программы мы получали новую строку. Затем мы заполняем массив случайными числами и передаем его в основную программу, где он присваивается массиву `$A`. Цикл **foreach** выводит значения массива.

8.3. Передача параметров функции

Передача значений

При объявлении функции нужно указать список параметров, которые могут передаваться функции, например:

```
function func($a,$b,$c) { … };
```

При вызове функции `func()` нужно обязательно указать все три параметра, поскольку они являются обязательными. В PHP функции могут обладать необязательными параметрами или параметрами по умолчанию — о них мы поговорим чуть позже.

Передача переменных по ссылке

Согласно традиции, во всех языках программирования есть два вида аргументов функций:

- параметры-значения;
- параметры-переменные.

Функция не может изменить параметр-значение, то есть он доступен функции «только для чтения» — она может его использовать, но не более. В качестве параметра-значения необязательно указывать переменную, можно указать само значение, отсюда название — параметр-значение, например:

```
function func($a)
{
echo "Параметр = $a\n";
}

$b = 3;
func(3);
func($b);
```

Данная функция выведет строку «Параметр = 3» дважды.

В отличие от параметров-значений, параметры-переменные могут быть изменены в процессе работы функции. Тут уже нельзя передавать значение, нужно обязательно передать переменную.

Во всех языках программирования способы объявлений параметров-значений и параметров-переменных отличались друга от друга. Язык PHP не стал исключением. В языке PHP, в отличие от C, нет указателей, поэтому для объявления параметров-переменных используется механизм передачи переменной по ссылке (см. листинг 8.2).

Листинг 8.2. Передача переменной по ссылке

```
<?

$First = $Second = 5;

function f1($f, &$s)
{
echo "Изменяем параметры\n";
$f = 7; $s = 10;
echo "First = $f, Second = $s\n";
}

echo "Значение переменных до вызова функции\n";
echo "First = $First Second = $Second\n";
f1($First, $Second);
echo "First = $First Second = $Second\n";
?>
```

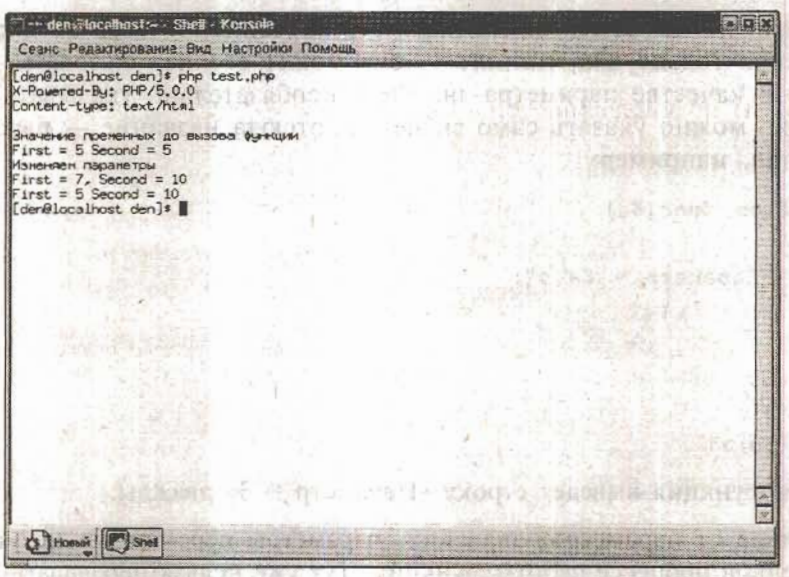


Рис. 8.1. *Вывод программы*

Сначала переменные `$First` и `$Second` равны 5. Затем мы в теле функции изменяем оба параметра — обыкновенный и переданный по ссылке. Как показывает рис. 8.1, внутри функции параметры были изменены, а вот в основной программе отобразилось изменение второй переменной-параметра `$s`, переданного по ссылке — &.

8.4. Параметры по умолчанию

При программировании довольно часто возникала потребность создания функции с переменным числом параметров. Тому есть две причины:

- Параметров слишком много. При этом нет особого смысла каждый раз указывать все параметры.
- Функции должны возвращать значения разных типов в зависимости от набора параметров.

Что же касается второй причины, то в C (в C++ можно перегружать функции) вообще невозможно изменить тип возвращаемого значения. Если вы объявили тип возвращаемого значения как int, то изменить его на double вы не сможете. Например, следующее объявление некорректно:

```
int Multi();
int Multi(int a);
double Multi(&avg);
```

В PHP функция может возвращать любое значение в зависимости от переданных ей параметров.

Параметры по умолчанию описываются так:

```
имя_параметра=значение
```

Например, `function my_func ($x = 10, $y = 5)`

Если параметр не задан при вызове функции, интерпретатор передаст функции значение параметра по умолчанию. Обычно, нужны не только параметры по умолчанию, но еще и самые обыкновенные параметры. В этом случае параметры по умолчанию должны следовать после обыкновенных параметров.

Пример:

```
function GetMax($Arr, $i = 0) // правильное объявление
function GetMax($i = 0, $Arr ) // ошибка
```

Следующий пример (см. листинг 8.3) демонстрирует гибкость языка PHP. Предположим, что нам нужно написать функцию `Multi`, которая использует глобальный массив **Array**. Если функции не переданы параметры, она возвращает максимальное значение, если же ей передано число, он попытается найти в массиве такой элемент. Если элемент с таким значением будет найден, она возвратит его номер, в противном случае он передаст значение −1.

Листинг 8.3. Работаем с функциями

```
<?

$Array = array();

function Multi($a="-999999") {

global $Array; // передаем в функцию глобальный массив Array()
$i = 0;

if ($a =="-999999")
 {
 // нет параметров, был передан параметр по умолчанию
a=-999999
   $max = $Array[0];
   for($i=0;$i<10;$i++)
    if ($max < $Array[$i]) $max = $Array[$i];
    return $max;
 }
 else {
 // поиск элемента
```

```
    for($i=0;$i<10;$i++)
    if ($Array[$i] == $a) return $i;
  }
}


for($i=0;$i<10;$i++) $Array[]=$i+7;
echo Multi()."\n";
echo Multi(10)."\n";

?>
```

Если взяться то же самое реализовать на языке C++, то код увеличится в полтора раза, а тело функции придется переписывать два раза.

8.5. Переменное число параметров в функции

Параметры по умолчанию — это, конечно, очень полезные функции. Но иногда мы точно не знаем, сколько параметров будет передано нашей функции. Специально для такого случая разработчики PHP предусмотрели возможность использования переменного числа параметров. Напишем функцию `print_str`, которая будет выводить переданные ей строки в виде таблицы HTML (листинг 8.4).

Листинг 8.4. Функция print_str

```
<?
function print_str()
{
echo "<table border=1 width=100%>";
echo "<tr><td bgcolor=blue><font color=white>Номер</font></td>";
echo "<td bgcolor=blue><font color=white>Параметр</font></td>";

for ($i=0; $i<func_num_args(); $i++)
 {
 echo "<tr><td>$i</td><td>".func_get_arg($i)."</td></tr>";
 }
echo "</table>";

}

print_str("Первый","Второй","Третий","Четвертый");

?>
```

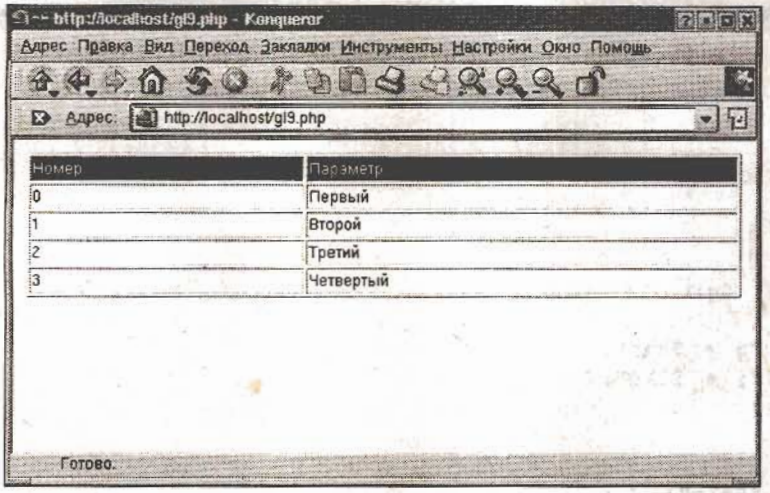


Рис. 8.2. *Созданная таблица параметров*

Основными в этой программе являются функции **func_num_agrs()** и **func_get_arg()**. Первая возвращает число переданных функции аргументов, а вторая — аргумент с указанным номером.

Кроме данных двух функций можно использовать еще одну — **func_get_args()**, которая возвращает список всех аргументов. Использовать ее еще удобнее, чем первые две:

```
foreach(func_get_args() as $a=>$v)
 {
 echo "<tr><td>$a</td><td>$v</td></tr>";
 }
```

И еще один важный момент: обратите внимание на объявление функции — в скобках мы ничего не пишем, как будто мы и не собираемся передавать ей параметры.

8.6. Локальные и глобальные переменные

Мы неоднократно сталкивались в этой книге с локальными и глобальными переменными. Локальные переменные доступны только в функции, а глобальные — доступны во всей программе. Они доступны всем функциям. В листинге 8.5 приведен небольшой пример:

Листинг 8.5. Использование глобальных и локальных переменных

```
<?
$i = 10;
$g = 7;

function f1()
{
$x = 5;
$i = 0;
global $g;

echo "g = $g\n";
echo "i = $i\n";
}

f1();
echo "$x\n";
?>
```

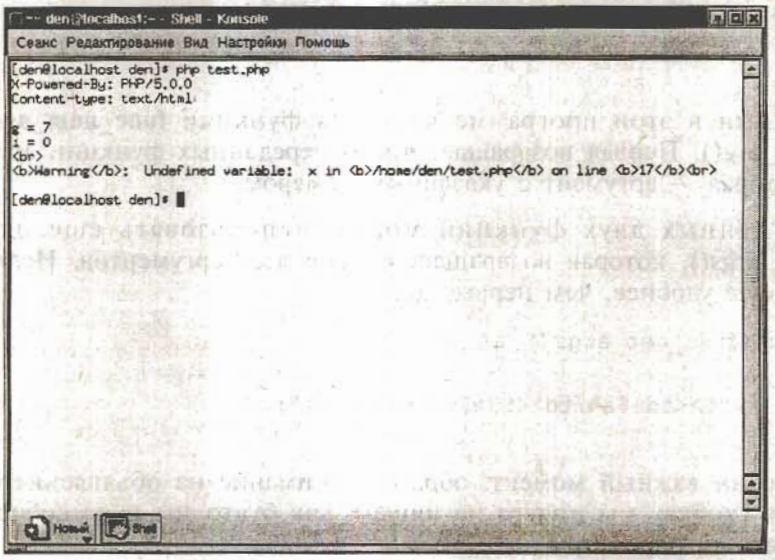


Рис. 8.3. *Локальные и глобальные переменные*

Переменные `$i` и `$g` — глобальные, они доступны во всей программе. Для передачи глобальных переменных в функцию используется инструкция **global**:

```
global список переменных
```

Если мы не выполним эту инструкцию, наша функция не получит доступ к переменной `$g`. Обратите внимание на две переменные `$i`: одна из них глобальная, а другая — локальная (в теле функции).

В функции будет использоваться локальная переменная, однако, если мы объявим переменную глобальной:

```
global $g, $i;
```

то будет использоваться значение глобальной переменной. Попробуйте изменить функцию f1 (см. пред. листинг 8.5) так (см. листинг 8.6):

Листинг 8.6. Некорректное использование глобальных и локальных переменных

```
<?
$i = 10;
$g = 7;

function f1()
{

$i = 5;
echo $i;    // выведет 5
global $i;
echo $i;    // выведет 10

}
f1();

?>
```

Переменная $x является локальной и недоступна в основной программе, поэтому мы получим сообщение о том, что данная переменная не определена (см. рис. 8.3).

Если функция изменит значение глобальной переменной, то значение переменной будет изменено и в основной программе.

Пример:

```
<?
$g = 5;
function f1() { global $g; $g = 10; };
f1();
echo $g; // выведет 10
?>
```

Мы уже говорили о массиве $GLOBALS, поэтому я не стану повторяться, а лишь скажу, что локальные переменные функций не являются элементами данного массива.

8.7. Статические переменные

Кроме локальных и глобальных переменных, в PHP существует еще один тип переменных — статические. Если в теле функции объявлена статическая переменная, то компилятор не будет ее удалять после завершения работы функции. Следующий пример демонстрирует работу со статической переменной:

```
<?

function f1() { static $x; $x++; echo "$x "; }

for ($i=0; $i++<5; ) f1();

?>
```

Следующий сценарий выведет строку 1 2 3 4 5. Если удалить инструкцию static, будет выведена строка 1 1 1 1 1, потому что переменная $x будет удаляться при завершении работы функции и обнуляться при каждом ее вызове. Почему 1 1 1 1 1, а не 0 0 0 0 0? Переменная $x инкрементируется сразу после обнуления, а только потом выводится. Статические переменные не являются элементами $GLOBALS.

8.8. Рекурсия

Рекурсией называется вызов функции самой себя. Такой вызов называется рекурсивным. Рекурсия бывает прямая и непрямая. Рассмотрим пример прямой рекурсии (см. листинг 8.7).

Листинг 8.7. Прямая рекурсия

```
<?
function fact($x)
{
if ($x==0) return 1;
else return $x*fact($x-1);
}
echo fact(7);
?>
```

Как вы уже успели догадаться, функция `fact()` вычисляет факториал x!. Функция `fact()` в своем теле вызывает саму себя — это и есть прямая рекурсия. Непрямая рекурсия возникает, когда первая функция вызывает вторую, а вторая — первую.

Механизм рекурсии одновременно очень полезный, но и очень опасный. При создании рекурсивных функций нужно быть особенно внимательным, чтобы избежать зацикливания. Следующая функция вызовет зацикливание (см. листинг 8.8).

Листинг 8.8. Некорректное использование рекурсии

```
function fact($x)
{
if ($x==0) return 1;
else return $x*fact($x);
}
```

8.9. Полезные примеры

В этом пункте мы рассмотрим два небольших, но полезных примера — функцию `init()`, выполняющую инициализацию переменных и функцию, демонстрирующую работу с массивами.

Инициализация переменных

Начнем с более простой функции `init()`. Довольно часто (если мы пишем «большой» сценарий) возникает потребность в инициализации переменных, то есть присваивании переменным начальных значений. Например, нам нужно после каждой итерации цикла обнулить, скажем, пять-шесть переменных, чтобы не писать:

```
$a=0; $b=0; $c=0; $d=0; $e=0;
```

Гораздо удобнее вызвать функцию init(0,"a","b","c","d","e").

Работать она будет так: мы передаем ей несколько параметров, первый — это значение, а все остальные — имена переменных, которым нужно присвоить это значение. Текст программы приведен в листинге 8.9.

Листинг 8.9. Функция Init – функция инициализации переменных

```
<?
function init()
{
if (func_num_args() == 0) return 0;
else
 {
 $value = func_get_arg(0);          // это значение нужно
                                    // присвоить переданным
                                    // переменным
 for($i=1; $i< func_num_args(); $i++)
```

```
  {
  $var_to_set = func_get_arg($i);  // имя переменной, которую
                                   // нужно инициализировать
  foreach($GLOBALS as $var=>$val)
   if ($var == $var_to_set) $GLOBALS[$var]=$value;
  }
  }
}

$i = 7; $j = 6;

init(1,"i","j");

echo "$i $j\n";
?>
```

Теперь небольшой комментарий к функции `init()`. Если ей не переданы параметры, функция завершает свою работу и возвращает 0. В противном случае она получает первый параметр — `$value`. Этим параметром она будет инициализировать все переменные. Затем она просматривает все параметры, начиная с номера 1 (0 — это значение). Если имя параметра совпадает с одним из ключей массива $GLOBALS, она присваивает значение `$value` элементу массива $GLOBALS с ключом `$var`. Напомню, что в массиве $GLOBALS имена переменных представлены в виде:

```
var = val
```

где `var` — это имя переменной, а `val` — ее значение.

После выполнения функции **init(1,"i","j")** переменным `$i` и `$j` присвоено значение 1.

Параметры-массивы

В этом небольшом пункте мы рассмотрим очень полезный пример, а именно, передачу массива в качестве аргумента функции. Для этого мы напишем свою функцию **array_param()**. Наша функция будет работать не с обыкновенной скалярной переменной, а с целым массивом! При этом, получив массив, она вычисляет номера минимального и максимального элементов, а затем рассчитывает среднее арифметическое элементов массива. В итоге функция возвращает список, состоящий из трех элементов:

- номера максимального элемента;
- номера минимального элемента;
- среднего арифметического.

Данная функция демонстрирует передачу массива в качестве параметра и передачу массива, как результат работы функции, в основную функцию (см. листинг 8.10).

Листинг 8.10. Передача массива в качестве параметра функции

```
<?
function array_param($Arr)
{
$min = $max = $Arr[0];    // сначала Min = Max = первый элемент
$mn_ind = $mx_ind = 0;    // Это индексы минимального и макс.
                          //элементов
$avg = 0;                 // Среднее арифметическое
foreach($Arr as $k=>$v)
 {
 if ($max < $v) { $max=$v; $mx_ind=$k; }; // Вычисление $max
 if ($min > $v) { $min=$v; $mn_ind=$k; }; // Вычисление min
 $avg = $avg + $v;
 }

 $avg = $avg / count($Arr);  // Сумма / количество элементов

 $Res[]=$mx_ind; $Res[]=$mn_ind; $Res[]=$avg;
 return $Res;                // Передача результата
}

for($i=1; $i<11; $i++) $Arr[]=$i;      // Заполнение массива
Arr[]
foreach(array_param($Arr) as $v) echo "$v ";

?>
```

Функция **array_param()** принимает список, обрабатывает его и возвращает также список. Обратите внимание на обработку списка в основной программе. Мы могли бы присвоить возвращаемое функцией значение другому массиву, а потому работать с этим массивом:

```
$Res = array_param($Arr);
foreach($Res as $v) echo "$v "; // Выводим только значения,
                                //индексы нам не нужны
```

Возврат функцией ссылки

В этой главе мы уже рассмотрели механизм передачи параметра по ссылке, позволяющий функции изменять переданные ей параметры-переменные. Однако, это еще не все. В PHP функции могут возвращать ссылки. Рассмотрим следующий пример (см. листинг 8.11).

Листинг 8.11. Возврат ссылок функциями

```
<?
// присваиваем переменной значение 100
$a=100;

// функция, возвращающая ссылку
function &SS ()
{
// переменная $a - глобальная
global $a;
// возвращаем ссылку на переменную $a
return $a;
}

// присваиваем ссылку переменной $b
$b=&SS();
$b=0;
echo $a; //выводит 0
?>
```

Программа выведет 0, а это означает, что $a и $b — синонимы, то есть, если вы измените значение одной переменной, автоматически будет изменено значение второй. Разорвать связь между объектом и ссылкой может оператор **UnSet()**.

8.10. Условно определяемые функции

Уже было сказано, что в PHP функции могут возвращать значения разных типов в зависимости от переданных аргументов. PHP предоставляет разработчику еще большую свободу: в зависимости от внешних факторов (например, от типа операционной системы или ее версии) одна и та же функция может выполнять различные действия. Рассмотрим следующий пример:

```
if ($OS=="Linux")
{
function GetOS() { return "Linux"; }
}
elseif { function GetOS() { return "Windows"; }}

GetOS();
```

Конечно, оператор сравнения можно было бы использовать и внутри функции, но данный пример слишком тривиален, чтобы ощутить все удобство условно определяемых функций. Представьте, что у нас не простенькая функция, возвращающая тип операционной системы, а сложная функция, производящая аутентификацию пользователя в зависимости от многих факторов, в том числе и от типа операционной системы. Вот тогда условно определяемые функции нам очень пригодятся.

Глава 9

Функции для работы со строками

9.1. Операции над строками

Строки в PHP

Строка — это самый универсальный объект в мире PHP. Очень часто строки содержат числа (как целые, так и вещественные). Причем PHP позволяет работать с такими строками, как с обыкновенными числами. Мы можем «упаковать» в строку любой объект, например, массив и работать с ним, как со строкой. Как это сделать и для чего это нужно, вы узнаете в самом последнем пункте этой главы.

То, что PHP лояльно относится к различным типам — это только его заслуга (точнее, разработчиков). Чем же хороши сами строки в PHP? В отличие от C, преимущества PHP-строк заключаются в следующем:

- Строка может содержать любые символы. Напомню, что C-строка не может содержать символы с кодом 0, поскольку первый такой символ трактуется, как конец строки.
- Строка может содержать произвольное количество символов. Длина строки ограничивается только объемом оперативной памяти, что позволяет читать в строку целые файлы и работать с ними, как с обыкновенными строками, например, сравнивать.

Базовыми операциями над строками являются:

- Конкатенация строк (присоединение одной строки к другой).
- Сравнение строк.

Конкатенация строк

В других языках программирования, например, языке Pascal, для конкатенации (присоединения) строк используется символ «+». Однако в PHP использование оператора «+» некорректно: если строки содержат числа, то вместо объединения строк, будет выполнена операция сложения двух чисел. Рассмотрим следующий листинг (листинг 9.1):

Листинг 9.1. Использование оператора «+» для конкатенации строк

```
<?
$a = "20";
$b = "10";
$c = $a + $b;     // число + число
echo "$c\n";    // выведет 30

$d = "Hello";
$c = $a + $d;     // число + строка
echo "$c\n";    // выведет 20 ($a)
?>
```

Как видите, для конкатенации строк оператор «+» не подходит. Вместо него следует использовать оператор «.» (точка) — см. листинг 9.2.

Листинг 9.2. Корректная конкатенация строк

```
<?
$a = "20";
$b = "10";
$c = $a.$b;   // число + число
echo "$c\n";    // выведет 2010

$d = "Hello";
$c = $a.$d;   // число + строка
echo "$c\n";    // выведет 20Hello
?>
```

Сравнение строк

Подобная ситуация с преобразованием типов происходит и при сравнении строк. Операторы сравнения == и != требуют преобразования типов. Поэтому они не должны использоваться для сравнения строк (см. листинг 9.3).

Листинг 9.3. Сравнение строк

```
<?
$a = 0;
$b = 1;
if ($a == "") echo "a - пустая строка";
if ($b == "") echo "b - пустая строка";
?>
```

Данный сценарий сообщит нам, что $a — это пустая строка. Связано это с тем, что пустая строка "", прежде всего, трактуется как 0, а только затем — как «пусто». В PHP операнды сравниваются, как строки, только в том

случае, если оба они — строки. В противном случае они сравниваются, как числа. При этом любая строка, которую интерпретатору не удастся перевести в число (в том числе и пустая строка), будет восприниматься, как 0.

Можно придумывать некоторые обходные пути (заключать числа в кавычки, чтобы они воспринимались, как строки, и т.п.), но при этом очень высока вероятность возникновения ошибок.

Лучше всего при сравнении использовать === (тройной оператор равенства — оператор эквивалентности). Этот оператор всегда позволяет производить корректно сравнение, так как он сравнивает величины и по значению, и по типу.

Примечание.

Для сравнения строк можно также использовать функцию **strval()**, однако она обладает определенными ограничениями и особенностями, что делает ее использование нежелательным. Поэтому данная функция, начиная с PHP 4, уже не используется.

9.2. Базовые функции работы со строками

Функция strlen() возвращает длину строку, то есть количество символов, которое содержит строка.

```
$a = "Hello";
echo strlen($a); // выведет 5
```

Функция strpos() ищет подстроку в строке. Функции нужно передать три параметра:

- Строку, в которой мы будем искать подстроку.
- Подстроку.
- Позицию, начиная с которой мы будем искать. Данный параметр необязательный и по умолчанию равен 0 (начало строки).

Если строка не найдена, функция возвращает `false`, в противном случае — номер позиции, с которой начинается вхождение подстроки в строку:

```
echo strpos("Hello","el"); // выведет 1
```

При работе с данной функцией очень часто начинающие программисты допускают одну и ту же ошибку. Когда я изучал PHP, я не был исключением из данного правила. А ошибка заключается в следующем. Предположим, что нам нужно найти подстроку `ab` в строке `abba`:

```
if(strpos("abba","ab")!=false) echo "Строка ab есть в abba";
```

Очевидно, наш сценарий должен вывести строку "Строка ab есть в abba". Однако он ничего не выводит! Дело в том, что вхождение строки ab в строку abba начинается с нулевой позиции, а 0 и false при использовании оператора сравнения != являются синонимами. Правильнее будет написать так:

```
if(strpos("abba","ab")!==false) echo "Строка ab есть в abba";
```

или

```
if(strpos("abba","ab")===false) echo "Строки ab нет в abba";
```

Функция substr(string $str, int $from, int $length) возвращает заданный участок строки, то есть подстроку. Возвращается подстрока строки $str, начиная с позиции $from длины $length. Если параметр $from будет отрицательным, то отсчет подстроки будет производиться с конца строки, а не с начала. Параметр $length является необязательным.

```
$str = "Book";
echo substr($str,1,1);  // выведет o
echo substr($str,-1,1); // выведет k
```

Функция strcmp($str1, $str2) сравнивает две строки и возвращает следующие значения:

0, если строки равны;

1, если строка $str1 лексикографически больше $str2;

−1, если строка $str1 «меньше» $str2.

Функция strcasecmp(string $str1, string $str2) сравнивает две строки без учета регистра символов, то есть, строки Book, BOOK и book для этой функции равны.

9.3. Замена в строке

Функция str_replace($str1, $str2, $source) заменяет в строке $source (которая в процессе работы функции не изменяется) все вхождения подстроки $str1 на подтроку $str2 и возвращает результат в виде новой строки. Предположим, что нам нужно заменить символы новой строки \n на тег <br>. Это можно сделать так:

```
$str =  str_replace("\n","<br>",$str);
```

или так:

```
$str = nl2br($str).
```

Функция **nl2br()** работает быстрее, чем **str_replace**, однако она не учитывает символов перевода каретки, которые часто встречаются в текстовых файлах Windows.

Функция WordWrap() оказывается очень полезной, если нам нужно отформатировать текст письма перед его оправкой. Данная функция разбивает текст на строки, завершая каждую символом разрыва (по умолчанию \n). В каждой строке должно быть не больше 75 символов (вы можете указать другое количество). Пример использования функции **WordWrap()** приведен в листинге 9.4.

Листинг 9.4. Использование функции WordWrap()

```
<?
$s .= "Очень длинный блок текста, который нужно разбить на части перед отправкой адресату";
echo $s."\n\n";
// разбиваем по 15 символов
$s = WordWrap($s, 15, "\n");
echo $s;
?>
```

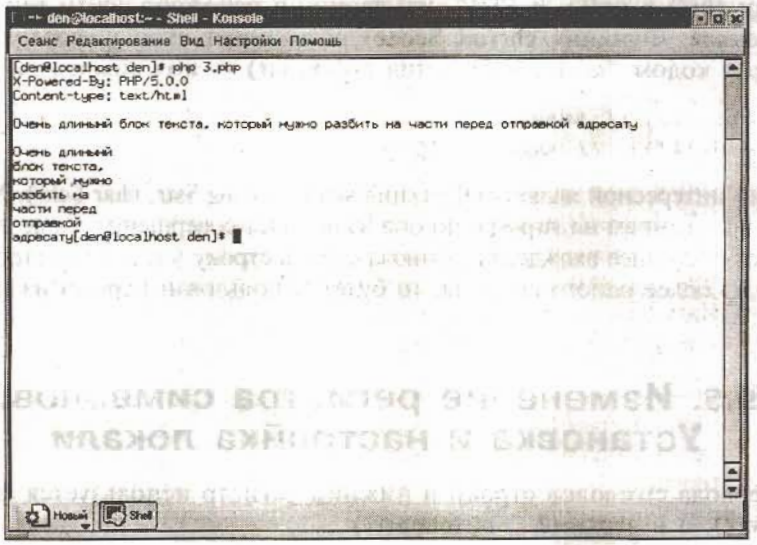


Рис. 9.1. *Результат работы WordWrap*

Примечание.

Функция WordWrap не разбивает блок на строки, как было сказано выше, а лишь вставляет символ разрыва через заданное число символов.

Функция strip_tags(). Очень полезной является функция **strip_tags()**, удаляющая из строки все HTML-теги:

```
$s = strip_tags($s);
```

Вы можете указать, какие теги не стоит удалять. Если вы не хотите удалять теги ссылок и тег заголовка <head>, тогда укажите их при вызове функции:

```
$s = strip_tags($s,"<a><head>");
```

9.4. Функции для работы с отдельными символами

Как и в других языках программирования, в PHP мы можем работать с символами строки отдельно. Обратиться к любому символу строки можно по его индексу:

```
$str = "Book";
echo $str[0]; // выведет B
```

Если вы программировали на языке Pascal, вам должны быть знакомы функции **chr()** и **ord()**. В PHP эти функции работают почти так же, как и в Паскале. Функция **chr(int $code)** возвращает строку, состоящую из символа с кодом `$code`, а функция **ord($char)** — код символа `$char`:

```
echo chr(75); // выведет K
echo ord('A'); // выведет 65 - код буквы A
```

Довольно интересной является функция **strrpos(string $str, char $chr)**. Хотя она и похожа названием на **strpos()**, но она выполняет совершенно иные действия. Она ищет последнее вхождение символа `$chr` в строку `$str`. Если строка `$chr` состоит из более одного символа, то будет использован первый из них.

9.5. Изменение регистра символов. Установка и настройка локали

Для перевода символов строки в нижний регистр используется функция **strtolower()**, а в верхний — **strtoupper()**:

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

При изменении регистра русских букв могут возникнуть проблемы, избежать которых поможет правильная установка локали. Локалью будем называть совокупность локальных настроек системы, таких как формат даты и времени, язык, кодировка.

Настройки локали очень зависят от операционной системы, поэтому если у вас дома все работает нормально, то это совсем не означает, что так и будет на машине вашего хостинг-провайдера.

Для установки локали используется функция **SetLocale()**, которой следует передать два параметра — категорию устанавливаемых параметров и локаль. Категория может быть следующей:

- LC_TYPE — используется для установки локали перевода символов нижний/верхний регистры (как раз то, что нам нужно).
- LC_NUMERIC — используется для установки локали для функции форматирования дробных чисел.
- LC_TIME — помогает настроить форматированный вывод даты и времени.
- LC_ALL — устанавливает все вышеперечисленные режимы.

Мы не будем рассматривать все категории, а остановимся на категории LC_TYPE, которая сейчас нужна нам. Любая локаль, установленная в системе, имеет свое уникальное имя, по которому можно к ней обратиться. Категория LC_TYPE устанавливает локаль для преобразования символов. Имя активизированной локали для категории LC_TYPE содержится в элементе с ключом LANG массива GLOBALS (это переменная окружения LANG). В моем случае — это ru_RU.koi8. Для корректного преобразования символов на *моей* машине нужно выполнить функцию:

```
SetLocale('LC_TYPE',' ru_RU.koi8 ');
```

Если вы работаете под Linux, в большинстве случаев вас устроит данный вызов (если, конечно, вы используете кодировку koi8-r). В большинстве случаев (в современных дистрибутивах), локаль уже настроена до вас, поэтому перед ее изменением с помощью **SetLocale()**, следует сначала убедиться, что параметры настроены неправильно. На моей машине с самого начала все было нормально, поэтому я не изменял локаль.

9.6. Преобразование символов

Функции удаления пробелов

В общем случае абсолютно невозможно предугадать, что может ввести в поле ввода пользователь. Иногда даже сам пользователь не может объяснить, почему он ввел *это*. Однако скрипт, если он создан добротно, должен уметь разобраться в предоставленных ему данных. Или по крайней мере попытаться это сделать. Специально для этого в PHP предусмотрен целый ряд функций.

После приема параметров, прежде чем приступить к их обработке, нужно преобразовать значения параметров к «товарному виду». Прежде всего, нужно удалить лишние пробелы. Функция **trim()** удаляет пробельные символы в начале и в конце строки, например:

```
$s = " string\n";
$s = trim($s); // $s = "string"
```

Напомню, что пробельными символами считаются символы " ",\n,\r,\t. Кроме функции **trim()** можно использовать функции **ltrim()** и **chop()**. Первая удаляет пробельные символы только в начале строки, а вторая — только в ее конце.

URL-кодирование и декодирование

Очень полезными являются функции Url-кодирования и декодирования данных. Так, функция **UrlEncode()** используется для кодирования данных. Кодирование необходимо, если вам нужно передать данные, содержащие специальные символы или символы национальных алфавитов, например, символов русского или украинского языков. Url-кодирование необходимо для передачи данных через Интернет, например, такое кодирование целесообразно, если вы хотите передать русскоязычную информацию в качестве параметра сценария. Если вы хотите передать целый файл, его также желательно закодировать с помощью функции **UrlEncode()**, а потом раскодировать с помощью **UrlDecode()**, чтобы потом не вышло недоразумений. А они могут возникнуть из-за банального отсутствия поддержки сервером 8-битных кодировок.

Вот пример автоматического формирования ссылки <a href>:

```
$Param = "Русский текст";
echo "<a href=http://locahost/
script.php?parametrs=".UrlEncode($Param)."">Click here</a>";
```

В листинге 9.5 приведен пример, демонстрирующий совместную работу функций **UrlEncode()** и **UrlDecode**:

Листинг 9.5. Демонстрация совместной работы функций UrlEncode() и UrlDecode

```
<?
$Param = "Русский текст";
$Param = UrlEncode($Param);
echo "$Param\n";
$Param = UrlDecode($Param);
echo "$Param\n";
?>
```

Строка "Русский текст" (см. рис. 9.2) была преобразована в не совсем понятную последовательность символов, символ пробела был заменен символом +. В таком виде строка готова к передаче по Всемирной паутине. Функция **UrlDecode()** безболезненно преобразовала жуткую после-

довательность символов в исходную строку. Напомню, что кодирования требуют только параметры сценария, если они содержат специальные символы или символы национальных алфавитов.

Кроме функции **UrlEncode()** и **UrlDecode()** вы можете использовать функции **RawUrlEncode()** и **RawUrlDecode()**. Эти функции работают аналогично предыдущим, но не воспринимают символ «+», как пробел.

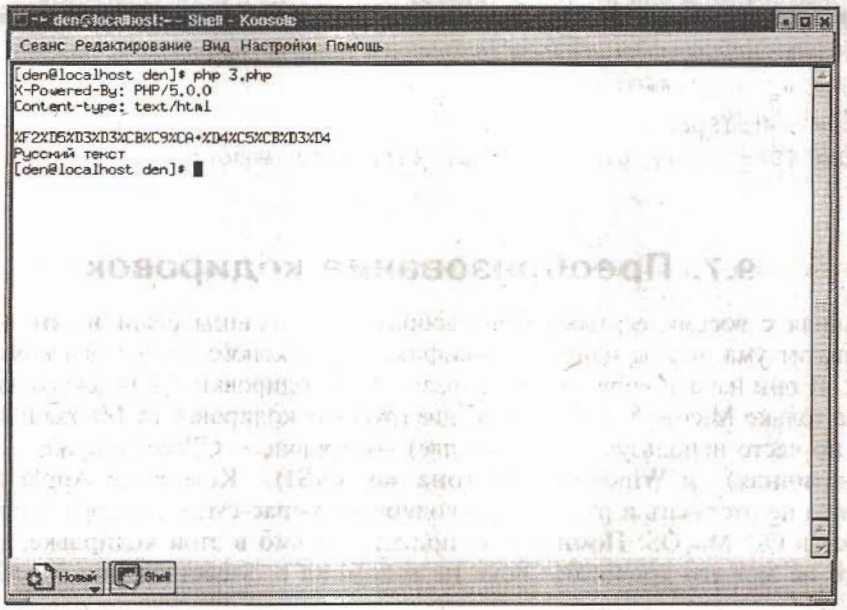


Рис. 9.2. *Функции UrlEncode() и UrlDecode*

Использование слэшей

Вы знаете, что для вывода кавычек или наклонной черты, нужно использовать цитирование, то есть, запись этих символов через наклонную черту:

```
$S = "\\ \"'";
echo $S;
```

Рано или поздно, вам будет нужно добавить перед указанными символами (или же, наоборот, удалить) лишние слэши. Для добавления слэшей используется функция **AddSlashes(string $str)**, а для удаления — **StripSlashes(string $str)**. Обе функции возвращают модифицированную строку и не изменяют исходную.

Замена специальных символов на их HTML-эквиваленты

Функция **HtmlSpecialChars(string $str)** заменяет специальные символы (кавычки, «больше», «меньше» и другие) на их HTML-эквиваленты. Делается это для того, чтобы они на Web-странице выглядели самими собой.

Например при использовании функции **HtmlSpecialChars**, знак «больше» в коде будет заменен строкой &gt, а кавычки — &quot. Пример использования **HtmlSpecialChars** приведен в листинге 9.6.

Листинг 9.6. Пример использования HtmlSpecialChars

```
<?
$S = "& < > \"";
$S =  HtmlSpecialChars($S);
echo $S;      // выведет &amp; &lt; &gt; &quot;
?>
```

9.7. Преобразование кодировок

Начиная с восьмидесятых годов «добрые» американцы решили, что у нас не хватит ума, чтобы написать русификатор знакового генератора компьютера. И они начали «придумывать» различные кодировки для русского языка. Одна только Microsoft «придумала» две (русских кодировок от Mirsosoft больше, но часто используются только две) кодировки — CP866 (она же — альтернативная) и Windows-1251 (она же ANSI). Компания Apple тоже решила не отставать и разработала кодировку x-mac-cyrillic, которая используется в ОС MacOS. Помню, мне пришло письмо в этой кодировке, так я долго не мог его прочитать, пока не вспомнил о существовании MacOS.

В добавок ко всему ISO выпустила свою кодировку для русского языка — ISO8859-5. К сожалению, американцы забыли, что у нас есть своя родная кодировка — KOI8-R. Используя эту кодировку в DOS, дядя Билл, в свое время, действительно облегчил жизнь русскому народу. А так получается следующее — у меня Linux, у соседа — Windows, а у кого-то вообще DOS установлен. И у всех поддерживает русский язык до тех пор, пока мы не станем обмениваться файлами. Конечно, сегодня это выглядит не так актуально — почти все почтовые клиенты поддерживают все вышеперечисленные кодировки, а также существует масса программ, выполняющих преобразование кодировок.

В PHP есть одна очень полезная функция, преобразующая русский текст из одной кодировки в другую — **convert_cyr_string()**, которой нужно передать три параметра:

- Исходную строку.
- Исходную кодировку.
- Требуемую кодировку.

Последние два параметра могут принимать следующие значения:

k — KOI8-R;
w — windows-1251;
d или **a** — x-cp866;
m — x-mac-cyrillic;
i — iso8859-5.

Пример использования функции **convert_cyr_string()** приведен в листинге 9.7.

Листинг 9.7. Пример использования функции convert_cyr_string()

```
<?
// Преобразование слова "Привет" из кодировки koi8-r в windows-
1251
echo convert_cyr_string("Привет","k","w");      // выведет "оПХ-
БЕР"
?>
```

9.8. Специальные функции

Хэш-функции

В этом пункте мы рассмотрим две функции — **md5()** и **crc32()**.

Функция **md5()** кодирует строку, используя алгоритм MD5 (Message Digest Algorithm). Если быть предельно точным, то функция **md5()** возвращает хэш-код переданной строки.

Чем же хорош алгоритм MD5? Вероятность того, что две разные строки будут иметь одинаковый хэш-код, стремится к нулю, а это позволяет эффективно использовать данную функцию для шифрования паролей. А тем, что зашифрованное с его помощью сообщение невозможно расшифровать — для алгоритма MD5 не существует алгоритма дешифровки, как для других алгоритмов. Для проверки подлинности пароля нужно зашифровать пароль заново и сравнить зашифрованные строки: если они равны, значит пароль правильный (см. листинг 9.8).

Листинг 9.8. Использование хэширования

```
<?
$a = "my_password";
$b = "my_password";
$c = "password";

if (md5($a)===md5($b)) echo "1: пароль правильный";
if (md5($a)===md5($c)) echo "2: пароль правильный";
?>
```

Сценарий выведет только первое сообщение, поскольку строки (значит, и их хэш-коды) равны. Вообще-то использовать эту функцию нужно немного не так:

- Пароль не должен храниться в сценарии, тем более в открытом виде. Пароль нужно хранить в зашифрованном виде в базе данных или в каком-нибудь файле на диске.
- Введенный пользователем пароль должен зашифровываться и сравниваться уже с зашифрованной строкой, которая прочитана из файла или БД.

Кроме функции **md5()** вы можете использовать функцию **crypt()**, которая реализует алгоритм DES, но данный алгоритм не так эффективен, как MD5, поэтому его нельзя использовать для серьезного проекта.

Функция **crc32()** используется для вычисления 32-битной контрольной суммы строки. Данная функция может вам пригодиться для регистрации какого-нибудь программного продукта через Internet, хотя я порекомендовал бы написать собственную функцию для этой цели, потому что **crc32()** — стандартная функция, и ее может использовать кто угодно.

```
<?
$SN = "012103343";
echo crc32($SN);
?>
```

Функция Serialize()

Перед нами поставлена практическая задача: мы разрабатываем систему контроля знаний (попросту говоря, тест), и нам нужно предусмотреть продолжение теста с того места, где он был прерван.

Мы уже знакомы с технологией Cookies, поэтому знаем, как сохранить текущий номер вопроса, начиная с которого тест будет продолжен. А как запомнить номера вопросов (массив), которые были сгенерированы для студента? Сохранить массив в Cookies мы не можем, зато можно сохранить строку. Функция **Serialize()** позволяет преобразовать любой объект в строку, ею мы и воспользуемся для сохранения нашего массива в Cookies. Обратите внимание: строка, полученная в результате работы serialize(), имеет специальный формат. Чтобы восстановить из этой строки объект обратно, используйте функцию unserialize (). Вернемся к нашему тесту:

- Мы генерируем список номеров вопросов, которые сериализируем и помещаем в Cookies.
- В программе работаем с обыкновенным массивом, а в случае разрыва соединения (при повторном запуске) получаем массив из Cookies.

Как это реализуется программно показано в листинге 9.8.

Листинг 9.8. Демонстрация использования функции Serialize()

```
<?

if (!isset($Questions))
{
// Cookies не установлены - первый запуск

for($i=0;$i<10;$i++) $Arr[]=$i;
// Преобразуем массив в строку
$Serialized_arr = serialize($Arr);
// Устанавливаем Cookies
setcookie("Questions",$Serialized_arr,time()+3600);

unset($Arr); // массив $Arr больше не существует
}
else
{
// Cookies установлены - "разворачиваем" массив
$Arr = unserialize($Questions);
foreach($Arr as $v) echo "$v ";
}
?>
```

Глава 10

Полезные стандартные функции

10.1. Работа с датой и временем

Практически ни один серьезный проект не обходится без работы с датой и временем, поэтому не уделить внимание функции **date()** просто невозможно. Функция **date (string $format [, int $timestamp])** использует два параметра:

- Формат даты/времени (строка).
- Сама дата (или время) — так называемый timestamp.

Функция выводит отформатированную дату на основе второго параметра или на основе текущей даты, если параметр **$timestamp** не указан. Строка формата может содержать обычный текст, а также модификаторы (см. табл. 10.1), которые при выводе заменяются на соответствующие значения.

Модификаторы формата функции Date() *Таблица 10.1*

Модификатор	Описание
U	Количество секунд, прошедших с полуночи 1 января 1970 года (с момента основания Unix)
Y	Год (4 цифры)
y	Год (2 цифры)
z	Номер дня от начала года (от «0» до «365»)
F	Полное английское название месяца («December»)
m	Номер месяца (2 цифры: от «01» до «12»)
n	Номер месяца без предваряющего нуля: от «1» до «12»
M	Трехсимвольная английская аббревиатура месяца («Dec»)
d	Номер дня в месяце (2 цифры: от «01» до «31»)
j	Номер дня в месяце без предваряющего нуля: от «1» до «31»
l	Текстовое название дня недели по-английски, например, «Monday»
w	Номер дня недели («0» – Воскресенье, «1» – Понедельник и т.д.)
D	Трехбуквенная английская аббревиатура дня недели (например, «Mon»)
A	«До» или «после» полудня: «AM» или «PM»
a	«До или «после» полудня «am» или «pm»
H	Часы (24-часовой формат) от «00» до «23»
h	Часы (12-часовой формат) от «01» до «12»
i	Минуты, от «00» до «59»
s	Секунды, от «00» до «59»

Примеры использования функции **date()**:

```
echo date(''Сегодня d.m.Y'');
echo date(''Дата создания файла
index.html'',filectime(''index.html''));
```

При обработке введенных параметров очень полезной окажется функция **checkdate()**, которая проверяет корректность введенной даты. Функции нужно передать три параметра — месяц, день и год (именно в такой последовательности). Дата проверяется так:

- Год должен быть не меньше 1900 (и не больше 32767).
- Месяц должен быть в диапазоне 1...12.
- Число дней должно соответствовать месяцу и году (функция учитывает, високосный год или нет).

В PHP для работы с датой и временем есть дополнительные функции, но рассмотренных функций вполне достаточно для работы.

10.2. Генератор случайных чисел. Вывод баннеров случайным образом

Чаще всего генерация случайных чисел нужна для отображения баннеров или криптографии. Функции для генерации случайных чисел — это функции **mt_rand()** и **mt_srand()**.

Функция **mt_rand(int $from, int $to=RAND_MAX)** возвращает случайное число в диапазоне от `$from` до `$to`. Если второй параметр не задан, используется значение по умолчанию — константа RAND_MAX. Значение данной константы можно узнать с помощью функции **mt_getrandmax()**.

В качестве примера сгенерируем 10 случайных чисел от 0 до 100:

```
for ($i=0; $i<10; i++) echo ''mt_rand(0, 100) '';
```

Сценарий выведет 10 случайных чисел. Запустите сценарий еще раз. И еще раз. Как видите, каждый раз сценарий выводит одну и ту же последовательность. Для настройки генератора случайных чисел на новую последовательность используется функция **mt_srand()**. С ней мы уже сталкивались, когда изучали перемешивание массивов в п. 7.6.

Функции **mt_srand()** нужно передать всего лишь один параметр, на основании которого она настроит генератор на новую последовательность. В качестве параметра можно передать значение функции **time()**, которое возвращает количество секунд, прошедшее с 1 января 1970 года:

```
mt_srand(time());
```

В начале этого пункта я неосторожно обмолвился о баннерах. В одной из своих книг я описал, как с помощью SQUID (прокси-сервер) «вырезать» все баннеры. В этой книге, чтобы уравнять шансы, я напишу сценарий, который выбирает случайный баннер и вставляет его в Web-страницу.

Предположим, что у нас есть файл `banner.txt`, в котором содержатся имена картинок-баннеров, которые нужно вставить в страницу. Подробнее о файлах мы поговорим в следующей главе, а сейчас не будем останавливаться на функциях для работы с файлами.

```
$Banners = file(''banner.txt'');
$k = mt_rand(0,99);
mt_srand(time() + (double)microtime()*$k);
$i = mt_rand(0, count($Banner));

echo ''<IMG SRC=$Banners[$i]''.'' ALT=Banner>'';
```

Сначала мы получаем список строк, содержащихся в файле `banner.txt`. Затем настраиваем генератор на новую последовательность, получаем случайный индекс списка `$Banners - $i` и выводим тег IMG. В качестве имени файла картинки (SRC) будет элемент списка `$Banners`, выбранный случайным образом.

10.3. Математические функции

Помните, мы писали функции для поиска минимального (или максимального) элемента массива? Зря мы это делали — в PHP есть стандартные функции — **min()** и **max()**.

Функция **max()** возвращает максимальный из переданных ей n аргументов. Например, **echo max**(`7,10,3,1,$a,$b`) выведет 10, если значения `$a` и `$b` меньше или равны 10.

Если функции передан один параметр, то он должен быть массивом, в котором будет произведен поиск максимального элемента.

Функция **min()** работает аналогично функции **max()**, но ищет минимальный аргумент (или элемент массива).

Математические функции PHP в особых примерах и объяснениях не нуждаются, поэтому удобнее всего представить их в виде таблицы (см. табл 10.2).

Надеюсь, что эти функции вам понадобятся при программировании сценариев.

Математические функции PHP *Таблица 10.2*

Функция	Описание
acos(float $x)	Вычисляет арккосинус аргумента $x
asin(float $x)	Вычисляет арксинус аргумента $x
atan(float $x)	Вычисляет арктангенс аргумента $x
atan2(float $x, float $y)	Вычисляет арктангенс числа $x/$y. Результат – в радианах
sin(float $x)	Вычисляет синус аргумента $x. Аргумент задается в радианах
cos(float $x)	Вычисляет косинус аргумента $x. Аргумент задается в радианах
tan(float $x)	Вычисляет тангенс аргумента $x. Аргумент задается в радианах
pi()	Возвращает число Пи. Вместо функции pi() удобнее использовать константу M_PI
sqrt(float $x)	Возвращает квадратный корень $x. Напомню, что $x должен быть положительным
log(float $x)	Возвращает натуральный логарифм $x
exp(float $x)	Возвращает экспоненту (2,71828182845) в степени $x
pow(float $x, $a)	Возвращает $x в степени $a

Результат, возвращаемый всеми перечисленными в табл. 10.2 функциями, имеет тип float. Только число pi возвращаяется с типом double.

10.4. Преобразования систем счисления

Данные функции также не очень нужны при программировании сценариев, но, возможно, они вам пригодятся при работе на низком уровне с бинарными файлами.

Самая универсальная функция для преобразования кодировок — это функция **base_convert()**. Ей нужно передать три параметра:

- Заданное в виде строки число, которое нужно преобразовать.
- Основание системы счисления, в которой число находится в данный момент.
- Основание системы счисления, в которую нужно преобразовать число.

Последние два параметра могут принимать значения от 2 до 36 включительно. Первый параметр может содержать, как цифры, так и латинские буквы. При этом цифры обозначают сами себя, буква «a» обозначает 10, буква «b» обозначает 11, и так далее до буквы «z».

Пример:

```
echo base_convert(''8'',10,2);        // выведет 1000
echo base_convert(''F'',16,10);   // выведет 15
echo base_convert(''F'',16,2);        // выведет 1111
```

Следующие две функции используются для преобразования числа из двоичной системы в десятичную, и наоборот:

bindec() — переводит двоичное число в десятичное;
decbin() — переводит десятичное число в двоичное.

Пример:

```
echo bindec(''1111'');   // выведет 15
echo decbin(''15'');   // выведет 1111
```

Существует функции для преобразования чисел из шестнадцатеричной и восьмеричной систем в десятичную, и обратно:

hexdec() — преобразует число из шестнадцатеричной системы в десятичную;
dechex() — преобразует число из десятичной системы в шестнадцатеричную;
octdec() — преобразует число из восьмеричной системы в десятичную;
decoct() — преобразует число из десятичной системы в восьмеричную.

10.5. Запуск внешних программ

Большинство задач не требуют вызова внешних программ, поскольку без этого можно обойтись, используя стандартные функции PHP. Однако иногда удобнее вызвать внешнюю программу, чем использовать функции PHP. Самый простой пример — вывод оглавления каталога. Можно использовать функции PHP чтения каталога и его текущего элемента, которые будут рассматриваться в следующей главе, а можно написать всего лишь два оператора:

```
$dir = 'ls';        // вывод оглавления текущего каталога
echo ''<pre> $dir </pre>;
```

Для вызова внешней программы (в данном случае этот Linux-команда ls, которая выводит содержимое каталога) заключите команду вызова в обратные апострофы.

Примечание.

Оператор $dir = 'ls' не сработает, если включен так называемый безопасный режим. Для его отключения установите параметр **Safe_Mode=Off** в файле /etc/php.ini.

Команда **ls** слишком тривиальна. Рассмотрим примеры более полезных команд (в Linux-Unix):

ps –ax — выводит список всех запущенных процессов;

ps −ax | grep httpd — информация о процессе httpd (сервер Apache);

ps −u den — выводит список всех процессов, владельцем которых является пользователь den;

free — информация о загрузке памяти;

vdf — информация о загрузке дисковой памяти.

Все эти команды будут работать в ОС Linux (или другой Unix-подобной ОС).

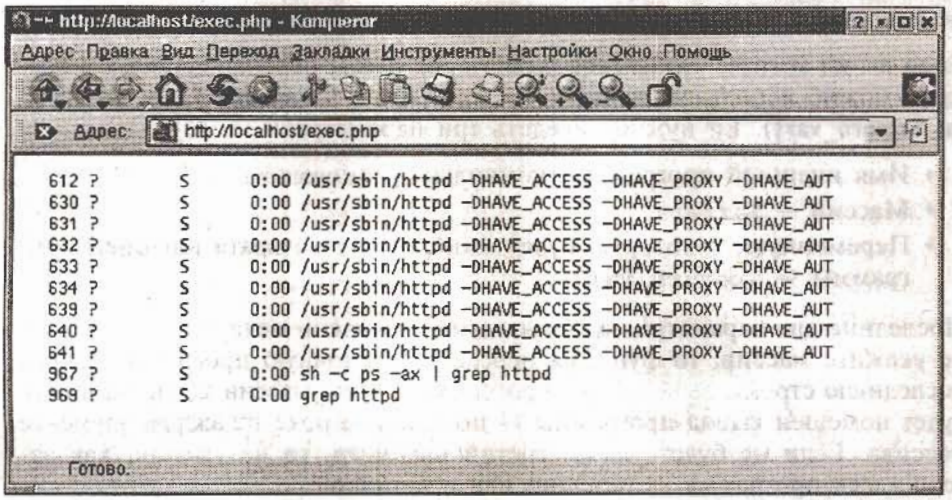


Рис. 10.1. *Вывод команды ps -ax | grep httpd*

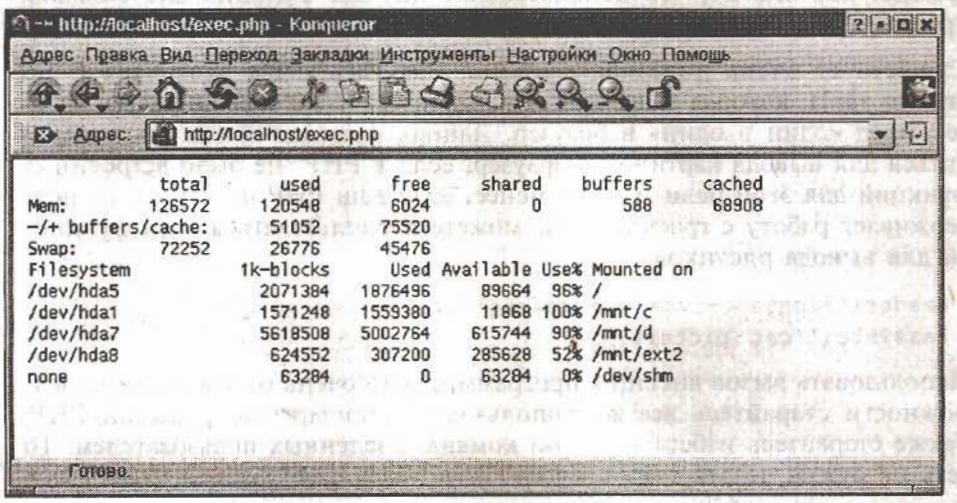


Рис. 10.2. *Вывод команд free; df*

Теперь рассмотрим стандартные функции PHP по запуску внешних программ. Если вы хотите, чтобы сразу вывод программы был перенаправлен в браузер, воспользуйтесь функцией **system()**:

```
system(string $command [, int $return_var]);
```

В качестве первого параметра этой функции передается имя запускаемой программы (команды). Второй параметр не обязаателен и обычно не задается. Однако, если он задан (это обязательно должна быть переменная, а не константа), то в него помещается код возврата вызванной программы.

Если вы не хотите, чтобы весь вывод программы перенаправлялся в браузер, можно использовать функцию **exec(string $command [, list $array][, int $return_var])**. Ей нужно передать три параметра:

- Имя внешней программы (команды) — `$command`;
- Массив — `$array`;
- Переменную, в которую будет записан код возврата внешней программы — `$return_var`.

Последние два параметра не обязательны, но очень желательны. Если вы не укажите массив, то функция передаст в основную программу только последнюю строку, выведенную программой. Если массив задан, то в него будет помещен вывод программы — по одной строке в каждом элементе массива. Если не будет указан третий параметр, то не узнаете, как завершилась программа — успешно или с ошибкой.

Стоит отменить, что функция **exec()** (как и **system()**) дожидается завершения программы и только потом уже передает управление основной программе. Это значит, что использование данных функций для больших программ недопустимо, иначе пользователь подумает, что сценарий завис.

Существует очень полезная функция **PassThru(string $command [, int $return_var])**, которая запускает команду и сразу же перенаправляет весь ее вывод «один в один» в браузер. Данная функция могла бы понадобиться для вывода картинок в браузер, если в PHP не было встроенных функций для этой цели. Тем не менее, если ваш PHP почему-то не поддерживает работу с графикой, вы можете воспользоваться этой функцией для вывода рисунков:

```
Header(''Content-type: image/png'');
PassThru(''cat picture.png'');
```

Использовать вызов внешних программ нужно очень осторожно. По возможности старайтесь все же использовать стандартные функции PHP. Также старайтесь избегать вызова команд, введенных пользователем. То есть не нужно делать так:

```
system($from_user)
```

Здесь предполагается, что переменная `$from_user` была получена из формы или Cookies. Ведь пользователь может ввести все, что угодно. Например, команда **cat /etc/passwd** выведет файл `passwd`, в котором содержатся данные об учетных записях пользователей системы (см. рис. 11.3). Конечно, при правильно настроенных правах доступа пользователь не получит доступ к файлу `/etc/shadow`, в котором хранятся пароли, но и знать ему имена всех зарегистрированных в вашей системе пользователей тоже не нужно.

Рис. 10.3. *Вывод /etc/passwd*

Если нужно все-таки обеспечить выполнение команд пользователя, например, для смены каталога внутри домашнего каталога пользователя, используйте функцию **escapeshellcmd()**, которая защитит все специальные символы слешами. Например, вы хотите обеспечить смену каталога пользователя:

```
system(''cd $dir'');
```

Переменная `$dir` пришла от пользователя. Если пользователь введет следующее значение "/; rm -R *", то он сможет удалить все файлы вашего сервера, если, конечно, сервер неправильно настроен.

Примечание.

Наверное, вас заинтересовал один вполне уместный вопрос: а как пользователь может изменить каталог (то есть выполнить команду *cd /home/usr/work*), если все слеши блокируются? А для этого используется функция **chdir()**, которую мы рассмотрим позже.

Некоторые администраторы запускают сервер Apache от имени пользователя root. Это неправильно. В этом случае уж точно любой желающий сможет «убить» ваш сервер. Надо запускать сервер от имени пользователя apache или бесправного пользователя nobody. В этом случае пользователь не может выполнить никаких разрушающих действий. Если администратором являетесь не вы, и проверить от чьего имени запускается Apache вы тоже не можете, используйте функцию **escapeshellcmd()** — это, прежде всего, в ваших интересах. Ведь восстановить систему — это дело двух-трех часов, а восстановить данные пользователей (ваши в том числе), порой вообще невозможно.

На рис. 10.4 изображен результат работы функции **escapeshellcmd()** — первая строка до вызова функции, а вторая — после. Специальные символы были предварены защитными слэшами — даже если пользователь введет такую разрушающую команду, у него ничего не выйдет.

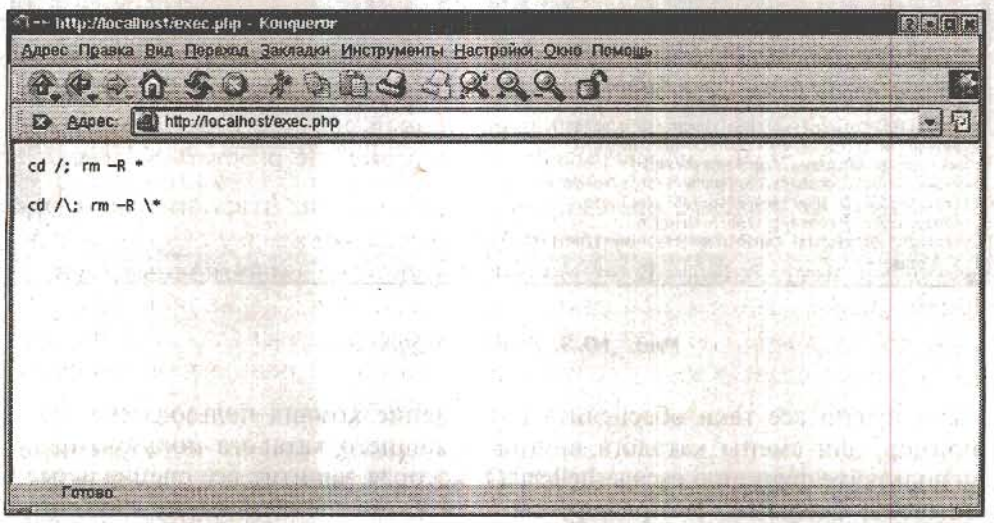


Рис. 10.4. *Результат работы escapeshellcmd()*

Глава 11

Работа с файлами и каталогами

11.1. Общие замечания

Прежде чем приступить к работе с файлами, нужно рассмотреть особенности работы с файлами при программировании на PHP и вообще под Unix. Даже если вы сейчас программируете под Windows, вам просто необходимо знать эти тонкости, поскольку у хостинг-провайдера обычно установлена Unix-подобная операционная система и ваша программа, нормально работающая под Windows, может не работать в Unix.

В отличие от Windows, операционная система Linux чувствительна к регистру имени файла, это означает, что файлы `A.txt` и `a.txt` — совершенно разные файлы. В операционной системе Linux полный путь к файлу записывается с помощью прямых слэшей (/), а не обратных (\), как в Windows. Но об этом вы можете не беспокоиться: для интерпретатора нет разницы, какие слэши вы используете — лишь бы только файл существовал, и путь к нему был указан правильно.

Интерпретатор PHP позволяет использовать не только обыкновенные файлы, но еще и целые сетевые соединения, например, вы можете открыть файл `http://dkws.narod.ru/index.html`, и работать с ним, как с файлом, то есть прочитать, модифицировать и вывести в браузер. Точно также вы можете работать и с протоколом FTP — просто укажите префикс протокола `ftp://`. Если у вас есть права на запись удаленных файлов, вы можете даже модифицировать эти файлы.

Самое главное отличие Windows-файлов от Unix-файлов проявляется при работе с файлом в текстовом режиме, когда данные читаются и записываются построчно. В Windows для обозначения конца строки используются два символа — `\r` (возврат каретки) и `\n` (перевод строки). В Unix для этой цели используется только один символ — `\n`. Вот как один и тот же файл будет выглядеть в разных операционных системах:

```
В Linux     В Windows     В MacOS
Line1\n     Line1\r\n     Line1\n\r
Line2\n     Line2\r\n     Line2\n\r
Line3       Line3         Line3
```

Интерпретатор PHP работает в так называемом режиме трансляции символов новой строки. Если вы работаете в Linux и открываете файл в формате Windows, последовательность символов `\r\n` будет преобразована в один символ `\n`. Наоборот, если вы работаете в Windows и открываете Unix-файл, символ `\n` будет предварен символом `\r`. То же самое происходит при записи файлов. Режим *трансляции символов* новой строки еще называют *текстовым режимом* работы. Если отключить этот режим, интерпретатор будет работать в *бинарном режиме*.

В бинарном режиме никакой трансляции символов производиться не будет, поэтому вы можете свободно работать с нетекстовыми файлами, например, картинками, архивами и другими двоичными файлами.

Для чего я все это говорил — ведь для программиста преобразование символов происходит незаметно? Дело в том, что при чтении символы `\r` удаляются, а при записи на диск — добавляются, а следовательно, изменяется размер файла. Соответственно, изменяется и значение, возвращаемое функцией **filesize()**, которая выдает размер файла. Это значение будет не соответствовать размеру реально прочитанных данных. Если вы используете текстовый режим, то данный момент нужно учитывать.

В бинарном режиме вы можете делать с файлом все, что захочется. Например, можно полностью прочитать файл в строку, при этом данные никак не повредятся. Представьте такую ситуацию: вы открываете картинку в формате GIF в текстовом режиме. Это хоть и маловероятная процедура, но нам она позволит продемонстрировать как подчас бывает необходим бинарный режим. Итак, вы открыли GIF-файл в текстовом режиме. Потом вы записываете содержимое файла в одну большую строку. Так как открытие файла произошло в текстовом режиме, из нашей строки «выпали» некоторые символы `\r`. Во-первых, длина нашей строки будет меньше (на количество символов `\r`) размера файла. А во-вторых, если мы потом попытаемся вывести эту строку (не забывайте — это же GIF-файл, но уже без символов `\r`) в браузер, то изображение будет искажено: ведь мы «вырезали» некоторые фрагменты изображения. Скорее всего, пользователь вообще ничего не увидит в окне браузера, потому что последний не сможет распознать формат GIF.

Выбор режима осуществляется при открытии файла — об этом мы поговорим в следующем пункте.

11.2. Основные операции с файлами

Как и в любом языке программирования, работа с файлом осуществляется в три этапа:

- Открытие файла — при этом возвращается целое число — идентификатор (дескриптор) файла. Дальнейшее обращение к файлу осуществляется через дескриптор, а не по имени.
- Обработка файла (чтение или запись).
- Закрытие файла.

11.2.1. Открытие файла

Функция fopen() и открытие файла

Функция **fopen($filename, $mode)** открывает файл (или URL) и возвращает его идентификатор. Данной функции нужно передать два параметра — имя файла (`$filename`) и режим (`$mode`), в котором необходимо открыть файл.

Примечание.

На самом деле, функция **fopen()** имеет три аргумента, но третий аргумент обычно не используется — он задан по умолчанию. Последний аргумент, который по умолчанию равен false, говорит, что если задано относительное имя файла (а не полный путь к файлу), то файл также нужно искать в списке путей, которые используются инструкциями **include** и **require**.

Имя файла — это обыкновенная строка. Можно указывать, как полный, так и относительный путь к файлу. При указании относительного пути используют каталоги «..» (две точки) — родительский каталог и «.» (точка) — текущий каталог.

`./array.txt` — файл находится в текущем каталоге;
`../array.txt` — файл в родительском каталоге;
`../command/array.txt` — файл в подкаталоге command родительского каталога.

Доступные режимы открытия файла приведены в табл. 11.1.

Режим открытия файла *Таблица 11.1*

Режим	Описание
r	Режим "только чтение". В этом режиме вы можете только читать файл, но не изменять его. После открытия указатель файла устанавливается на его начало (первый байт). В случае, если файл не существует, функция возвращает false
r+	Файл открывается в режиме "чтение/запись", то есть вы можете не только читать, но и записывать данные в файл. После открытия файла указатель устанавливается на его начало. Как и в режиме r, если файл не существует, функция возвращает false. ВАЖНО: если в момент записи указатель файла находится где-то в середине файла, то записываемые данные будут записаны поверх уже имеющихся, а не будут "вставлены" между ними

Режим открытия файла *Таблица 11.1 (продолжение)*

Режим	Описание
w	Будет создан новый пустой файл. Если на диске уже есть файл с таким именем, он будет перезаписан
w+	Напоминает режим r+, но со следующими отличиями: — если файл не существует, он будет создан; — если файл существует, он будет перезаписан. Вы можете как читать, так и записывать данные в файл
a	Режим дополнения (append). Файл открывается в режиме записи, а указатель файла сдвигается в конец файла. Данные будут добавлены в конец файла. Если файл не существует, функция возвращает false
a+	Самый универсальный режим. Файл открывается в режиме чтения и записи, при этом его содержимое не удаляется, а указатель файла перемещается на конец файла. Если файл не существует, он будет создан
b, t	После любого из вышеперечисленных режимов может следовать один модификатор — b или t. Модификатор b указывает на то, что файл будет открыт в бинарном режиме, а t — в текстовом. Если модификатор не указан, по умолчанию файл открывается в бинарном режиме

Примечание.

Почему-то о модификаторе режима t ничего не сказано в документации, но мои эксперименты показывают, что он работает. Если не верите, можете убедиться в этом сами.

Конструкция or die()

Функция **fopen()** возвращает `false`, если ей не удалось открыть файл: например, указанный файл не существует. Нам нужно проанализировать результат функции **fopen()** и, если он равен `false`, вывести соответствующее сообщение и завершить работу сценария. Сделать все это позволяет конструкция **or die($err_message)**. Синтаксис данной конструкции выглядит следующим образом:

```
<вызов_функции> or die("сообщение об ошибке");
```

Если вызов функции неуспешен (функция возвратила **false**), будет выведено сообщение об ошибке и работа сценария будет завершена. Преимущество конструкции **or die()** заключается в том, что она, во-первых, позволяет писать более компактный код, а во-вторых, позволяет использовать себя вместе с присваиванием, так как приоритет оператора **or** ниже, чем оператора =.

Вот примеры открытия файлов:

```
// Открываем файл /home/den/counter.txt в бинарном режиме
// "только чтение"
$f = fopen("/home/den/counter.txt","r") or die("Ошибка при от-
крытии файла\n");
// Открываем файл http://dkws.narod.ru/index.html в текстовом
```

```
// режиме "только чтение"
$f = fopen("http://dkws.narod.ru/index.html","rt") or
die("Ошибка при открытии файла\n");

// Открываем FTP-соединение в бинарном режиме для записи (с
// указанием логина и пароля)
$f = fopen("ftp://user:123456@ftp.narod.ru","r") or die("Ошибка
при открытии файла\n");

// Открываем журнал для добавления сообщения в конец файла
// (текстовый режим)
$f = fopen("/home/den/counter.log","a+t") or die("Ошибка при
открытии файла\n");
```

Примечание.

Я рекомендую всегда использовать прямые слеши для указания полного имени файла — даже, если вы работаете в Windows. В PHP символ \ требует цитирования, поэтому его использовать просто-напросто неудобно: `$f = fopen("c:\\home\\counter.txt","r");`

При открытии файлов рекомендуется использовать конструкцию **or die()**, чтобы не анализировать результат открытия файла вручную. В случае ошибки (результат открытия — `false`) будет выведена строка, переданная функции **die()**.

Оператор **or** нельзя заменить созвучным ему логическим оператором ||, поскольку последний имеет более высокий приоритет. Например:

```
$f = fopen("/home/den/counter.txt","r") or die("Ошибка при от-
крытии файла\n");
$f = fopen("/home/den/counter.txt","r") || die("Ошибка при от-
крытии файла\n");
```

В первом случае, при отсутствии открываемого файла, программа нормально завершит свою работу с выводом сообщения "Ошибка при открытии файла". Во втором случае, в этой же ситуации, мы увидим ругательство интерпретатора:

```
<br>
<b>Warning</b>: fopen("/home/den/counter.txt", "r") - No such
file or directory in <b>/home/den/gl12.php</b> on line <b>2</
b><br>
```

11.2.2. Временные файлы и их использование

При написании сложных проектов нам могут потребоваться временные файлы, то есть такие файлы, которые нужно удалить после завершения работы программы. Обычно в таких файлах сохраняются промежуточные

результаты работы программы. Кроме того, временные файлы могут использоваться в качестве буфера при обработке больших файлов. Например, файл занимает 10 Мб, а свободной виртуальной памяти осталось только 8 Мб. В этом случае прочитаем часть файла, обработаем, запишем во временной файл, прочитаем опять часть файла, и т.д.

Для создания временного файла в PHP используется функция **tmpfile()**. Функция возвращает идентификатор файла, при этом нам не нужно указывать имя временного файла — оно выбирается случайным образом. После вызова этой функции, с временным файлом можно работать как с обыкновенным файлом, открытым в режиме чтения и записи. По окончании работы программы, все временные файлы будут удалены. Временные файлы также удаляются из памяти при их закрытии.

Временные файлы сохраняются во временном каталоге. Обычно в Linux — это `/tmp`, а в Windows — `C:\Windows\Temp` (точнее `WINDIR\Temp`).

11.2.3. Чтение и запись в бинарном режиме

Мы уже умеем открывать файлы. Теперь поговорим об их обработке, то есть чтении и записи. На рис. 11.1 и 11.2 показано, как происходит чтение и запись бинарных файлов.

Для каждого открытого файла система хранит величину, которая называется указателем текущей позиции файла. При открытии файла этот указатель устанавливается на начало файла. С каждым вызовом функции чтения файла указатель текущей позиции сдвигается на определенное количество байтов (см. рис. 11.1).

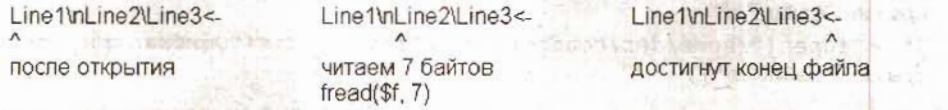


Рис. 11.1. *Чтение бинарного файла*

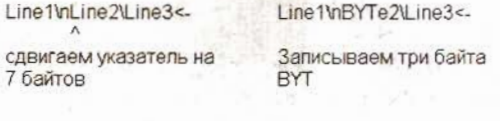


Рис. 11.2. *Запись бинарного файла*

Напомню, что `\n` — один символ. Запись данных в файл начинается с позиции, на которую указывает указатель (см. рис. 11.2).

Чтение двоичного (бинарного) файла осуществляется функцией **fread()**. Данной функции нужно передать всего два параметра — идентификатор файла и количество символов, которое нужно прочитать. Функция возвращает строку, в которой будут прочитанные символы. Если

в качестве параметра функции будет указано большее число символов, чем есть в файле на самом деле, то ошибки не будет — просто будет выведено все, что есть.

Пример чтения двоичного файла приведен в листинге 11.1.

Листинг 11.1. Чтение двоичного (бинарного) файла

```
<?
$f = fopen("/home/den/lines.txt","r") or die("Ошибка при откры-
тии файла\n");
$s = fread($f, 4);
echo $s;
?>
```

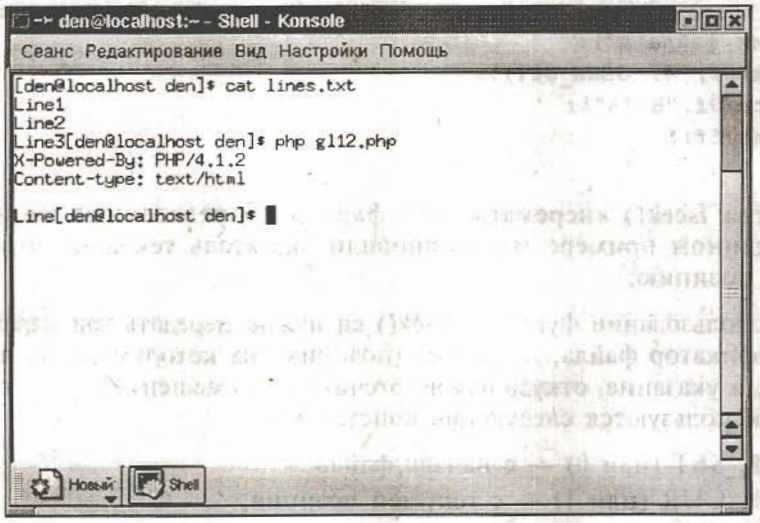


Рис. 11.3. *Чтение двоичного (бинарного) файла*

Мы открыли файл в бинарном режиме "только чтение". Затем мы прочитали 4 байта и записали их в строку `$s`, которая была выведена на экран. На рис. 11.3 мы видим сам файл `lines.txt`, а также вывод сценария — строка `Line`.

Для записи символов в файл используется функция **fwrite()**, которой нужно передать два параметра — идентификатор файла и строку, которая будет записана в файл.

В приведенном выше примере после чтения четырех байтов указатель остановился на пятом байте: следующая операция чтения или записи начнется с пятой позиции. Это означает, что если нам нужно записать что-нибудь после символов `Line`, мы можем сразу после операции чтения записать нужные нам данные. Но мы ведь открыли файл в режиме

чтения, поэтому сейчас запись невозможна. Исправим это упущение. Напишем сценарий, который будет работать так:

- Открывает файл в режиме записи.
- «Перематывает» файл на пятую позицию, то есть устанавливает указатель текущей позиции.
- Записывает информацию в файл.
- Сохраняет изменения в файле.

Текст сценария приведен в листинге 11.2.

Листинг 11.2. Запись в бинарный файл

```
<?
$f = fopen("/home/den/lines.txt","r+") or die("Ошибка при от-
крытии файла\n");
fseek($f, 4, SEEK_SET);
fwrite($f,"BYTE");
fflush($f);
?>
```

Функция **fseek()** «перематывает» файл на определенную позицию. В приведенном примере мы установили указатель текущей позиции на пятую позицию.

При использовании функции **fseek()** ей нужно передать три параметра — идентификатор файла, смещение (позицию, на которую необходимо перейти), и указание, откуда нужно отсчитывать смещение. В качестве указания используются следующие константы:

SEEK_SET (или 0) — с начала файла;
SEEK_CUR (или 1) — с текущей позиции;
SEEK_END (или 2) — с конца файла.

В случае успешного завершения функция возвращает 0, а в случае ошибки −1.

Сразу после **fseek()** мы вызываем функцию **fwrite**, которая записывает строку `BYTE` поверх (режим **r+**) уже имеющихся в файле данных. Обратите внимание, что мы перезаписали четыре символа: `1,\n,Li`. А это означает, что строк в файле будет уже не три, а две, поскольку мы перезаписали символ новой строки (см. рис. 11.4).

Функция **fflush()** записывает на диск все изменения, которые были произведены с указанным в качестве параметра открытым файлом. То есть, сохраняет текущее состояние файла из буфера ввода/вывода на диск. Почему мы использовали эту функцию? Сохранение происходит при закрытии файла, но мы еще не знаем, как это делать, поэтому мы используем функцию **fflush()**.

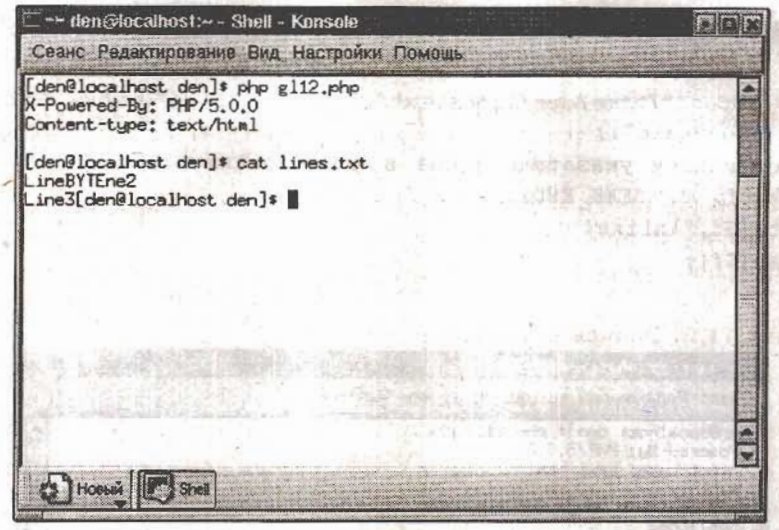


Рис. 11.4. *Запись файла*

Функцию **fflush()** рекомендуется вызывать в программе несколько раз — хотя бы два раза — в середине обработки файла и в самом конце. Ведь если произойдет какой-нибудь сбой (например, сбой питания), ваши данные будут потеряны, а если вы вызовете **fflush()**, то уцелеет какая-то часть обработанных данных. Перед закрытием также желательно вызвать функцию **fflush()**, чтобы принудительно заставить систему сбросить содержимое буферов на диск. Функцию **fflush()** можно даже использовать вместо закрытия файла: PHP сам закроет файл при завершении работы сценария.

Если вы не хотите по десять раз вызывать функцию **fflush()**, вы можете установить размер буфера для файла таким, чтобы в случае отказа системы или питания, потери информации были минимальными. Можно вообще отключить буферизацию для определенного файла, но в этом случае, особенно, при большом объеме передаваемой информации, производительность программы резко снизится. Чтобы установить размер буфера, используйте функцию **set_file_buffer(int $f, int $bufer_size)**. Следующий вызов отключает буферизацию (все изменения сразу же будут записаны прямо на диск) для файла `$f`:

```
set_file_buffer($f, 0);
```

Теперь попробуем записать что-нибудь в конец файла. Конечно, намного проще открыть файл в режиме **a+** — при этом новые данные будут добавлены в конец файла. Но мы будем использовать тот же режим — **r+**. Мы «перемотаем» файл в самый конец — для этого нужно указать параметр SEEK_END и смещение 0 в функции **fseek()**. Текст сценария приведен в листинге 11.3.

Листинг 11.3. Запись в конец файла

```
<?
$f = fopen("/home/den/lines.txt","r+") or die("Ошибка при от-
крытии файла\n");
// Перемещаем указатель файла в конец файла
fseek($f, 0, SEEK_END);
fwrite($f,"\nLine4");
fflush($f);
?>
```

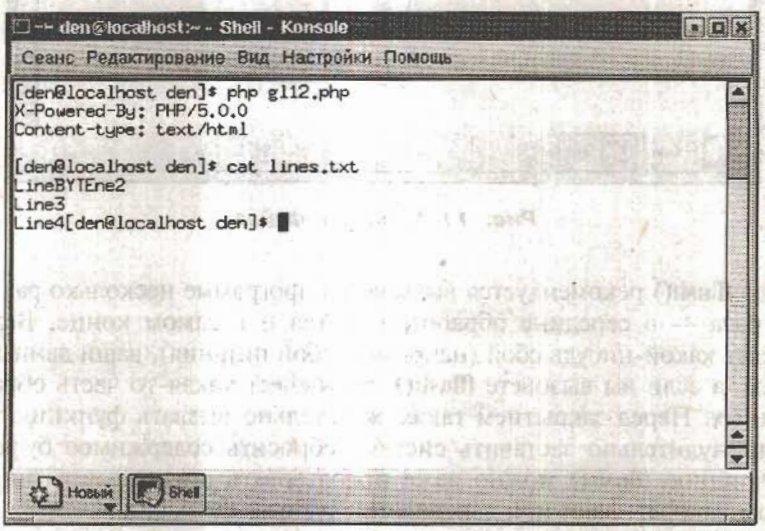


Рис. 11.5. *Запись в конец файла*

Из рис. 11.5 видно, что была произведена запись в файл еще одной строки (\n — символ новой строки). При добавлении в конец файл как бы «расширился», то есть не произошло ошибки, а файл просто стал больше на 6 байтов, дописанных в конец файла.

Теперь рассмотрим очень важную функцию — **feof()**, которая возвращает **true**, если достигнут конец файла. Использование данной функции продемонстрированно в листинге 11.4.

Листинг 11.4. Использование функции feof()

```
<?
$f = fopen("/home/den/lines.txt","r+") or die("Ошибка при от-
крытии файла\n");
while (!feof($f))
{
```

```
$s = fread($f,4);
echo $s;
}
?>
```

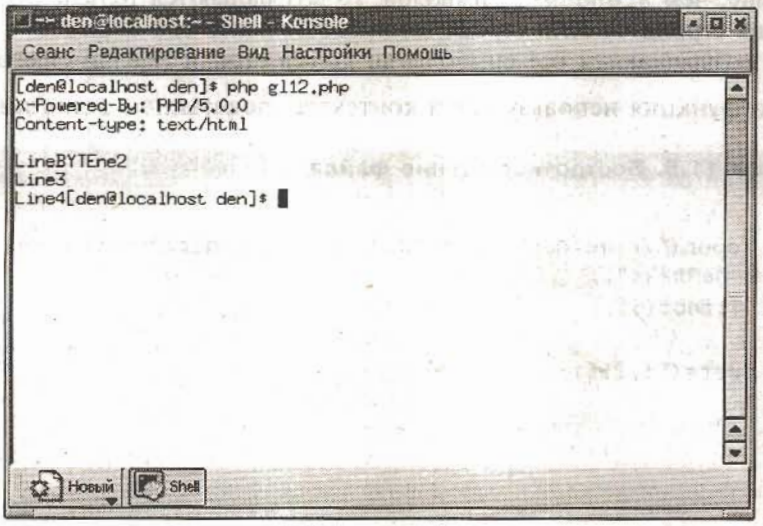


Рис. 11.6. *Чтение файла блоками по 4 байта*

Наш сценарий прекратит работу, когда будет достигнут конец файла. Мы читаем из файла информацию блоками по 4 байта. Можно, конечно, читать посимвольно, но это неэффективно — гораздо эффективнее прочитать блок в строку, а потом работать посимвольно со строкой. Если функции **fread()** не удастся прочесть заданное количество байтов, то она возвратит то, что удалось ей прочитать. Поэтому не нужно беспокоиться о том, что будет достигнут конец файла, а функция будет пытаться прочитать недостающее количество байтов. Функцию **feof()** можно использовать, как при работе с двоичными, так и с текстовыми файлами.

Прочесть весь файл в одну строку можно с помощью такого оператора:

```
$s = fread($f, filesize($f));
```

Однако для этого существуют более эффективные функции, например, **file()**, которую мы рассмотрим позже.

Еще одна функция, которую полезно использовать при работе с файлами в бинарном режиме — это **ftell()**, которая возвращает текущее положение указателя файла. Передать этой функции нужно всего лишь один параметр — идентификатор файла.

11.2.4. Чтение и запись текстовых файлов (построчные чтение и запись)

Для построчного чтения файлов используется функция **fgets(int $f, int $length)**, которая читает из файла `$f` строку длиной `$length` символов. Символ новой строки также включается в результат. Если строка занимает больше, чем `$length-1` символов, то возвращаются только `$length-1` символов строки. А если строка занимает больше, чем `$length-1` символов, то возвращаются все символы до конца строки (т.е. до символа `/n`).

Обычно функция используется в контексте, показанном в листинге 11.5.

Листинг 11.5. Построчное чтение файла

```
<?
$f = fopen("/home/den/lines.txt","rt") or die("Ошибка при от-
крытии файла\n");
while (!feof($f))
{
$s = fgets($f,255);
echo $s;
}

?>
```

Приведенный сценарий проходится по строкам файла `lines.txt` и выводит их. Обычно для построчного чтения файла намного удобнее использовать функцию **file()**, которую мы рассмотрим позже. Эта функция работает намного быстрее, а о функции **fgets()** нужно знать только для общего развития.

Для записи строки в текстовом режиме используется функция **fputs(int $f, string $s)**, которая записывает строку `$s` в текстовый файл `$f`.

11.2.5. Закрытие файлов

Последняя операция над файлом — это его закрытие. Закрыть файл `$f` поможет функция **fclose($f)**. Обычно закрытие файла не требуется, поскольку PHP сам закроет все открытые программой файлы по окончании работы программы. Однако, все же желательно не полагаться на PHP, а выполнить закрытие файла вручную. Во-первых, если программа будет еще продолжаться (после обработки файла), то мы сэкономим память. Во-вторых, если вы что-нибудь записали в файл, то, во избежание потери информации, нужно его закрыть (мало ли что может случиться), а еще лучше вызвать функцию **fflush()**. В-третьих, закрытия файла требует сама культура программирования.

11.2.6. Функция file(). Чтение файла в список

Очень полезной является функция **file (string $filename)**, которая читает весь файл в список, каждый элемент которого равен строке, прочитанной из файла. Путь у нас есть файл `lines.txt`:

```
Line1\n
Line2\n
Line3
```

После вызова `$List = file("lines.txt")` в массиве `$List` будет три элемента `"Line1\n","Line2\n","Line3"`. Нужно заметить, что функция не вырезает символы новой строки и не выполняет их трансляцию. Это означает, что если наш файл будет записан в Windows-формате:

```
Line1\r\n
Line2\r\n
Line3
```

То элементы массива `$List` будут содержать символы `\r\n`.

11.2.7. Чтение файлов в строку

Используя функции **file()** и **join()**, можно прочитать весь файл в одну строку. Как это сделать, можно увидеть в листинге 11.6.

Листинг 11.6. Чтение файла в одну строку

```
<?
$s = join('',file("lines.txt"));
echo $s;
?>
```

С функцией **file()** мы уже знакомы. Остается выяснить, что из себя представляет функция **join()**. Функция **join (string $sss, array $maslist)** возвращает строку, содержащую последовательно расположенные все элементы массива `$maslist`. Первый параметр функции `$sss` — это разделитель элементов массива, то есть символ (символы), который будет вставляться в строке между элементами массива.

Предположим, что нам нужно прочитать HTML-файл и удалить из него все теги. Нет ничего проще (см. листинг 11.7)!

Листинг 11.7. Вывод HTML-файла без тегов

```
<?
$Index = join('',file("http://localhost/index.html"));
$Index = strip_tags($Index);
echo $Index;
?>
```

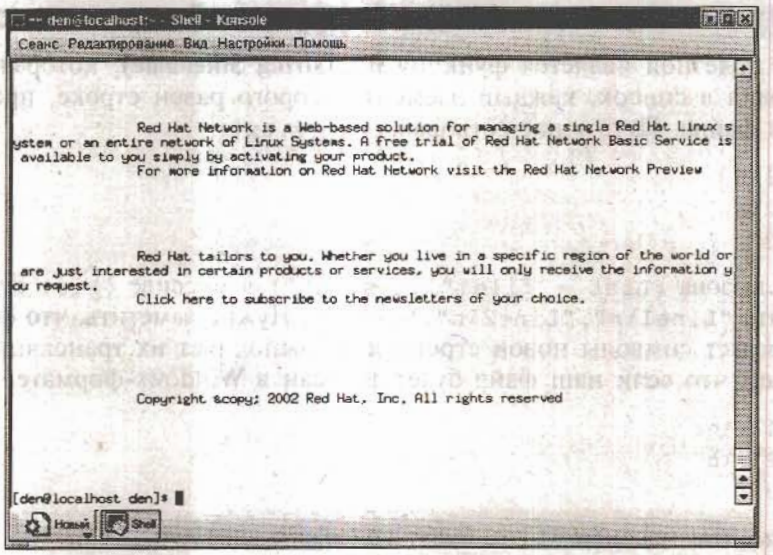


Рис. 11.7. *Выведен только текст – без HTML-тегов*

Сначала мы получаем файл с помощью функции **file()**. Обратите внимание: мы можем получить файл по протоколу HTTP и даже FTP. Функция **file()** возвращает список строк, которые содержит файл, а функция **join()** объединяет все элементы этого списка в строку. В качестве разделителя ничего не используется — " (две одинарные кавычки). Результат работы функции **join()** мы записываем в строку `$Index`, которая будет выведена оператором **echo** после удаления HTML-тегов функцией **strip_tags()**.

Можно было бы обойтись без функции **join()**, поочередно обрабатывая элементы массива, но предложенный вариант работает намного быстрее. Хотя, если размер выводимого файла очень большой, тогда может понадобиться построковая обработка, чтобы не было задержек с выводом в браузер, иначе пользователь подумает, что сценарий «завис». Для этого случая можно предложить сценарий, приведенный в листинге 11.8.

Листинг 11.8. Вывод HTML-файла без тегов (2)

```
<?
$Index = file("http://localhost/index.html");
foreach($Index as $v)
 {
 echo strip_tags($v);   // удаляем теги
 flush();          // выводим данные в браузер
 }
?>
```

11.2.8. Чтение CSV-файлов

Некоторые программы электронных таблиц (включая небезызвестную Excel от Microsoft) позволяют сохранять электронные таблицы в формате CSV. Особенностью CSV-формата является использование запятой в качество разделителя. Если заполнить четыре ячейки электронной таблицы (A1, A2, B1, B2 — значения «A1», «A2», «B1», «B2»), а потом сохранить таблицу в формате CSV, то мы получим файл, показанный на рис. 11.8.

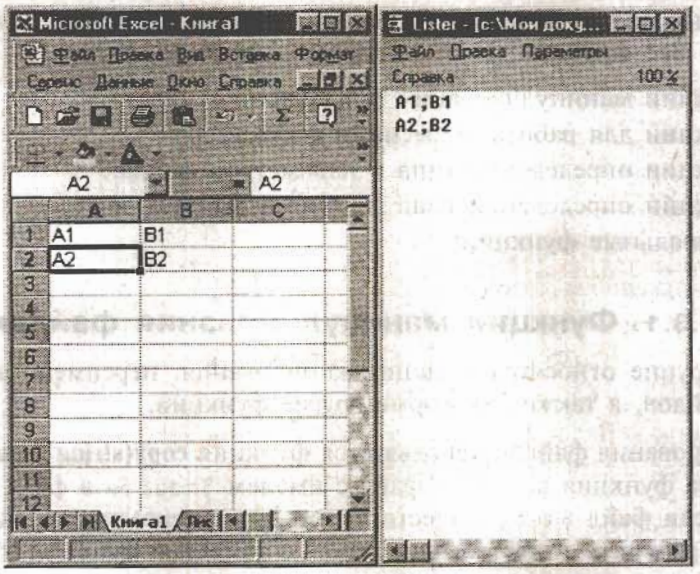


Рис. 11.8. Файл в формате CSV

Для чтения CSV-файлов используется функция **fgetcsv(int $file, int $length, char $separator = ',')**. Функция читает одну строку из файла `$file` и разбивает ее по символу `$separator`, который по умолчанию равен ','. Если последний параметр является строкой из более, чем одного символа, то берет только первый из них. Функция возвращает список или `false`, если достигнут конец файла. После прочтения первой строки нашего файла («A1; B1») функция возвратит список из двух элементов — «A1» и «B1». Параметр `$length` задает максимальную длину строки.

Пример:

```
$f=fopen("Книга1.csv","r") or die("Невозможно открыть файл");
echo "<PRE>";
for($i=0;$array=fgetcsv($f,1024); $i++) {
$j = count($array);
```


9 Зак. 623

```
    if ($j==1 && array[0]==="") continue;
    for ($k=0; $k<$j; $j++) echo "$array[$k] ";
    echo "<BR>";
  }
 fclose($f);
```

11.3. Функции для работы с файлами

PHP предоставляет большой выбор самых разных функций для работы с файлами. Все функции можно разделить на такие группы:

- Функции манипулирования файлами.
- Функции для работы с именами файлов.
- Функции определения типа и параметров файла.
- Функции определения прав доступа к файлу.
- Специальные функции.

11.3.1. Функции манипулирования файлами

К этой группе относятся функции копирования, переименования, удаления файлов, а также некоторые другие функции.

Для копирования файлов используется функция **copy(string $source, string $dest)**. Эта функция копирует файл с именем `$source` в файл с именем `$dest`. Если файл `$dest` существует, он будет перезаписан, поэтому вы должны проверить существование файла с помощью функции **file_exists()**, которая будет рассмотрена ниже.

Еще один момент, о котором вы должны позаботиться прежде, чем начать копирование. Вы должны иметь право на запись в каталог, в котором должен находиться файл `$dest`, и в сам файл, если он до вызова функции существовал, чтобы перезаписать его.

Функция **copy()** возвращает `true`, если файл удачно скопирован, и `false` — в обратном случае.

Переименовать файл можно с помощью функции **rename(string $oldname, string $newname)**. Функция переименует файл `$oldname` в файл `$newname`. В случае, если переименовать файл нельзя, функция возвратит `false`. Это может произойти в следующих случаях:

- Файл `$oldname` не существует.
- У вас нет прав доступа к файлу `$oldname`.
- Файл с именем `$newname` уже существует. Обратите внимание, функция **rename()**, в отличие от функции **copy()**, не перезаписывает файл.

- Имя `$newname` находится на другой файловой системе, то есть на другом разделе диска или сетевой файловой системе.
- Другие причины.

Удалить файл можно функцией **unlink(string $name)**. Функция возвращает `false`, если файл не удалось удалить.

Примечание.

Может возникнуть вопрос: «Почему эта функция называется **unlink()**, а не **remove()**, **erase()** или **delete()**»? Дело в том, что файловая система Unix-подобных операционных систем позволяет создавать жесткие и мягкие (или символические) ссылки на файлы – hard and soft (symbolic) links. Жесткая ссылка – это ни что иное, как имя файла. При этом файловая система может содержать один и тот же файл, но использовать для доступа к нему разные имена. Определить жесткие ссылки просто – имена файлов разные, а номера инодов (информационных дескрипторов, индивидуальных для каждого файла) – одинаковые. Все жесткие ссылки (имена) являются равноправными, и вы не можете удалить файл до тех пор, пока существует хотя бы одна *жесткая* ссылка на него. Прежде чем удалить файл, вы должны удалить все жесткие ссылки, файл будет удален, когда будет удалено его последнее имя. Все вышесказанное не относится к символическим ссылкам. Таким образом, становится понятно, почему функция удаления файла называется **unlink()**.

11.3.2. Функции для работы с именами файлов

Довольно часто приходится иметь дело с именами файлов. При этом требуется, то присоединить имя файла к имени каталога, то наоборот — выделить из полного пути непосредственное имя файла, и т.д.

Из всех функций данной группы на поверку оказались полезными только четыре:

file_exists() — проверяет существование файла с указанным именем;
basename() — выделяет имя файла из полного пути;
dirname() — возвращает имя каталога из полного пути;
realpath() — преобразует относительный путь к файлу в абсолютный;
tempname() — генерирует имя временного файла.

Первая функция — **file_exists(string $name)** проверяет существование файла с именем `$name`:

```
<?
if (!file_exists("/home/den/lines.txt"))
 {
 echo "Файл не существует";
 }
 else echo "Файл существует";
?>
```

Очевидно, что в приведенном сценарии, если файл существует, то будет выведено сообщение «Файл существует». В противном случае вы увидите сообщение «Файл не существует». После вывода этого сообщения можно вставить код, создающий новый файл, а можно завершить работу сценария.

Функция **basename(string $full_filename)** возвращает основное имя файла, то есть имя файла без пути к нему.

```
echo basename("/home/den/lines.txt"); // выведет lines.txt
echo basename("/."); // выведет.
```

Обратите внимание, функция **basename()** не проверяет существование файла, она просто возвращает часть строки, стоящую после последнего слеша. Данная функция правильно обрабатывает, как прямые слеши, так и обратные (что важно для Windows).

Функция **dirname(string $path)** возвращает имя каталога:

```
echo dirname("/home/den/lines.txt"); // выведет /home/den
echo dirname("../lines.txt"); // выведет ..
echo dirname("lines.txt"); // выведет . (текущий каталог)
echo dirname("/swap_file"); // выведет / - в Linux, \ - в Windows
```

Функция **realpath()** преобразует относительный путь к файлу в абсолютный (полный):

```
echo realpath("./lines.txt"); // выведет /home/den/lines.txt
```

Функция **tempname(string $dir, string $prefix)** генерирует имя файла в каталоге `$dir` с префиксом `$prefix`, причем делает это так, что получаемое имя файла уникально. В качество аргумента `$dir` обычно указывается временный каталог, а в качестве `$prefix` — имя нашего сценария (чтобы знать, какая программа сгенерировала файл), но это необязательно — вы можете указать все, что угодно:

```
$fname = tempnam("/var/www/html/tmp","dev");
```

11.3.3. Функции определения типа и параметров файла

Для определения типа файла используется функция **filetype(string $filename)**, которая возвращает тип файла — строку (см. табл. 11.2).

Для определения параметров файла используется функция **stat(string $filename)**, которая возвращает подробную информацию о файле — возвращается массив, состоящий из тринадцати элементов (см. табл. 11.3).

Типы файлов *Таблица 11.2*

Возвращаемое значение	Тип файла
file	Самый обыкновенный файл. Может быть, как текстовым, так и двоичным
dir	Каталог
link	Символическая ссылка
fifo	FIFO-канал
block	Блочно-ориентированное устройство (например, жесткий диск — /dev/hddN)
char	Символьно-ориентированное устройство (например, терминал — /dev/ttyN)
unknown	Тип определить не удалось

Результат функции stat() *Таблица 11.3*

Ключ массива	Значение
0	Устройство
1	Номер inode
2	Атрибуты файла
3	Число жестких ссылок на файл
4	Числовой идентификатор (User ID – UID) владельца файла
5	Числовой идентификатор группы (Group ID – GID) владельца файла
6	Тип устройства
7	Размер файла в байтах
8	Время последнего доступа (с 1 января 1970 года) в секундах
9	Время последнего изменения файла
10	Время последнего изменения атрибутов файла
11	Размер блока
12	Число блоков, которые занимает файл

Сценарий, представленный в листинге 11.9, собирает детальную информацию о файле `/home/den/lines.txt`:

Листинг 11.9. Детальная информация о файле

```
<?
$f_info = stat("/home/den/lines.txt");

echo "Файл: /home/den/lines.txt\n";
echo "Устройство: \t\t$f_info[0]\n";
echo "Номер inode: \t\t$f_info[1]\n";
echo "Атрибуты \t\t$f_info[2]\n";
echo "Число жестких ссылок: \t$f_info[3]\n";
echo "UID $f_info[4] GID $f_info[5]\n";
echo "Тип устройства: \t$f_info[6]\n";
```

```
echo "Размер: \t\t$f_info[7]\n";
echo "Время последнего доступа: ".date("d M Y
H:i:s",$f_info[8])."\n";
echo "Время последнего изменения файла: ".date("d M Y
H:i:s",$f_info[9])."\n";
echo "Время последнего изменения атрибутов файла: ".date("d M Y
H:i:s",$f_info[10])."\n";
echo "Размер блока: $f_info[11]\n";
echo "Число блоков, которые занимает файл: $f_info[12]\n";
?>
```

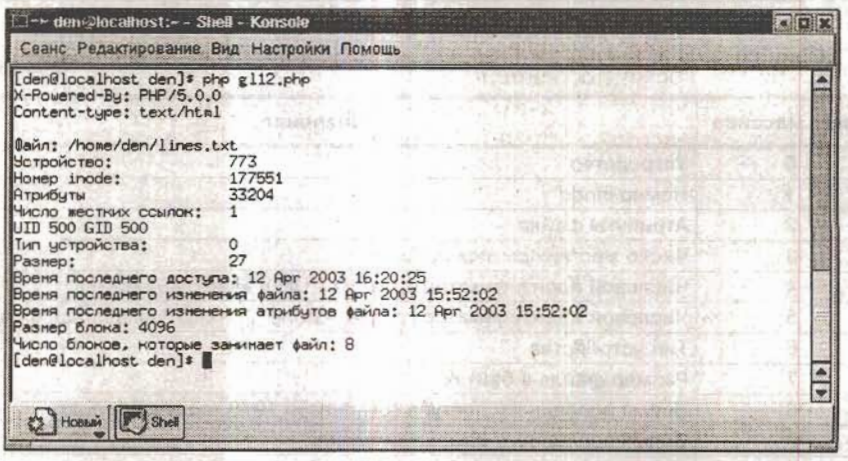


Рис. 11.9. *Информация о файле*

Если вы работаете под Windows 9x, многие элементы массива будут содержать пустую строку.

Если вы вызвали функцию **stat()** для символической ссылки, то будет возвращена информация о файле, на который ссылается ссылка. Чтобы получить информацию о ссылке, используйте функцию **lstat()**. Формат возвращаемого нею массива такой же, как и для функции **stat()**.

11.3.4. Специальные функции

Разработчики PHP предусмотрели несколько полезных и удобных функций для работы со временем создания и модификации файла. Конечно, можно использовать функцию **stat()**, но если нам только нужно узнать время создания или модификации файла использовать **stat()** не только не удобно, но и не рационально.

Функция **fileatime(string $filename)** возвращает время последнего доступа (a — access) к файлу. Если файл не существует, она возвращает `false`.

Функция **filectime(string $filename)** возвращает время создания (c — create) файла. Функция **filemtime(string $filename)** возвращает время последнего изменения (m — modify) файла.

Пример использования функций:

```
A = fileatime("/home/den/lines.txt");
C = filectime("/home/den/lines.txt");
M = filemtime("/home/den/lines.txt");
echo "Время последнего доступа: ".date("d M Y H:i:s",A)."\n";
echo "Время создания файла: ".date("d M Y H:i:s",C)."\n";
echo "Время последнего изменения файла: ".date("d M Y H:i:s",M)."\n";
```

Функция **touch(string $filename, int $time)** позволяет установить для файла `$filename` время последней модификации `$time`. Если второй параметр не указывается, то используется текущеее время. Если же указанный файл не существует, то он будет создан.

Пример:

```
M = filemtime("/home/den/lines.txt");
// устанавливаем время модификации файла some_file.txt таким
// же, как у файла lines.txt
touch("/home/den/some_file.txt",M);
// время модификации - текущая дата (второй параметр
// не указан)
touch("/home/den/file.txt");
// Файл no_such_file не существует, поэтому функция возвратит
// false и создаст новый пустой
// файл с именем no_such_file
touch("/home/den/no_such_file");
```

11.4. Права доступа

Для каждого файла и каталога в ОС Linux(Unix) и ОС Windows задаются права доступа. Права доступа определяют, кто имеет доступ к объекту и какие операции над объектом он может выполнять. Под объектом следует понимать файл или каталог. Выполнять можно три основных операции: чтение, запись и выполнение.

11.4.1. Права доступа в ОС Linux

В ОС Linux право на чтение файла означает, что его можно просматривать и печатать, а для каталога — что может отображаться список содержащихся в нем файлов. Право на запись для файла означает возможность его редактирования, а для каталога — возможность создания и удаления в нем файлов.

Если для файла установлено право выполнения, то его можно запускать как программу. Данная возможность используется при написании сценариев командных интерпретаторов. Право выполнения для каталога означает право доступа к каталогу, но не право на выполнение расположенных в нем файлов, как это может показаться, исходя из названия режима доступа. То есть право выполнения для каталога позволяет перейти в этот каталог (командой cd) и выбирать в нем файлы.

Кстати говоря, с помощью атрибутов для каталогов можно достичь интересного эффекта, когда каталог невидим, но из него можно запускать (или записывать в него) файлы, если знать их полное имя.

При установке права доступа принимается во внимание категория пользователей. В общем случае существует три категории пользователей: владелец, группа и прочие.

- **Владелец** — пользователь, создавший файл. Само собой разумеется, для того, чтобы создать файл, вы должны иметь право записи в каталог, в котором вы создаете файл. При создании файла обычно устанавливаются права на чтение и запись для владельца, и только чтение для всех остальных пользователей.
- **Пользователи** объединяются в группы, например, для работы над одним проектом. Владелец может разрешить или запретить доступ к файлам для членов группы.
- **Прочие** — это все остальные пользователи.

Для каждой категории имеется свой набор прав доступа, просмотреть который вы можете с помощью команды **ls –l**:

```
ls -l file.txt
-rw-r----- 1 den group 300 Feb 10 12:00 file.txt
```

Строка **-rw-r-----** — характеризует набор прав доступа к файлу `file.txt`. Владельцем файла является пользователь `den`, который входит в группу `group`.

Первый символ — это тип файла. При этом «-» означает файл, а «d» — каталог.

Следующие три символа «**rw-**» задают права доступа для владельца файла. Символ «r» — это право на чтение, «w» — на запись, а «x» — на выполнение. Права задаются именно в таком порядке: чтение, запись, выполнение. Если право на какой-нибудь вид доступа отсутствует, то ставится «-». В данном случае отсутствует право на выполнение.

Второй трехсимвольный набор задает права доступа для группы, а третий — для прочих пользователей. В нашем примере (r--) члены группы имеют право только на чтение, а другие пользователи вообще не имеют никакого доступа к файлу (---).

Для изменения прав доступа в Linux используется команда **chmod**. При этом права доступа можно задавать двумя способами: символьным и абсолютным. В этой книге будет рассмотрен абсолютный, поскольку он нам еще пригодится.

Примечание.
О символьном способе вы можете прочитать в моей книге "Linux-сервер своими руками", которая вышла в издательстве "Наука и техника".

Как мне кажется, абсолютный способ несколько удобнее, чем символьный, поскольку не нужно помнить символику прав доступа. Этот метод еще называют методом двоичных масок.

Для изменения прав доступа абсолютным методом используется та же команда **chmod**:

```
chmod число <файл|каталог>
```

Число называется маской прав доступа и представляет собой число в восьмеричной системе, задающее наборы прав доступа. Напомню, что восьмеричная система — это система с основанием 8 (см. табл. 11.4). Не спешите переворачивать страницу, услышав слово «восьмеричная», все на самом деле намного проще, чем звучит.

Каждое число, задающее права доступа, состоит из трех разрядов, например, 760:

7 — первый разряд;
6 — второй разряд;
0 — третий разряд.

Первый разряд задает права доступа для владельца файла, второй — для группы, третий — для остальных пользователей. Одному разряду восьмеричной системы соответствует три разряда в двоичной.

Соответствие разрядов восьмеричной системы разрядам в двоичной системе

Таблица 11.4

Восьмеричный формат	Двоичный формат
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Каждый двоичный разряд задает соответствующий ему тип доступа: первый — чтение, второй — запись, третий — выполнение. Будем считать, что разряды нумеруются слева направо. 0 — если данная операция запрещена, а 1 — если разрешена. Теперь все становится на свои места, например, право доступа, задаваемое числом 777 (111 111 111) означает право на чтение, запись и выполнение для всех пользователей.

Наиболее часто используется маска 644, разрешающая чтение и модификацию файла для владельца, и только чтение для других пользователей. Иногда используется набор прав доступа, состоящий из четырех цифр. Старшая цифра обозначает флаги SETUID (4), SETGID (2), sticky-бит (1).

Какие же права доступа будут у только что созданного каталога командой **mkdir**? Обычно маска прав доступа равна 0777 минус значение, выводимое командой **umask**. Значение, выводимое командой **umask,** обычно равно 022. Следовательно, маска прав доступа будет равна 0777 – 0022 = 0755. Это значит, что владелец (см. первую цифру — 7, в восьмеричной системе — 1111) файла может читать (1), записывать (1) и присматривать (1) файлы в каталогах. Группа владельца (вторая 5-ка, в восьмеричной системе 101) и все остальные (третья 5-ка) имеют право читать (1) файлы в каталоге, не имеют права записывать (0), и могут просматривать оглавление каталога (1).

И действительно, создайте каталог и введите команду **ls –l**:

```
mkdir new
ls -l
...
drwxr-xr-x 2 den den 4096 Янв 14 14:30 new/
...
```

Набор **rwx** (111) равен семи, а набор **r-x** (101) равен пяти в восьмеричной системе. Получается, что маска доступа к новому каталогу = 755. Установить значение **umask** можно с помощью самой этой команды. Просто выполните команду **umask 000** от имени суперпользователя и вы установите новое значение, равное нулю. Обратите внимание, команда **mkdir** автоматически устанавливает право выполнения (доступа) для каталога, иначе к каталогу нельзя будет получить доступ.

11.4.2. Права доступа в Windows

Вот тут складывается интересная ситуация: под Windows (имеется в виду 95/98/ME) вы можете просмотреть, изменить и даже удалить любой файл. Ваш сценарий может сделать все: даже записать команду `echo y | format c: /u` в ваш `c:\autoexec.bat`. Вот поэтому в Интернете нет ни одного сервера на базе Windows 98.

В операционных системах Windows NT/2000/XP существуют права доступа: вы можете разрешить читать/записывать и т.д. свои файлы конкретному пользователю или группе пользователей, но только в том случае, если ваш файл находится на файловой системе NTFS. Если же ваши диски отформатированы под FAT32, повторяется ситуация с Windows 9x.

11.4.3. Функции определения прав доступа к файлу

Подробно рассмотрев права доступа, теперь вернемся в PHP. Рассмотрим функции, которые предоставляет нам PHP для определения прав доступа:

- Функция **is_readable(string $filename)** возвращает `true`, если мы можем прочитать файл, то есть открыть его в режиме «r».
- Функция **is_writeable(string $filename)** возвращает `true`, если мы можем записать информацию в файл.
- Функция **is_executable(string $filename)** возвращает `true`, если файл исполняемый (является программой).

Эти функции рекомендуется использовать перед открытием важных файлов для обеспечения корректности доступа к ним.

11.5. Блокирование доступа

Общие положения

Операционная система UNIX, равно как и Linux, является многозадачной, поэтому нужно предусмотреть совместную (одновременную) работу нескольких программ с одним файлом.

Представим такую ситуацию: процесс А пытается открыть файл для записи, а процесс Б — для чтения (или оба процесса — для записи; режим доступа тут особой роли не играет). При этом непонятно, что делать с тем процессом, который обратился к файлу позже. Вместе эти два процесса могут вам такого натворить, что вы потом никогда не поймете, что и откуда взялось. Поэтому в Linux есть механизмы блокирования доступа к файлу.

Когда мы знаем о существовании механизма блокирования доступа, можно уже догадаться, как будут вести себя процессы. Например, процесс Б пытается открыть файл, который уже открыт процессом А. Процесс А сразу же после открытия блокировал файл. Тогда процессу Б ничего другого не остается, как подождать, пока процесс А снимет блокировку с файла. Обратите внимание, *снимет блокировку*, а *не завершит работу*. Поэтому, когда вы установите блокировку, постарайтесь как можно быстрее ее снять — сразу же после окончания работы с файлом.

Предположим, что вы хотите обработать массив данных, которые хранятся в файле. После того, как вы прочитаете массив из файла, вы должны освободить файл, чтобы он не был блокирован на время обработки вашей программой. Перед записью в файл опять заблокируйте файл, чтобы никакой другой процесс не повредил информацию. Конечно, в этом случае не исключена ситуация, когда другой процесс заблокирует файл и уже вам придется ждать его освобождения.

Виды блокировки

Существует два вида блокировки:

- **исключительная** — только мы можем использовать файл (обычно используется для записи данных в файл);
- **разделяемая** — используемым файлом мы можем «поделиться с соседями», то есть другими процессами, запущенными на нашей машине.

Предположим, что у нас есть какой-нибудь файл и два процесса, которым нужно записать информацию в этот файл. Процесс А первым *открыл* файл для записи и *заблокировал* его, используя исключительную блокировку. Тогда процесс Б будет ждать, пока процесс А не разблокирует файл или не закроет его. Только тогда он сможет получить доступ к файлу для записи. Исключительную блокировку нужно использовать тогда и только тогда, когда вам нужно что-нибудь записать в файл.

Если вам нужно только прочитать данные из файла, целесообразнее использовать разделяемую блокировку. Представьте, что мы пишем счетчик посещений. Допустим, что есть несколько процессов, которым нужно прочитать информацию. Если мы используем исключительную блокировку для чтения, эти процессы будут ждать, пока мы не разблокируем файл. Но ведь чтение — это не критическая операция, поэтому с точки зрения повышения производительности нужно предоставить другим процессам возможность прочитать файл.

Мы рассмотрели два случая совместного использования файла: когда несколько процессов одновременно пытаются записать что-нибудь в файл, и когда несколько процессов одновременно пытаются прочитать файл. А что делать, если один процесс хочет записать в файл, а несколько — прочитать файл. В этом случае процессы, открывшие файл для чтения, будут обязаны подождать, пока первый процесс завершит запись и разблокирует файл. И наоборот, если в данное время какой-то процесс читает файл, процесс, которому нужно писать в файл, должен подождать, пока первый процесс не разблокирует файл.

Немного подытожим: если вам нужно писать в файл, обязательно используйте исключительную блокировку, а если читать — только разделяемую.

Реализация блокировки в PHP

Для блокирования файла в PHP используется функция **flock(int $f, int $operation, [$block])**. В качестве первого параметра этой функции указывается дескриптор блокируемого файла (`$f`) для текущего процесса, а в качестве второго (`$operation`) — режим блокировки. Второй параметр (режим блокировки) может принимать следующие значения:

- LOCK_SH — разделяемая блокировка (разделяемый доступ для чтения). Это значение говорит о том, что файл в данный момент

читается. Другие программы(сценарии) также могут открыть его для чтения и читать из него данные. Однако, если кому-либо из них понадобится записать что-либо в файл, то ему придется подождать своей очереди.

- LOCK_EX — исключительная блокировка (монопольный доступ для записи). При этом все остальные сценарии, кроме текущего, не могут ничего делать с файлом.
- LOCK_UN — снять блокировку.
- LOCK_NB — прибавьте эту константу к одной из вышеуказанных, если вы не хотите ждать, пока получите разрешение на блокировку файла.

Третий необязательный параметр принимает значение `true`, если в режиме без ожидания не удалось получить разрешение на блокировку файла (то есть кто-то уже использовал файл).

Функция **flock()** устанавливает для файла флаг, который говорит о том, что с файлом ведется работа. Другой сценарий (программа) при обращении к этому файлу прочитает этот флаг и сделает соответствующие выводы.

Рассмотрим пример. Допустим, нам необходимо дописать какую-либо информацию в конец файла. В обычной ситуации мы бы поступили, как показано в листинге 11.10.

Листинг 11.10. Запись в конец файла без блокировки (неправильно)

```
<?
$str="stroka"
$f = fopen($f, "a+") of die("Error");
fwrite ($f, $str."\n")
fclose($f);
```

Теперь же знаем, что это неправильно и что необходимо поступать, как показано в листинге 11.11.

Листинг 11.11. Запись в конец файла с блокировкой (правильно)

```
<?
$f = fopen($f, "a+") of die("Error");
flock($f, LOCK_EX);      // Ждем, пока не получим разрешение на
                         // исключительную блокировку
                         // и потом устанавливаем ее
                         // Обратите внимание, что программа
                         // не получит управления от функции
                         // flock(), пока не будет установлена
                         // блокировка
fwrite ($f, $str."\n")
flock($f, LOCK_UN);      // снимаем блокировку
fclose($f);
?>
```

Теперь наш скрипт будет работать безопасно. И если несколько файлов (или экземпляров данного скрипта) захотят записать что-либо в файл, то они это сделают по очереди.

Для полноты картины приведу пример процесса, читающего из файла (см. листинг 11.12).

Листинг 11.12. Чтение файла с блокировкой

```
<?
$f = fopen($f, "r") or die("Error");

// Ждем, пока не получим разрешение на разделяемую блокировку
flock($f, LOCK_SH);

// чтение
$s = fgets($f,255);

flock($f, LOCK_UN); // снимаем блокировку

// продолжаем работу
fclose($f);
?>
```

Зачастую бывает полезно не дожидаться, пока функция **flock()** разрешит доступ к файлу и позволит сценарию выполняться дальше. Можно ведь это время ожидания потратить с пользой и выполнить какие-либо другие действия. Это как раз и позволяет сделать параметр LOCK_NB, указываемый дополнительно к основному параметру блокировки.

Пример блокировки без ожидания:

```
<?

$f = fopen($f, "r+") or die("Error");

// Ждем, пока не получим разрешение на исключительную блокировку
// Обратите внимание, что программа не получит
// управления от функции
// flock(), пока не будет установлена блокировка

$i = 0;

while(!flock($f, LOCK_EX+LOCK_NB))
{
$i++;
```

```
echo "Попытка номер $i";
sleep(1);  // засыпаем на 1 секунду
}

// Мы получили доступ, можно работать с файлом

fclose($f); // Закрываем файл - он автоматически разблокируется
?>
```

При работе с блокировками нужно учитывать, что они не передаются процессам-потомкам. То есть, если вы заблокировали файл и вызвали какой-то процесс, но он не получит доступ к этому файлу до тех пор, пока вы не разблокируете его.

Также необходимо помнить, что применение функции **flock()** возможно только к открытому файлу. На закрытый файл она никакого действия не возымеет.

При записи в файл с использованием блокировки лучше использовать режимы **r+** и **a+**, но не **w**.

11.6. Запись данных в начало файла

Довольно часто возникает потребность записи в начало файла. Взять даже самый тривиальный пример: запись нового сообщения в гостевую книгу. Пользователь, который читает нашу гостевую книгу, хочет видеть сначала самые «свежие» сообщения, а уже потом — все остальные. Можно, конечно, читать с конца файла, но мне кажется, это довольно неудобно.

В PHP нет стандартных функций, позволяющих записать информацию в начало файла. Если поставить указатель файла на самое начало (при условии, что вы открыли файл в бинарном режиме) и что-то записать в файл, то новая информация перезапишет уже существующую. Однако, следующий пример продемонстрирует вам, как можно организовать запись в начало файла (см. листинг 11.13).

Листинг 11.13. Запись в начало файла

```
<?
$file_gb="gb.txt"; // файл гостевой книги
$file_tmp="gb_tmp.txt"; // временный файл
$str="новая строка текста";

// проверяем, не было ли сбоя в предыдущем запуске скрипта
if(file_exists($file_tmp)) die("Временный файл существует!");
```

```
// копируем содержимое файла в tmp
if(copy($file_gb, $file_tmp))
{
// если удачно скопировался, можно перезаписывать основной файл
 if($w=fopen($file_gb,"w"))
 {
  flock($w,2);       // блокируем файл
  fwrite($w,$str."\n"); // записываем первую строку

  if(!$r=fopen($file_tmp,"r")) die("невозможно открыть файл");
  flock($r,1);
  while($str=fgets($r,10240)) // читаем построчно
  {
   fputs($w,$str);    // пишем построчно
  }
  flock($r,3);
  fclose($r);
  flock($w,3);
  fclose($w);
  unlink($file_tmp);    // удаляем временный файл
 }
}
?>
```

11.7. Функции для работы с каталогами

11.7.1. Манипуляции с каталогами

При написании сценариев, кроме уже описанных функций для работы с каталогами, нам пригодятся еще три:

- **mkdir (string $dir_name int $perms)** — создает каталог с именем `$dir_name` и правами доступа `$perms`.
- **rmdir(string $dir_name)** — удаляет *пустой* каталог.
- **getcwd()** — возвращает текущий каталог.
- **chdir(string $dir_name)** — изменяет текущий каталог.

Первая функция создает каталог и устанавливает права доступа к этому каталогу. Например:

```
// Создает каталог dir в текущем каталоге. Вы должны
// иметь право на запись в текущий
// каталог
mkdir("dir",0755)
```

Обратите внимание, что используется абсолютный (числовой) способ указания прав доступа, который был рассмотрен выше. При этом, поскольку они задаются в восьмеричной системе, то необходимо сказать об этом PHP, для чего указать первым символом 0. Чаще всего устанавливают атрибуты доступа 0770, что означает доступ на запись, чтение и исполнение для владельца файла и его группы.

Вторая функция **rmdir(string $dir_name)** удаляет пустой каталог. Это означает, что если каталог содержит файлы, он не будет удален.

Третья функция **getcwd()** возвращает текущий каталог. Последняя функция, **chdir(string $dir_name)**, изменяет текущий каталог, например, **chdir("/var/www/html/docs")**.

11.7.2. Открытие, чтение и закрытие каталогов

Каталог — это обыкновенный файл, содержащий информацию о находящихся «в нем» файлах. Последовательность работы с каталогом такая же, как и с файлом: открыть, прочитать (писать в каталог мы не можем) и закрыть. Для этого в PHP есть три функции:

- **opendir(string $dir_name)** — открывает каталог, после чего мы можем работать с каталогом: читать оглавление каталога. Функция возвращает дескриптор каталога `$handle`, который нужно указывать в качестве аргумента при вызове других функций для работы с каталогом;
- **readdir(int $hadle)** — считывает следующее имя файла или подкаталога, которые находятся в каталоге, заданным дескриптором $handle;
- **closedir(int $hadle)** — закрывает каталог.

Пример совместного использования всех трех функций приведен в листинге 11.14. Результат работы приведен на рис. 11.10.

Листинг 11.14. Открытие, чтение и закрытие каталогов

```
<?

$dir = opendir("/var/www/html");
chdir("/var/www/html");

echo "<html><head><title>Печать каталога</title></head><body>";
echo "<h1>Оглавление каталога</h1><p>";
echo "<table width=100%>";

while ($d=readdir($dir))
{
echo "<tr><td>";
```

```
if (is_dir($d)) echo "$d</td><td>Каталог</td>";
if (is_file($d)) echo "$d</td><td>".filesize($d)."</td>";
echo "<tr>";
}

echo "</table></body></html>";
closedir($dir);
?>
```

Теперь проанализируем программу. Сначала мы открываем каталог (**opendir**), потом делаем его текущим (**chdir**). В цикле `while($d=readdir($dir))` мы читаем каталог и анализируем его записи - `$d`. Если `$d` — это каталог (**is_dir**), выводим в соответствующую ячейку таблицы слово «Каталог», а если `$d` — файл (**is_file**), то выводим его размер (**filesize**). Прочитав все записи каталога, мы его закрываем (**closedir**).

Примечание.

Как вы уже успели догадаться, функция **is_dir(string $name)** возвращает true, если переданное ей имя является именем каталога, а не файла. А функция **is_file(string name)**, наоборот, возвращает true, если мы передали ей имя файла, а не каталога.

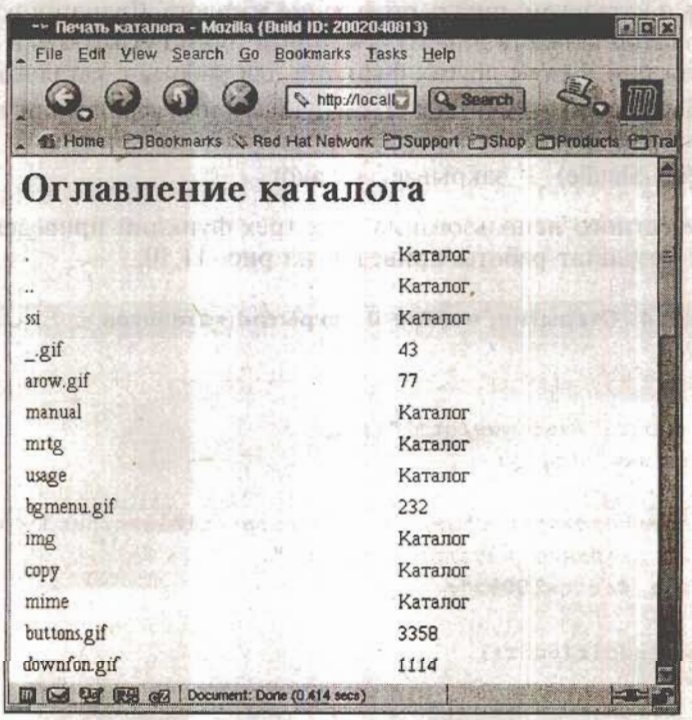


Рис. 11.10. Содержимое каталога

Цикл `while ($d=readdir($dir))` не очень практичен: если в каталоге окажется файл с именем 0, то цикл прекратит свою работу. Поэтому лучше использовать следующий цикл:

```
while(($d = readdir($d))!==false)
```

Каталог, как и файл, обладает указателем текущей позиции. Чтобы по десять раз не закрывать/открывать каталог, для сброса указателя можно использовать функцию **rewinddir()**, которая «перематывает» каталог на начало:

```
$dir = opendir("dir");
....
rewinddir($dir);
...

closedir($dir);
```

11.8. Каналы

11.8.1. Общие положения

Честно говоря, каналы вряд ли пригодятся Web-программисту — они более полезны для системного программиста. Поэтому сначала мы опишем работу с каналами на языке C, а потом — на PHP.

Канал — это связь между родительским и дочерним процессом, то есть каналы позволяют обмениваться информацией между процессами, минуя файлы. Передача информации производится через оперативную память. Можно было бы записать информацию в файл одной программой, а второй читать, то, что записали первой (о том, как использовать блокировку файлов вы уже знаете). Но такой подход нерационален, в первую очередь, из-за его низкого быстродействия, ведь скорость работы с файлами намного ниже скорости работы с оперативной памятью. Ясное дело, что для сохранения информации вам все равно придется работать с файлами, но для передачи «оперативной» информации другому процессу гораздо удобнее использовать каналы.

11.8.2. Управление каналами с помощью языка C

Перенаправление ввода/вывода можно осуществить и программным путем, то есть без вмешательства пользователя. Наша программа сама перенаправит свой ввод/вывод в нужный файл или какой-нибудь программе.

*Вызов **system()** порождает дочерний процесс, позволяя ему читать данные* со стандартного ввода (**stdin**) и писать на стандартный вывод (**stdout**). Иногда нам нужно передать данные дочернему процессу или, наоборот,

получить информацию от порожденного процесса. Другими словами, мы хотим, чтобы дочерний процесс получал данные не со стандартного ввода, а от родительского процесса или/и выводил информацию не на стандартный вывод, а передавал ее процессу-предку. Ввод/вывод между процессами осуществляется с помощью системного вызова **popen()**.

```
FILE * popen(const char * команда, const char * режим_доступа);
```

Первый параметр — это название программы, которую мы хотим запустить (дочерний процесс). Второй параметр определяет режим доступа. Установите значение «r», если вам нужно читать вывод дочернего процесса, если же вам нужно передать информацию на стандартный ввод порожденного процесса, установите значение «w». Вы не можете комбинировать сразу оба режима доступа, то есть нельзя установить режим «wr», позволяющий двусторонний обмен.

Вызов **popen()** возвращает указатель FILE или пустой указатель NULL, если вызов не удался. Так же, как и при работе с обыкновенными файлами, вы должны закрыть канал вызовом **pclose()** после завершения операции ввода/вывода. Во время работы с потоком рекомендую использовать вызов **fflush()** — как и в PHP, чтобы предотвратить задержки из-за буферизации.

Теперь несколько простых примеров. Предположим, что нам нужно вывести на стандартный вывод имена всех текстовых файлов, содержащихся в текущем каталоге. Это можно очень просто сделать с помощью вызова **system()**:

```
system("ls *.txt");
```

Это уж совсем тривиальная задача — мы просто выводим данные, но никак не обрабатываем их. Как получить все имена текстовых файлов и обработать их в программе?

```
// открываем поток
FILE *fp = popen("ls *.txt", "r");

// в цикле читаем имена всех текстовых файлов
while ((fname = fgets(...,fp);) != EOF)
 {
// обрабатываем полученную информацию - fname
 }

// закрываем поток
pclose(fp);
```

Данный код в особых комментариях не нуждается. Сначала мы создаем поток для чтения (доступ «r») информации от порожденного процесс (ls *.txt). Затем в цикле while читаем имена файлов и обрабатываем. Чтение производим до тех пор, пока не будет достигнут конец файла. После окончания операции ввода/вывода закрываем поток вывозом **pclose(fp)**.

Вот теперь мы готовы к тому, чтобы рассмотреть более серьезный пример. В этом примере мы будем передавать данные дочернему процессу. Задача такова: у нас есть две программы. Первая программа (см. листинг 11.15) передает второй какую-нибудь информацию, вторая (см. листинг 11.16) обрабатывает ее и выводит на стандартный вывод результат.

Листинг 11.15. Программа-родитель

```
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{
 char buff[1024]={0};
 FILE * cp; // cp - child process - дочерний процесс
 int status;

 // Открываем канал для записи. Дочерний процесс - /usr/bin/child
 cp=popen("/usr/bin/child", "w");
   if (!cp)
    {
      printf("не могу открыть канал.\n");
      exit(1)
     }

 printf("Введите информацию для передачи дочернему процессу " );

 // читаем ввод пользователя
 fgets(buff, sizeof (buff), stdin);

 // передаем данные дочернему процессу
 fprintf(cp, "%s\n", buff);
 // "выталкиваем" содержимое буфера в канал
 fflush(cp);

 // закрываем канал и проверяем состояние вызова pclose()
 status=pclose(cp);
 if (!WIFEXITED(status))
  printf("ошибка при закрытии канала\n");

 printf("Завершение работы родительского процесса\n");
 return 0;
}
```

Теперь несколько комментариев. Мы используем вызов **popen()**, чтобы установить связь между родительским и дочерним процессами. Если мы не можем этого сделать по какой-либо причине, например, исполнимый файл дочернего процесса `/usr/bin/child` не существует, мы выводим сообщение об ошибке и завершаем работу программы. Потом мы читаем со стандартного ввода родительского процесса информацию для передачи дочернему процессу.

С помощью вызова **fprintf()** мы передаем информацию по открытому каналу порожденному процессу. При этом вызов **fprintf()** используется так же, как и при работе с обыкновенными файлами. Во избежание задержек при передаче данных по каналу мы используем вызов **fflush()** для «выталкивания» данных из буфера в порожденный процесс. После окончания работы с каналом мы его закрываем и проверяем состояние вызова **pclose()**. При невозможности корректного закрытия канала мы выводим соответствующее предупреждение.

Как выглядит дочерняя программа показано в листинге 11.16.

Листинг 11.16. Дочерняя программа

```
#include <stdio.h>
int main()
{
 char buff[1024]={0};
 fgets(buff, sizeof (buff), stdin);
 printf("Прочитал со стандартного ввода: %s\n",buff);
 printf("Завершение работы дочернего процесса\n");
 return 0;
}
```

Как видите, ничего сверхъестественного в ней нет — программа просто читает данные со стандартного ввода и обрабатывает их: выводит на стандартный вывод.

На экране при запуске родительской программы (позаботьтесь о существовании программы `/usr/bin/child`) вы увидите примерно следующее:

```
Введите информацию для передачи дочернему процессу hello
Прочитал со стандартного ввода: hello
Завершение работы дочернего процесса
Завершение работы родительского процесса
```

11.8.3. Управление каналами в PHP

В PHP также имеется функция **popen()**, которая работает аналогично системному вызову **popen()**. Функции **popen()** нужно передать два параметра:

- имя запускаемой программы — дочернего процесса;
- режим работы канала — чтение («r») или запись («w»).

Режим *«чтение»* нужно выбирать, если вы хотите *получить* информацию от дочернего процесса, а *«запись»*, если вы хотите *передать* данные порождаемому процессу. Вы не можете одновременно открыть канал в режиме чтения и записи, поскольку возникнет «чрезвычайное положение» — взаимная блокировка процессов. Чтобы объяснить, что это такое, потребовалось бы не одна страница, поэтому вкратце скажу, что родительский процесс будет ждать данные от дочернего процесса, а дочерний процесс — от родительского. В результате оба процесса будут приостановлены.

Пример использования функции **popen()** приведен в листинге 11.17.

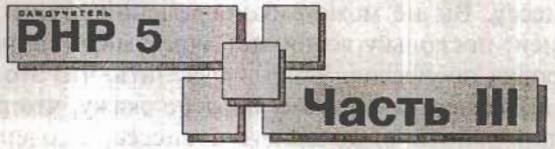
Листинг 11.17. Управление каналами в PHP. Функция popen()

```
<?
// Открываем канал
$p = popen("/bin/ls","r");
$Files = array();

// Читаем вывод программы
for(; !eof($p); ) $Files[] = gets($p,255);

// Закрываем канал
pclose($p);
?>
```

Особенностью вызова **popen()** является то, что дочерний процесс будет выполняться параллельно нашему процессу, и интерпретатор не будет ждать, пока завершится дочерний процесс, а сразу перейдет на следующую строчку нашего сценария.

Теперь систематизируем наши знания о каналах:

- Канал — это связь между родительским и дочерним процессом.
- Каналы можно использовать только в Unix-подобных операционных системах (например, в Linux).
- В отличие от вызова **system()**, программа, запущенная с помощью **popen()**, выполняется параллельно нашей программе, то есть нам не нужно ждать завершения вызванной программы.
- Вывод запущенной с помощью **popen()** программы не дублируется на стандартном выводе (**stdout**).
- После открытия канала мы можем работать с ним как с обыкновенным файлом. Читая из этого файла, мы получим информацию, отправляемую дочерним процессом на **stdout**, а записывая в этот файл, *мы передаем данные на стандартный ввод дочернего процесса (**stdin**).*
- Мы не можем отрыть канал в режиме чтения и записи одновременно, иначе возникнет состояние взаимной блокировки, и оба процесса зависнут.



Расширенные функции PHP

Глава 12. Графические функции PHP

Глава 13. PHP и базы данных MySQL

Глава 14. Сетевые функции

Глава 12

Графические функции PHP

12.1. Основные понятия

12.1.1. Библиотека GD

При написании сценариев на PHP очень часто возникает потребность создания различных рисунков «на лету» исходя из каких-то данных. Данная возможность оказывается очень полезной при написании различных счетчиков, формировании графиков или же просто для создания красивых картинок.

Для того чтобы рядовой Web-программист почувствовал себя еще и художником, нужна библиотека GD. Если вы используете Linux, то, скорее всего, данная библиотека уже будет установлена и PHP будет собран с ее поддержкой. Если же это не так, установите библиотеку GD и пересоберите PHP с поддержкой этой библиотеки. Как это сделать, описано в документации по PHP.

Если вы используете PHP для Windows, вам понадобится всего один файл — `php_gd.dll`. Этот файл нужно скопировать в каталог %WINDIR%\SYSTEM (обычно это `C:\WINDOWS\SYSTEM` или `C:\WINNT\SYSTEM`). Затем добавьте в ваш конфигурационный файл `php.ini` следующую строку:

```
extension=php_gd.dll
```

Скорее всего, данная строка уже будет добавлена в разделе Dynamic Extensions, вам нужно лишь убрать комментарий.

12.1.2. Методика создания графического оформления сайтов в PHP

Как и в любом другом языке программирования, который позволяет работать с графикой, совсем необязательно формировать картинку полностью. Взять самый тривиальный пример: вам нужно написать счетчик, содержащий логотип вашей компании. Логотип довольно замысловатый

и вы даже не знаете как начать его рисовать: то ли сбоку, то ли сверху. Гораздо проще взять готовый логотип, поместить его слева, а справа вывести количество посещений сайта. PHP позволяет сделать вам это: вы смело можете подготовить «прототип» будущего изображения в вашем любимом редакторе: The GIMP, Adobe Photoshop, да хоть в Paint. Потом, в вашем сценарии, вы загружаете изображение, как основу, и можете делать с ним, все, что вам захочется: нарисовать что-нибудь поверх изображения, вывести текст, переместить часть изображения, и т.д. В этой главе вы узнаете почти обо всех функциях графической библиотеки PHP.

12.2. Вывод готовых изображений

После установки библиотеки проверим корректность ее работы. Создайте небольшой сценарий (см. листинг 12.1).

Листинг 12.1. Сценарий test.php

```
<?
$image = ImageCreateFromPng("image.png");
Header("Content-type: image/png");
ImagePng($image);
ImageDestroy($image);
?>
```

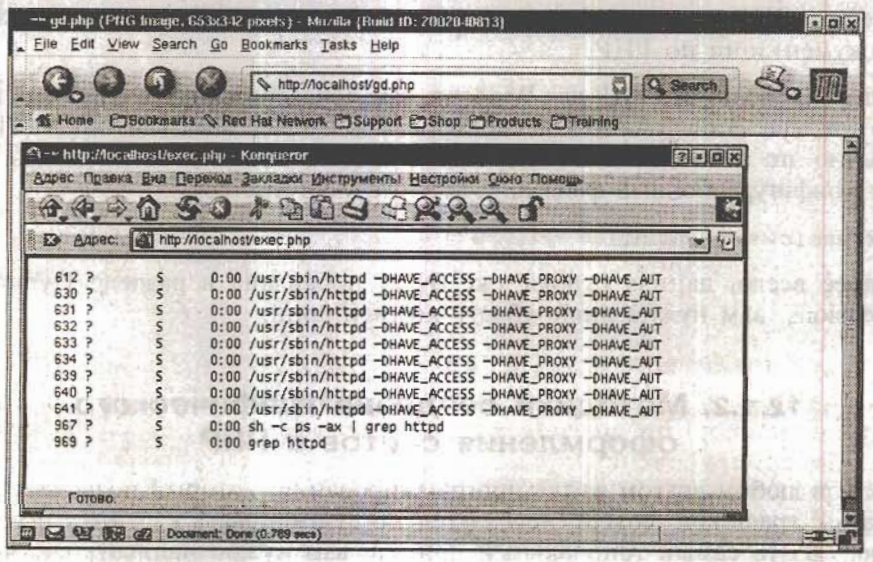


Рис. 12.1. Вывод рисунка в браузер

Использовать данный сценарий очень просто — в нужном месте вашего HTML-документа добавьте тег:

```
<img src=test.php>
```

Перед запуском сценария поместите в каталог, в котором расположен сценарий, файл `image.png`. Сценарий `test.php` не будет изменять наш рисунок `image.png`, он просто скопирует его в браузер. Вместо использования данного сценария мы могли бы просто использовать тег `<img src=image.png>`.

Сценарий `test.php` может использовать только в демонстрационных целях — он демонстрирует несколько важных моментов при работе с рисунками: создание, вывод формата и сохранение рисунка.

Во второй строке мы создаем рисунок `$image` из файла `image.png`. Точнее надо было бы написать «загружаем», а не «создаем». Затем, в третьей строке, мы сообщаем браузеру тип содержимого — рисунок формата PNG (image/png). В следующей строке с помощью функции **ImagePng** мы передаем рисунок на стандартный вывод, то есть в браузер. Функция **ImageDestroy** освобождает память, которая была выделена для рисунка.

Нужно более подробно рассмотреть процессы создания и сохранения рисунков. В качестве прототипа вашего рисунка вы можете использовать файлы других форматов, например, JPEG или GIF, а не только PNG. Для этого вместо функции **ImageCreateFromPng** нужно использовать функции **ImageCreateFromJpeg()** или **ImageCreateFromGif()**.

Итак, для вывода готовых изображений, хранимых в файлах графического формата, в PHP предусмотрены следующие функции:

- **ImageCreateFromPng()** — загружает новое изображение, точнее, создает новое, взявши за основу «прототип» в формате PNG. Это как раз тот случай, о котором мы говорили в самом начале этой главы: вы можете загрузить готовое изображение и потом слегка модифицировать его с помощью PHP. Кроме функции **ImageCreateFromPng()** вы можете использовать функции **ImageCreateFromJpeg()** и **ImageCreateFromGif()**, если вам нужно загрузить изображение из формата JPEG или GIF, соответственно. После загрузки изображения формат роли не играет.
- **ImagePng()** — отправляет рисунок в браузер в формате PNG. Вы также можете использовать функции **ImageGIF()** и **ImageJpeg()**.
- **ImageDestroy()** — освобождает память. Если вы после вывода рисунка не планируете завершить работу сценария, обязательно освободите память, так как рисунок находится в памяти не в сжатом формате (PNG, JPEG, GIF), а в растровом, поэтому может занимать очень много памяти.

12.3. Создание изображений в PHP

Если вы хотите нарисовать картинку «с ноля», не используя заготовку, используйте функцию **ImageCreate(int $x, int $y)**.

После вызова функции **ImageCreate()** будет создана пустая картинка размера X×Y. Дальнейшая работа с рисунком возможна через его идентификатор.

После загрузки изображения с диска производится его распаковка и в дальнейшем изображение хранится в собственном растровом неупакованном формате (подобном формату BMP), что обеспечивает высокую скорость выполнения различных операций. Неупакованный формат требует намного больше памяти, чем сжатый, поэтому после завершения работы с картинкой обязательно освободите память с помощью функции **ImageDestroy()**.

12.4. Графические форматы и библиотека GD

Форматы и их указание

Следует отметить, что ваша библиотека может не поддерживать работу со всеми тремя форматами — PNG, JPEG и GIF. Поддержка формата PNG будет всегда, а вот поддержки формата GIF может не оказаться.

После того, как рисунок уже сформирован, его нужно отправить в стандартный вывод — в браузер. Перед сохранением (выводом) нужно указать, и желательно правильно, формат рисунка. Это можно сделать с помощью функции **Header**:

```
Header("Content-type: image/png"); // если рисунок в формате PNG
Header("Content-type: image/gif"); // если рисунок в формате GIF
Header("Content-type: image/jpeg"); // если рисунок в формате JPEG
```

Почему я написал «желательно правильно»? Потому что некоторые браузеры сами определяют тип картинки. Например, вы отправляете заголовок, в котором сказано, что картинка в формате PNG, а сами выдаете браузеру картинку в формате GIF. Браузер по первым байтам самостоятельно определит тип рисунка и правильно отобразит его. Однако не нужно надеяться, что все браузеры такие умные — многие просто надеются на вашу честность и верят содержимому заголовка.

Кстати говоря, после загрузки изображения в память (функцией **ImageCreateFromPng** или т.п.), в памяти оно хранится в собственном растровом формате и может быть передано браузеру в другом формате, отличном от первоначального. Информация о первоначальном формате нигде не хранится. Таким образом, вы спокойно можете открыть изображение в одном формате, а сохранить — в другом. Пример приведен в листинге 12.2.

Листинг 12.2. Конвертация графических форматов. Сценарий convert.php

```
<?
 $image = ImageCreateFromPng("image.png");
 Header("Content-type: image/jpeg");
 ImageJpeg($image);
 ImageDestroy($image);
?>
```

Этот сценарий преобразует картинку из формата PNG в формат JPEG.

12.5. Определение размера картинки

После загрузки изображения полезно было бы знать его размеры (ширину и высоту), чтобы при выводе текста или графических примитивов не промахнуться: ваш рисунок имеет ширину всего лишь 100 пикселей, а вы пытаетесь вывести на него текст, занимающий 120 пикселей.

При создании пустой картинки размер изображения нам уже известен, а вот при загрузке рисунка его точный размер нам неизвестен. Для определения горизонтального размера можно воспользоваться функцией:

ImageSX(resource $image) — возвращает ширину изображения `$image`.

Аналогично, для определения вертикального размера используется функция:

ImageSY(resource $image) — возвращает высоту изображения `$image`.

12.6. Графические примитивы

Вам уже надоело рассматривать различные форматы и хочется что-нибудь нарисовать? Не спешите! Перед тем, как нарисовать какой-нибудь примитив (линию, эллипс или дугу) нам нужно выбрать цвет и создать его. Выбрать — это означает решить, каким цветом вы будете рисовать, а создать — это означает задать тройку RGB-значений.

Тройка RGB (Red, Green, Blue) значений — это три числа (от 0 до 255), которые определяют содержание красной, зеленой и синей составляющих в создаваемом цвете. Число 0 означает минимальную яркость, а 255 — максимальную. Например, тройка (0, 0, 0) задает черный цвет, а (255, 255, 255) — белый.

Может возникнуть вопрос. А зачем необходимо задавать цвет, когда можно работать прямо с тройкой RGB-значений? Этому есть причина и GD не может работать с такой тройкой напрямую. Каждой тройке должен быть определен ее специальный идентификатор.

12.6.1. Создание (определение) цвета

Задание нового цвета в палитре рисунка

Как было сказано выше, вы не можете работать с цветами напрямую, например, выбрать примитив и указать цвет, каким он должен быть нарисован — red, yellow, blue, white, как это возможно в других языках. Вам нужно создать новый цвет в палитре рисунка или выбрать уже существующий цвет из палитры. Работать с цветом можно только через его идентификатор, аналогично тому, как мы работаем с рисунком.

Палитра — это набор цветов, которые использованы в рисунке. Некоторые форматы, например, GIF, поддерживают лишь небольшое количество цветов в рисунке (а именно 256).

Кстати говоря, чем меньше количество цветов, тем меньше рисунок занимает памяти. При небольшом размере палитры рисунок очень хорошо сжимается, а при переходе через степень двойки, эффективность сжатия снижается. Если ваш рисунок хорошо сжимался при 32 цветах, то если добавить хотя бы еще один цвет, размер файла рисунка будет заметно увеличен.

Создать новый цвет в палитре можно с помощью функции **ImageColor Allocate(resource $image, int $red, int $green, int $blue).**

При этом данная функция возвращает идентификатор цвета, заданного тройкой `$red`, `$green`, `$blue`. Таким образом, каждый вызов функции **ImageColorAllocate()** увеличивает число цветов в палитре на единицу (в палитру добавляется новый цвет).

Определять все нужные цвета я рекомендую в самом начале сценария. При этом желательно, чтобы идентификаторы цветов несли смысловую нагрузку, например, `$white`, `$black`, а не `$c0`, `$c1`.

Подбор ближайшего цвета из палитры

Может получиться так, что с новым цветом палитра будет переполнена. Например, если вы работаете с изображением в формате GIF, то у него может быть всего 256 цветов. Что же произойдет, если достигнут предел — 256 цветов? Тогда функция **ImageColorAllocate()** попытается найти в палитре цвет, который, по ее мнению, ближе всех к запрашиваемому цвету.

Таким образом, функция **ImageColorAllocate()** действует примерно следующим образом: если есть свободные места в палитре, то создается новый цвет и добавляется в палитру, а если свободных мест нет, то производится поиск похожего цвета.

Однако в PHP предусмотрена специальная функция, которая изначально не задает новые цвета в палитре, а сразу ищет цвет, похожий на ука-

занный. Такой функцией является функция **ImageColorClosest(resource $image, int $red, int $green, int $blue)**, которая возвращает похожий цвет из уже имеющихся цветов в палитре. Эту функцию рекомендуется использовать, если вы не хотите, чтобы палитра рисунка разрасталась. При этом функцию **ImageColorAllocate()** следует использовать лишь в исключительных случаях, когда цвет нужно задать абсолютно точно или в палитре не содержится нужного оттенка.

Узнать текущее количество цветов в палитре можно с помощью функции **ImageColorsTotal(resource $image)**. Данная функция возвращает количество цветов в палитре изображения.

Эффект прозрачности

Довольно часто нужно задать прозрачный цвет для некоторых частей рисунка. При этом прозрачный цвет не будет отображен в браузере и вместо них будет отображены части фона документа HTML.

Определить прозрачный цвет можно с помощью функции **ImageColor Transparent(resource $image, int $color)**. После вызова этой функции цвет `$color` (заданный идентификатором) в изображении `$image` будет считаться прозрачным. Например, у нас есть изображение на красном фоне и нам нужно сделать красный цвет прозрачным. Это можно сделать вызовом таких операторов:

```
$red = ImageColorClosest($image,255,0,0);
// Мы используем функцию ImageColorClosest, потому
// что красный цвет уже есть в палитре
ImageColorTransparent($image,$red);
```

Помните, что не все форматы поддерживают прозрачность, например, JPEG прозрачность не поддерживает.

Вот теперь мы можем приступить к рисованию графических примитивов.

12.6.2. Рисование графических примитивов в PHP

Рисование точки

Простейшим примитивом является точка. Нарисовать ее можно с помощью функции **ImageSetPixel(resource $image, int $x, $y, int $color)**. Эта функция нарисует одну точку цвета `$color` с координатами (`$x`, `$y`).

Если вам нужно узнать цвет какой-нибудь точки изображения, используйте функцию **ImageColorAt(resource $image, int $x, int $y)**. Данная функция похожа на инструмент **Пипетка** во многих графических пакетах.

Рисование линий

Нарисовать линию можно функциями:

```
ImageLine(resource $image, int $x1, int $y1, int $x2, int $y2,
int $color);
ImageDashedLine(resource $image, int $x1, int $y1, int $x2, int
$y2, int $color);
```

Первая функция рисует обыкновенную линию, а вторая — пунктирную. Линия проходит через точки ($x1, $y1) и ($x2, $y2). Цвет, которым будет нарисована линия, определяется идентификатором $color.

Рисование прямоугольников

Нарисовать прямоугольник можно, вызвав функцию

```
ImageFilledRectangle(resource $image, int $x1, int $y1, int
$x2, int $y2, int $color);
```

Данная функция нарисует закрашенный прямоугольник в изображении $image. Размеры прямоугольника задаем параметрами ($x1, $y1), ($x2, $y2). Первая пара определяет координаты верхнего левого угла, а вторая — правого нижнего. Цвет, которым будет закрашен прямоугольник, задается идентификатором $color.

Для рисования незакрашенных прямоугольников в PHP используется функция **ImageRectangle(resource $image, int $x1, int $y1, int $x2, int $y2, int $color)**, которая рисует незакрашенный прямоугольник с толщиной границы 1 пиксель и заданным цветом.

Рисование многоугольников

С помощью функции **ImagePolygon(int $image, list $dots, int $num, int $color)** на изображении $image можно нарисовать многоугольник. При этом рисуется многоугольник, заданный своими вершинами. Количество вершин указывается параметром $num. Координаты вершин передаются в массиве $dots и делается это следующим образом:

```
$dots[0]=x0; $dots[1]=y0; $dots[2]=x1; $dots[3]=y1; и т.д.
```

Если вам нужно нарисовать закрашенный многоугольник, используйте функцию **ImageFilledPolygon (resource $image, list $dots, int $num, int $color)**, параметры которой идентичны предыдущей функции.

Рисование эллипсов и дуг

Для изображения дуги сектора используйте функцию **ImageArc(resource $image, int $x, int $y, int $width, int $height, int $s_a, int $e_a, int $col)**. Эта функция рисует дугу сектора эллипса от угла $s_a (start angle) до угла

$e_a (end angle). Углы отсчитываются в градусах против часовой стрелки от горизонтали. Сам эллипс должен вписаться в прямоугольник, заданный параметрами:

- $x, $y — координаты верхнего левого угла;
- $width — ширина;
- $height — высота.

Фигура не закрашивается, а цветом $color обводится ее контур.

Понятно, что, чтобы нарисовать эллипс, а не дугу, следует в качестве значений $s_a и $e_a указать 0 и 360.

Заполнение области

Закрасить произвольную одноцветную область, которая содержит точку ($x, $y), цветом $color, можно функцией **ImageFill(resource $image, int $x, int $y, int $color)**. Данная функция произведет сплошную заливку одноцветной области.

Если вам нужно закрасить не только одноцветные точки, а любые, то используйте функцию **ImageFillToBorder(resource $image, int $x, $y, int $border, int $color)**. Эта функция будет закрашивать все точки, пока не будет достигнута граница цвета $border.

Рис. 12.2. Концентрические квадраты

В качестве примера рассмотрим сценарий (листинг 12.3), который я использовал для создания рис. 12.2.

10 Зак. 623

Листинг 12.3. Рисование концентрических квадратов

```
<?
// Создаем рисунок 200x200
$im = ImageCreate(200,200);
// первый размещенный цвет - это цвет фона
// RGB(0,0,0) - это черный цвет
$black = imagecolorallocate($im,0,0,0);
// Создаем массив $c[] и заполняем его цветами
$c[] = imagecolorallocate($im,255,0,0);     // красный
$c[] = imagecolorallocate($im,0,255,0);     // зеленый
$c[] = imagecolorallocate($im,0,0,255);     // синий
$c[] = imagecolorallocate($im,255,0,255);
$c[] = imagecolorallocate($im,100,0,100);
$cl = 1; // индекс массива $c[]

// Рисуем концентрические квадраты
for ($i = 1; $i < 7; $i++)
{
 $cl = $i;
 if ($cl>5) $cl = 1; // если цвет > 5, то цвет = красный (1)
 ImageRectangle($im,10*$i,10*$i,200-10*$i,200-10*$i,$c[$cl-1]);
 }

// Сообщаем браузеру формат картинки
Header("Content-type: image/png");
// Выводим изображение в формате PNG
ImagePng($im);
// Освобождаем память
ImageDestroy($im);

?>
```

12.7. Вывод текста в изображении

Вывод строки текста

Зачастую, например, при написании сценария-счетчика, возникает необходимость вывести какую-нибудь строку в изображение. В PHP это можно сделать, используя функцию **ImageString(resource $image, int $font, int $x, int $y, string $s, int $color)**.

После вызова этой функции в изображение `$image` будет выведена строка `$s`. Будут использованы шрифт `$font` и цвет `$color`. Координаты верхнего левого угла задаются параметрами `$x` и `$y`. Стандартный шрифт задается числом от 1 до 5 (1 — меньший шрифт, 5 — больший). Как использовать нестандартные шрифты, описано в следующем разделе.

Задание собственного шрифта

Вы можете использовать один из встроенных шрифтов — их номера 1...5 или загрузить свой шрифт с помощью функции **ImageLoadFont(string $file)**, где `$file` — имя файла со шрифтом. При этом номер шрифта будет больше 5.

Файлы, в которых содержатся шрифты — бинарные и имеют следующую структуру, представленную в табл. 12.1.

Формат файла шрифтов *Таблица 12.1*

Смещение от начала файла (в байтах)	Тип	Описание
0...3	long	Количество символов в шрифте
4...7	long	Индекс первого символа шрифта (обычно это пробел, 32)
8...11	long	Ширина каждого знака в пикселях – W
12...15	long	Высота каждого знака в пикселях – H
16...EOF	array	Массив, содержащий информацию о каждом символе – по одному байту на пиксель. На один символ приходится WxH байтов

Вывод строки текста в вертикальном направлении

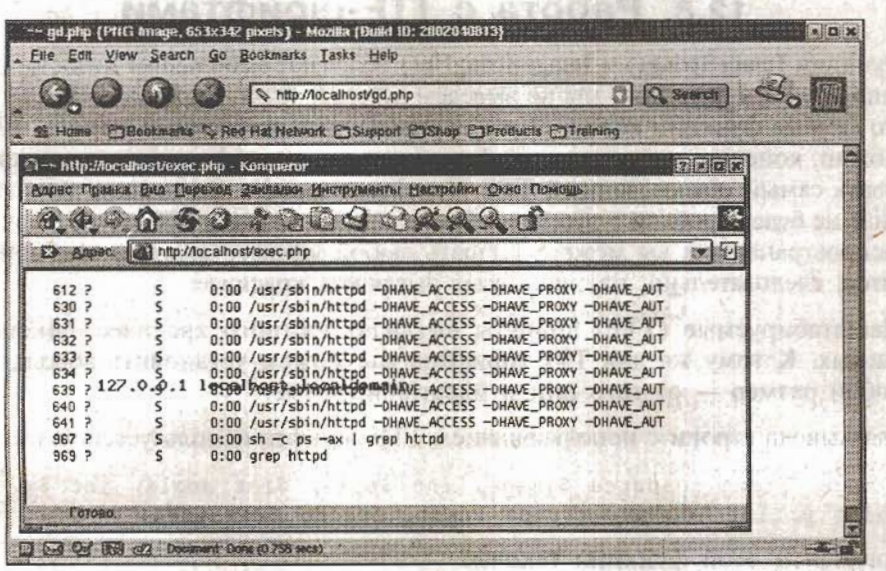


Рис. 12.3. *Вывод текста*

Для вывода текста (рис. 12.3) я использовал сценарий из листинга 12.4.

Листинг 12.4. Вывод текста в изображении

```
<?
$im = ImageCreateFromPng("./11-1.png");
$color = imagecolorallocate($im,255,0,0);
// красным цветом - цвет шрифта

// Наша строка - IP-адрес клиента + имя сервера
$str = "$REMOTE_ADDR $SERVER_NAME";

// Выводим строку $str в картинку $im, шрифт - 5,
// координаты начала строки (70,200)
// цвет - $color (красный)

imagestring($im,5,70,200,$str,$color);

Header("Content-type: image/png");
ImagePng($im);
ImageDestroy($im);
?>
```

Вывести строку в вертикальном направлении можно функцией **ImageStringUp**, которая обладает такими же параметрами что и функция ImageString().

12.8. Работа с TTF-шрифтами

Функции **ImageString()** и **ImageStringUp()** обладают небольшим недостатком: используя эти функции, мы не выведем русский текст. Это объясняется тем, что данные функции используют фиксированные встроенные шрифты PHP. Можно, конечно, найти русский фиксированный шрифт, но проще использовать самый обыкновенный TTF (TrueType) шрифт. Во-первых, при этом у нас не будет проблем с русскими буквами, а во-вторых, TTF-шрифты очень распространены и вы можете выбрать любой шрифт, который вам понравится, следовательно, надпись будет выглядеть красивее.

Масштабируемые (TTF) шрифты выглядят намного красивее фиксированных. К тому же для TTF-шрифтов мы можем установить абсолютно любой размер — от того они и масштабируемые.

Для вывода строки с использованием TTF-шрифта используется функция:

```
ImageTTFtext(resource $image, int $size, $int angle, int $x,
$int y, int $color, string $font, string $message).
```

Аргументы этой функции таковы:

- `$image` — идентификатор картинки, в которую выводится текст;
- `$size` — размер шрифта, который будет использоваться при выводе строки;

- $angle — угол наклона выводимой строки;
- $x, $y — координаты левого нижнего угла выводимой строки;
- $color — идентификатор цвета;
- $font — полное имя файла шрифта;
- $message — выводимая строка.

Практически все параметры не нуждаются в объяснении, кроме параметров $angle, $font и $message. Угол $angle задается в градусах и отсчитывается против часовой стрелки от горизонтали. Имя файла шрифта должно быть полным, то есть нужно указывать полный путь к нему, например, /var/www/html/ttf/arial.ttf. Если шрифт находится в том же каталоге, что и сценарий, обычно удобно использовать функцию **getcwd()**, возвращающую текущий каталог.

Сообщение $message должно быть в кодировке Windows 1251, потому что большинство TTF-шрифтов используют именно эту кодировку.

В листинге 12.5 приведен небольшой пример вывода русской строки.

Листинг 12.5. Вывод русской строки

```
<?
// Создаем картинку 300x400
$im = imagecreate(300,400);
// Определяем белый цвет
$white = imagecolorallocate($im, 255,255,255);
// Определяем красный цвет
$red = imagecolorallocate($im, 255,0,0);

// Это наша строка
$str = "Русский текст";
// Преобразовываем строку из кодировки KOI8-R в
// кодировку CP1251
// Вам не нужно этого делать, если вы работаете под Windows
$str = convert_cyr_string($str,"k","w");

// Выводим строку с использованием шрифта assuan.ttf,
// размер - 24 пункта,
// цвет - красный, координаты нижнего левого
// угла строки - (20,50)
imagettftext($im, 24,0,20,50,$red,getcwd()."/assuan.ttf",$str);

Header("Content-type: image/png");
// Выводим картинку в формате PNG
ImagePng($im);
?>
```

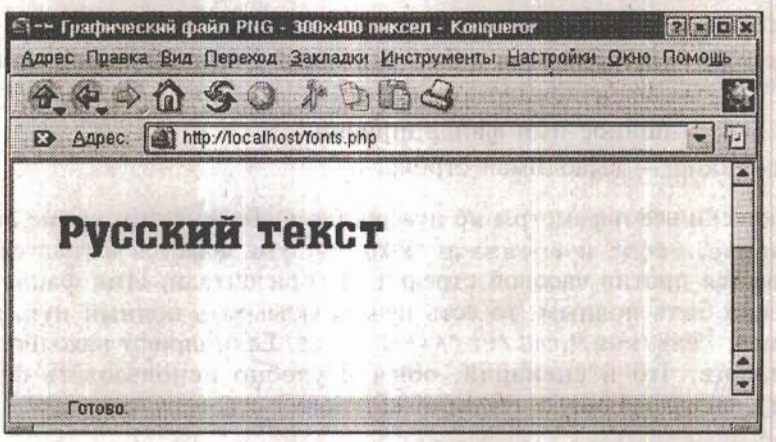


Рис. 12.4. *Вывод текста с использованием TTF-шрифта*

Рис. 12.5. *Вот что бывает, если программист не преобразовал кодировку*

Помимо того, что функция **imageTTFtext()** выводит текст, она в качестве своего значения возвращает массив-список, содержащий координаты прямоугольника, описанного вокруг строки (см. табл. 12.2).

Результат функции imageTTFtext() *Таблица 12.2*

Индексы массива	Содержание
0, 1	Координаты (x, y) левого верхнего угла
2, 3	Координаты (x, y) правого верхнего угла
4, 5	Координаты (x, y) правого нижнего угла
6, 7	Координаты (x, y) левого нижнего угла

Если вы часто работаете с TTF-шрифтами, вы должны оценить функцию **ImageTTFBox(int $size, int $angle, string $font, string $message)**, которая позволяет вычислить размер строки, без вывода ее на изображение.

Функции нужно передать четыре параметра:

- Размер шрифта.
- Угол наклона.
- Файл шрифта (полное имя).
- Строку.

Функция **ImageTTFBox()** возвращает массив-список, который содержит информацию о размере выводимой строки – координаты прямоугольника, описанного вокруг строки (табл. 12.3).

Результат функции imageTTFbox() *Таблица 12.3*

Индексы массива	Содержание
0, 1	Координаты (x, y) левого нижнего угла
2, 3	Координаты (x, y) правого нижнего угла
4, 5	Координаты (x, y) правого верхнего угла
6, 7	Координаты (x, y) левого верхнего угла

```
$Rect = imageTTFbox(24,0,"/var/www/html/ttf/times.ttf",$str);
foreach ($Rect as $v) echo $v;
```

Не понимаю, из каких соображений исходили разработчики PHP: если бы совпадали форматы списков, возвращаемых функциями **ImageTTFtext()** и **ImageTTFbox()**, работать с ними было бы гораздо проще.

12.9. Копирование участков изображения

Нам осталось рассмотреть очень мощную функцию, которая «умеет» копировать участки изображения и целые изображения, масштабировать участки изображения, копировать и масштабировать участки изображения в пределах одного рисунка.

ImageCopyResized(resource $dest, resource $src, int $destX, int $destY, int $srcX, int $srcY, int $destW, int $destH, int $srcW, int $srcH);

Выглядит просто ужасающе. Однако все намного проще, когда разберешься, что здесь и к чему. Параметры `$dest` и `$src` задают рисунок-результат и рисунок-источник. Эти параметры могут совпадать.

Параметры `$srcX`, `$srcY`, `$srcW`, `$srcH` определяют область исходного изображения, над которой будет произведена операция.

Параметры `$destX`, `$destY`, `$destW`, `$destH` задают область изображения-результата, в которую будет «вписан» прямоугольник, определенный предыдущими параметрами. Если ширина и высота прямоугольников не совпадают, то изображение-результат будет сжато или растянуто в соответствии с параметрами.

В PHP есть еще одна функция, позволяющая копировать части изображения. Это функция **ImageCopy()**:

imagecopy (resource $dest, resource $src, int $dest_x, int $dest_y, int $src_x, int $src_y, int $src_w, int $src_h)

Функция **ImageCopy()** гораздо проще в использовании, чем предыдущая, но и менее универсальна. **ImageCopy()** копирует часть изображения `$src` в изображение `$dest`, начиная с координат `$src_x`, `$src_y` изображения `$src`, шириной и высотой `$src_w` и `$src_h`, соответственно. Скопированное изображение вставляется в изображение $dest, начиная с координат `$dest_x` и `$dest_y`.

Следует заметить, что в этой главе рассмотрены далеко не все графические функции, но и их будет вполне достаточно. Нам же нужно всего лишь построить какой-нибудь график или вывести информацию о посещениях нашего узла, а для создания более сложной графики разработано множество специальных пакетов — тот же GIMP, Photoshop, и т.п.

Глава 13

PHP и базы данных MySQL

13.1. Общие положения

База данных, таблицы записи

База данных — это совокупность связанных двухмерных таблиц, содержащих определенную информацию. Программное обеспечение, которое управляет базой данных (создает новые базы данных, таблицы, а также позволяет использовать уже созданные объекты), называется системой управления базой данных (СУБД). В это книге рассмотрены две базы данных: самая популярная — MySQL, и база данных SQLite (см. гл. 32), поддрежка которой включена в PHP 5. В PHP есть средства для работы с другими СУБД (Oracle, Sybase), но в большинстве случаев, если вы не разрабатываете что-нибудь действительно грандиозное, возможностей MySQL вам вполне хватит.

Стоит сразу же оговориться, что эта глава не претендует на полноту информации: СУБД MySQL и языку SQL посвящены целые тома, объемы которых превышают эту книгу. Однако для написания несложных сценариев работы с базой данных приведенного материала будет достаточно.

Клиентская и серверные части MySQL

Начнем с клиентской части СУБД MySQL. База данных в MySQL — это обыкновенный каталог (обычно, это подкаталог каталога `/var/lib/mysql` — в Linux или `c:\mysql\data` — в Windows). Таблица физически представляется файлами с расширениями `frm`, `MYD`, `MYI`. Логически — таблица является совокупностью записей. В свою очередь, запись — это совокупность полей разного типа (тип полей может повторяться). Имя базы данных уникально в пределах системы, таблицы — в пределах базы данных, а поля — в пределах таблицы.

Удобной программой для просмотра структуры базы данных является **mysqlshow**. Введите следующую команду:

```
mysqlshow -p mysql
```

Вы увидите список таблиц, которые находятся в базе данных `mysql`:

```
Database: mysql
+-------- +
| Tables |
 +--------+
| db    |
| host  |
| user  |
+--------+
```

Программа **mysqlshow** может вызываться с дополнительными параметрами, указанными в табл. 13.1.

Параметры программы mysqlshow *Таблица 13.1*

Параметр	Описание
--host=hostname	Задает имя хоста, к которому вы хотите подключиться
--port=port_number	Определяет номер порта для сервера MySQL
--socket=socket	Указывает сокет
--user=username	С помощью этого параметра можно указать нужное имя пользователя
-p	Запрашивает ввода пароля

Для самих же операций с данными используется программа **mysql**. Она и является клиентом сервера. В этой программе можно использовать те же опции, что и **mysqlshow**. Среди многочисленных параметров программа **mysql** имеет один очень важный параметр "-s". Я рекомендую вам всегда его использовать. Этот параметр подавляет большинство ненужных сообщений, выводимых клиентом. На медленных линиях связи это должно повысить производительность. Да и наблюдать за всеми рамочками и ненужными сообщениями особо не хочется.

Создать базу данных можно с помощью программы **mysqladmin**. Естественно, что использовать эту программу может только администратор, например:

```
mysqladmin -u admin -p create my_db
```

Как работает сервер SQL? Клиент посылает запрос, в котором он указывает, какую информацию он хочет получить от сервера, или какую операцию с данными он хочет выполнить, сервер посылает клиенту ответ, в котором указывается, выполнил ли сервер запрос клиента, и, если сервер выполнил запрос, результат запроса.

Язык SQL

Для описания запросов клиента был разработан целый язык — SQL (Structured Query Language — структурированный язык запросов). С помощью запросов SQL вы можете:

- Создавать базы данных и таблицы.
- Добавлять информацию в таблицы.
- Удалять информацию.
- Модифицировать информацию.
- Получать нужные вам данные.

Естественно, пользователь `admin`, кроме того, что должен существовать, должен обладать соответствующими правами. Каждый запрос MySQL должен заканчиваться точкой с запятой. Если вы введете `SELECT * FROM test` клиент **mysql** будет ждать ввода точки с запятой:

```
->
```

Вот теперь можно перейти к следующему пункту, в котором рассматривается язык запросов SQL.

13.2. Основы SQL

Давайте договоримся, что будем писать запросы согласно стандарту SQL, то есть для улучшения восприятия будем разбивать их на части. Программа **mysql** допускает ввод запроса во всю строчку. Например, запрос, записанный в соответствии со стандартом SQL:

```
SELECT *
FROM S
WHERE Q > 10;
```

в программе **mysql** можно записать так (в конце каждого запроса должна стоять точка с запятой):

```
SELECT * FROM S WHERE Q > 10;
```

13.2.1. Базовая терминология

Если вы раньше работали с какой-нибудь СУБД (Система Управления Базой Данных), вы смело можете пропустить этот пункт — ничего нового для себя вы не прочитаете. Но если вы сталкиваетесь с СУБД впервые, без основных терминов вам не обойтись.

Поле (field) — это базовый элемент данных в БД. Поле имеет имя и тип. Например, поле `Tovar_No` и обозначающее количество товара, скорее всего, будет целым. Подробнее о типах полей мы поговорим чуть позже.

Запись (record) — это набор полей, содержащих связанную информацию. Например, запись с полями `C_No`, `C_Name` и `C_Address` содержит информацию о клиенте — его номер, имя и адрес.

Таблица (table) — это набор записей одной структуры полей. Если у нас есть запись структуры `C_No`, `C_Name`, `C_Address`, то все записи в таблице Clients будут иметь такую структуру.

База данных (database) — это совокупность связанных таблиц. Например, в одной таблице может храниться информация о клиенте, а в другой — информация о заказе, который сделал клиент, и т.д.

Индекс (index) — используется для быстрого поиска нужной записи в базе данных. Обычно поиск производится по значению одного поля или по значению нескольких полей.

Первичный индекс (index) — управляет порядком отображения записей в таблице. Поле первичного индекса должно быть уникальным, то есть в одной таблице не должно быть два поля (по которому строился первичных индекс) с одним и тем же значением. В нашей таблице `Clients`, очевидно, первичный индекс должен строиться по полю `C_No` — по номеру клиента, который должен быть уникален.

Вторичный индекс (secondary index) — в отличие от первичного индекса, может строиться по нескольким полям и необязательно должен быть уникальным. Вторичные индексы используются для связывания таблиц (об этом мы поговорим позже).

Запрос (query) — оператор, выбирающий записи и поля из одной или нескольких таблиц. При этом выбираемые записи и поля должны соответствовать условию, заданному оператором.

Индексы также называются **ключами**. Соответственно, первичный индекс называется **первичным ключом**, а вторичный — **вторичным ключом**.

13.2.2. Поля и их типы

Мы уже знаем, что база данных с точки зрения MySQL (и некоторых других СУБД) — это обыкновенный каталог, содержащий бинарные файлы определенного формата — таблицы. Таблицы состоят из записей, а записи, в свою очередь, состоят из полей. Поле имеет два атрибута — имя и тип.

Тип поля может быть:

- Целым.
- Вещественным.

- Строковым.
- Бинарным.
- Дата и время.
- Перечисления и множества.

Возможные типы данных, диапазоны и описания представлены в соответствующих таблицах (табл. 13.2...13.6).

Целочисленные типы данных *Таблица 13.2*

Тип	Диапазон
TINYINT	–128... +127
SMALLINT	–32768... +32767
MEDIUMINT	–8 388 608... +8 388 607
INT	–2 147 483 648... –2 147 483 647
BIGINT	–9 223 372 036 854 775 808 ... ±9 223 372 036 854 775 807

Вещественные типы записываются в виде:

```
ТИП (ДЛИНА, ЗНАКИ) [UNSIGNED]
```

ДЛИНА — это количество знакомест, в которых будет размещено все число при его передаче, а ЗНАКИ — это количество знаков после десятичной точки, которые будут учитываться. Если указан модификатор UNSIGNED, знак числа учитываться не будет.

Вещественные числа *Таблица 13.3*

Тип	Описание
FLOAT	Небольшая точность
DOUBLE	Двойная точность
REAL	То же, что и DOUBLE
DECIMAL	Дробное число, хранится в виде строки
NUMERIC	То же, что и DECIMAL

Любая строка — это массив символов. При поиске с помощью оператора SELECT не учитывается регистр символов: строки "HELLO" и "HElLo" считаются одинаковыми.

Можно настроить базу данных на автоматическое перекодирование символов — в этом случае в базе данных строки будут храниться в одной кодировке, а выводиться — в другой.

В большинстве случаев вас устроит тип VARCHAR или просто CHAR, позволяющий хранить строки, содержащие до 255 символов. В скобках после типа указывается длина строки:

```
VARCHAR(20);
CHAR(20);
```

Если же 255 символов для вашей задачи — это слишком мало, можно использовать другие типы, например, TEXT.

Строки *Таблица 13.4*

Тип	Описание
TINYTEXT	Максимальная длина строки — 255 символов
TEXT	Максимальная длина строки — 65535 символов (64 Кб)
MEDIUMTEXT	Максимальная длина строки — 16 777 215 символов
LONGTEXT	Максимальная длина строки — 4 294 967 295 символов

Бинарные данные также можно использовать для хранения текста, но при поиске будет учитываться регистр символов. К тому же, любой текстовый тип можно превратить в бинарный, указав модификатор BINARY:

```
VARCHAR(20) BINARY;
```

Бинарные типы данных *Таблица 13.5*

Тип	Описание
TINYBLOB	Максимум — 255 символов
BLOB	Максимум — 65535 символов
MEDIUMBLOB	Максимум — 16 777 215 символов
LONGBLOB	Максимум — 4 294 967 295 символов

Примечание.

Бинарные данные не перекодируются на лету, если установлена перекодировка символов.

Дата и время *Таблица 13.6*

Тип	Описание
DATE	Дата в формате ГГГГ-ММ-ДД
TIME	Время в формате ЧЧ:ММ:СС
TIMESTAMP	Дата и время в формате timestamp, однако при отображении оно выводится в виде ГГГГММДДЧЧММСС
DATETIME	Дата и время в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС

Другие типы данных MySQL рассматривать мы не будем, поскольку их использование в PHP нецелесообразно.

13.2.3. Создание таблиц

Создать таблицу позволяет оператор CREATE. Вот его синтаксис:

```
CREATE TABLE Имя_таблицы
(
Имя_поля1   Тип   Модификатор,
...
Имя_поляN   Тип   Модификатор,
[Первичный ключ]
[Внешний ключ]
)
```

Кстати говоря, с помощью оператора CREATE можно создавать и другие объекты, но в этой книге мы не будем их рассматривать.

Примечание.

Подробно о ключах мы поговорим позже, а пока будем использовать таблицы без ключей.

В качестве модификаторов можно использовать следующие значения:

- NOT NULL — поле не может содержать неопределенного значения (NULL), то есть поле должно быть явно инициализировано;
- PRIMARY KEY — поле будет первичным ключом — идентификатором записи, по которому можно однозначно идентифицировать запись;
- AUTO_INCREMENT — при вставке новой записи значение этого поля будет автоматически увеличено на единицу, поэтому в таблице никогда не будет двух записей с одинаковым значением этого поля;
- DEFAULT — задает значение, которое будет использовано по умолчанию, если при вставки записи поле не будет инициализировано явно. Значение по умолчанию задается так:

Имя_поля	Тип	DEFALUT	Значение, например
NO	INT	DEFAULT	0,
NAME	INT	DEFAULT	'Иванов'

Теперь создадим три таблицы — "Товар", "Клиенты" и "Заказы".

```
CREATE TABLE CLIENTS
(
C_NO    int     NOT NULL,
FIO     char(40) NOT NULL,
ADDRESS char(30) NOT NULL,
CITY    char(15) NOT NULL,
PHONE   char(11) NOT NULL
);
```

Таблица CLIENTS содержит поля C_NO (номер клиента), FIO (Фамилия, Имя, Отчество), ADDRESS (Адрес), CITY (Город) и PHONE (Телефон). Все эти поля не могут содержать пустого значения (NOT NULL).

```
CREATE TABLE TOVAR
(
T_NO    int            NOT NULL,
DSEC    char(40)       NOT NULL,
PRICE   double(9,2)    NOT NULL,
QTY     double(9,2)    NOT NULL
);
```

Эта таблица будет содержать данные о товарах. Тип double(9,2) означает, что 9 знаков относим под целую часть, и два — под дробную. QTY — это количество товара на складе.

```
CREATE    TABLE          ORDERS
(
O_NO      int            NOT NULL,
DATE      date           NOT NULL,
C_NO      int            NOT NULL,
T_NO      int            NOT NULL,
QUANTITY  double(9,2)    NOT NULL,
AMOUNT    double(9,2)    NOT NULL
);
```

Данная таблица содержит сведения о заказах — номер заказа (O_No), дату заказа (DATE), номер клиента (C_NO), номер товара (T_NO), количество (QUANTITY) и сумму всего заказа AMOUNT (то есть AMOUNT = T_NO * TOVAR.PRICE).

13.2.4. Добавление данных в таблицу

Для добавления записей используется оператор INSERT:

```
INSERT INTO Имя_таблицы [(Список полей)]
VALUES (Список констант);
```

После выполнения оператора INSERT будет создана новая запись, в качестве значений полей будут использованы соответствующие константы, указанные в списке VALUES.

Теперь добавим данные в наши таблицы. Добавить данные можно с помощью оператора INSERT. Рассмотрим использование оператора INSERT:

```
INSERT INTO CLIENTS
VALUES (1,'Иванов И.П.', 'Ленина 6', 'Кировоград','80522111111');
```

Добавляемые значения должны соответствовать тому порядку, в котором поля перечислены в операторе CREATE. Если вы хотите добавлять информацию в другом порядке, то вы должны указать этот порядок в операторе INSERT:

```
INSERT INTO CLIENTS (FIO,ADDRESS,C_NO,PHONE,CITY)
VALUES ('Петров', 'Пушкина 9',2,'-','Кировоград');
```

С помощью INSERT мы можем добавлять данные и в определенные поля, например, C_NO и FIO:

```
INSERT INTO CLIENTS (C_NO, FIO)
VALUES (1,'Петров');
```

Однако в нашем случае сервер не выполнит такой запрос, поскольку все остальные поля равны NULL (пустое значение), а наша таблица не принимает пустые значения. Аналогично можно добавить данные в другие таблицы.

Добавим данные в таблицу TOVAR:

```
INSERT INTO TOVAR
VALUES (1,'Монитор LG',550.74);
```

Обратите внимание, что мы пока еще не указали первичные ключи таблицы, поэтому нам никто не мешает добавить в таблицу одинаковые записи. Добавить дату в поле DATE можно с помощью функции TO_DATE:

```
INSERT INTO ORDERS
VALUES (1,TO_DATE('01/01/02,'DD/MM/YY'),1,1,1,550.74);
```

Данная запись означает, что первого января 2002 года Иванов И.П. (C_NO=1) заказал один (QUANTITY=1) монитор LG (T_NO=1).

13.2.5. Обновление записей

Синтаксис оператора UPDATE, который используется для обновления записей, выглядит так:

```
UPDATE Имя_таблицы
SET Поле1 = Значение1, ..., ПолеN = ЗначениеN
[WHERE Условие];
```

Если не задано условие WHERE, будет модифицирована вся таблица, а это может повлечь за собой непредсказуемые последствия, поскольку для всех записей будут установлены одинаковые значения полей, поэтому всегда указывайте условие WHERE.

Предположим, что нам нужно обновить запись, если, например, клиент Иванов переехал в другой город и нам нужно отметить это событие в нашей базе данных. Это делается так:

```
UPDATE CLIENTS
SET CITY = 'Киев'
WHERE C_NO = 1;
```

Данный запрос нужно понимать так: найти запись, поле C_NO которой =1 (это код клиента Иванова), и установить значение CITY, равным "Киев".

13.2.6. Удаление записей

Теперь удалим всех клиентов, номера которых превышают 10:

```
DELETE FROM CLIENTS
WHERE C_NO > 10;
```

С помощью команды DELETE можно удалить все записи таблицы, указав условие, которое подойдет для всех записей, например:

```
DELETE FROM CLIETNS;
```

Если вторая часть оператора DELETE-WHERE не указана, значит, действие оператора распространяется на все записи сразу.

13.2.7. Выбор записей

Добавление, изменение и удаление записей — это, безусловно, очень важные команды, но чаще всего вы будете использовать оператор SELECT, который выбирает данные из таблицы. Синтаксис этого оператора, наверное, наиболее сложен:

```
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]
```

Сейчас мы не будем рассматривать оператор SELECT полностью: лучше сделаем это на примере — так намного нагляднее.

Например, для вывода всех записей из таблицы CLIENS, введите:

```
SELECT * FROM CLIENTS;
```

В результате вы получите такой ответ от сервера:

```
C_NO    FIO          ADDRESS        CITY          PHONE
1       Иванов И.П.  Ленина 6       Кировоград    80522111111
1       Иванов И.П.  Ленина 6       Кировоград    80522111111
2       Петров В.К.  П Пушкина 9    Кировоград    80522112111
```

Обратите внимание на первые две записи — они одинаковые. Теоретически, добавление одинаковых записей возможно — мы ведь не указали первичный ключ таблицы. Если вы хотите исключить одинаковые записи из ответа сервера (но не из таблицы!), введите запрос:

```
SELECT DISTINCT *
FROM CLIENTS;
```

Предположим, что вы хотите вывести только фамилию и номер телефона клиента, тогда введите такой запрос:

```
SELECT DISTINCT FIO, PHONE
FROM CLIENTS;
```

Теперь займемся усложнением наших запросов. Выведем все товары, цена которых превышает 500 рублей.

```
SELECT *
FROM TOVAR
WHERE PRICE > 500;
```

Вы можете использовать другие знаки отношений: <,>,=,<>,<=,>=.

Если ваша компания обслуживает несколько однофамильцев, и вы хотите вывести информацию обо всех Ивановых, используйте шаблон LIKE:

```
SELECT *
FROM CLIENTS
WHERE FIO LIKE '%Иванов%';
```

Запрос читается так: вывести всю информацию о клиентах, фамилия которых похожа на 'Иванов'. Если вы хотите выбрать данные из разных таблиц, перед именем поля нужно указывать имя таблицы. Следующий запрос выведет имена всех клиентов, которые хотя бы раз покупали у нас товар:

```
SELECT DISTINCT CLIENTS.FIO
FROM CLIENTS, ORDERS
WHERE CLIENTS.C_NO = ODREDS.C_NO;
```

Оператор SELECT позволяет использовать вложенные запросы. Однако MySQL их не поддерживает. Я это говорю для того, чтобы вы не пытались их использовать, тратя свое время впустую.

13.2.8. Дополнительные функции MIN, MAX, AVG, SUM

При работе с оператором SELECT вам доступно несколько полезных функций, вычисляющих количество элементов (COUNT), сумму элементов (SUM), максимальное и минимальное значение (MAX и MIN), а также среднее значение (AVG).

Следующие операторы выведут, соответственно, количество записей в таблице CLIENTS, самый дорогой товар и сумму всех товаров на складе.

```
SELECT COUNT(*)
FROM CLIENTS;

SELECT MAX(PRICE)
FROM TOVAR;

SELECT SUM(PRICE)
FROM TOVAR;
```

13.2.9. Группировка записей

Оператор SELECT позволяет группировать возвращаемые значения. Например, клиент Иванов (C_NO=1) несколько раз заказывал у нас какой-то товар. Значит его номер встречается в таблице ORDERS несколько раз. Другой клиент также мог сделать несколько заказов. Мы можем сгруппировать все записи по полю C_NO (номер клиента), а затем вывести сумму заказа каждого клиента.

```
SELECT CLIENTS.FIO, SUM(ORDERS.AMOUNT) AS TOTALSUM
FROM CLIENTS, ORDERS
WHERE CLIENTS.C_NO = ORDERS.C_NO
GROUP BY ORDERS.C_NO;
```

Группировку выполняет оператор GROUP BY, который является частью оператора SELECT. Оператор GROUP BY можно ограничить с помощью HAVING. Этот оператор используется для отбора строк, возвращаемых GROUP BY. HAVING можно считать аналогом WHERE, но только для GROUP BY:

```
HAVING <условие>
```

Например, нас интересуют только клиенты, которые заказали товаров на общую сумму, превышающую 1000.

```
SELECT CLIENTS.FIO, SUM(ORDERS.AMOUNT) AS TOTALSUM
FROM CLIENTS, ORDERS
WHERE CLIENTS.C_NO = ORDERS.C_NO
GROUP BY ORDERS.C_NO
HAVING TOTALSUM > 1000;
```

В этом запросе мы использовали псевдоним столбца TOTALSUM. В некоторых серверах SQL для определения псевдонима не нужно писать служебное слово AS, а некоторые требуют применение знака равенства:

```
SUM(ORDERS.AMOUNT) TOTALSUM или TOTALSUM = SUM(ORDERS.AMOUNT).
```

13.2.10. Сортировка записей

Пока мы не установили первичный ключ, сортировка нашей таблицы не выполняется. Данные будут отображены в порядке занесения их в таблицу. Для сортировки по полю C_NO результата вывода таблицы CLIENTS используется следующий оператор (сама таблица при этом не сортируется):

```
SELECT *
FROM CLIENTS
ORDER BY C_NO;
```

13.2.11. Ключи

Предположим, что кто-то добавил в таблицу CLIENTS запись:

```
1 Сидоров Егорова 11Кировоград 80522345111
```

В то же время, до этого номер 1 был закреплен за Ивановым. У нас получилось, что один и тот же номер сопоставлен разным клиентам. Теперь представим, что об этом ничего не знает оператор, находящийся в магазине. Сидоров заказывает монитор LG, но не оплачивает его вовремя (а, может, и вовсе передумал его покупать). Проходит некоторое время, и все забывают, кто именно заказал монитор: Иванов или Сидоров — ведь в таблице заказов стоит номер 1, который ассоциируется с обоими клиентами.

Чтобы избежать подобной путаницы, нужно использовать первичные ключи:

```
ALTER TABLE CUSTOMER
ADD PRIMARY KEY (C_NO);
```

После этого запроса поле C_NO может содержать только уникальные значения. В качестве первичного ключа нельзя использовать поле, допускающее значение NULL. Создать первичный ключ можно проще — при создании таблицы. Это делается так:

```
CREATE TABLE CLIENTS
(
C_NO    int     NOT NULL,
FIO     char(40) NOT NULL,
ADDRESS char(30) NOT NULL,
CITY    char(15) NOT NULL,
PHONE   char(11) NOT NULL,
PRIMARY KEY (C_NO);
);
```

Таблица ORDERS содержит сведения о заказах. По полю C_NO этой таблицы идентифицируется заказчик. Предположим, что в таблицу ORDERS кто-то ввел значение, которого нет в таблице CLIENTS. Кто же заказал товар? Нам нужно не допустить подобной ситуации, поэтому введите следующий запрос:

```
ALTER TABLE ORDERS
ADD FOREIGN KEY(C_NO) REFERENCES CLIENTS;
```

Введенные в таблицу ORDERS номера клиентов C_NO должны существовать в таблице CLIENTS. Аналогично нужно добавить внешний ключ по полю T_NO. Эта возможность называется декларативной целостностью.

Команда ALTER используется не только для добавления ключей. Она предназначена для реорганизации таблицы в целом. Вы хотите добавить еще одно поле? Или установить список допустимых значений для каждого из полей. Все это можно сделать с помощью команды ALTER:

```
ALTER TABLE CLIENTS
ADD ZIP char(6) NULL;
```

Этот оператор добавляет в таблицу CLIENTS новое поле ZIP типа char. Обратите внимание, что вы не можете добавить новое поле со значением NOT NULL в таблицу, в которой уже есть данные. Наша компания работает с клиентами только из Киева и Кировограда, поэтому целесообразно ввести список допустимых значений для таблицы CLIENTS:

```
ALTER TABLE CLIENTS
ADD CONSTRAINT INVALID_STATE SHECK (CITY IN ('Кировоград','Киев'));
```

13.2.12. Отключение от базы данных

Вам уже надоело работать с этой базой данных? Тогда с помощью запроса DISCONNECT отключитесь от нее, и, используя запрос CONNECT, подключитесь к другой базе данных. В некоторых серверах SQL запрос DISCONNECT не работает, а вместо CONNECT нужно использовать оператор USE.

При программировании на PHP вам этот запрос не нужен, поскольку для отключения используется функция **mysql_close()**.

13.2.13. Внешние ключи

Теперь, когда вы уже знакомы с основами SQL, немного в него углубимся. Мы уже знаем, как добавлять первичный ключ, теперь добавим внешний ключ при создании таблицы. Внешние ключи используются для связи одной таблицы с другой. Например, в таблице CLIENTS у нас есть

(C_NO=1) и Сидоров (C_NO=2). Оператор в мага-
ента — Иван заказа ошибся и указал несуществующий номер —
оформ к потом понять, кто заказал товар? Вот для решения
чнос и существуют внешние ключи.

- CASCADE — удаление или обновление зна
  встречается. Например, у нас есть таблиц
  хотим удалить клиента с номером С
  будут удалены сведения обо вс
- NOACTION — вы не
  клиентов до тех по
  клиентом. То
  основ

два клиента — Иванов (C_NO=1) и Сидоров (C_NO=2). Оператор в магазине при оформлении заказа ошибся и указал несуществующий номер — пусть C_NO = 3. Как потом понять, кто заказал товар? Вот для решения данной неоднозначности и существуют внешние ключи.

```
CREATE TABLE T
(
/* Описания полей таблицы */
FOREIGN KEY KEY_NAME (LIST)
REFERENCES ANOTHER_TABLE [(LIST2)]
[ON DELETE OPTION]
[ON UPDATE OPTION]
);
```

Здесь:

- KEY_NAME — это имя ключа. Имя не является обязательным, но я очень рекомендую всегда указывать имя ключа — если вы не укажете имя ключа, вы потом не сможете его удалить. А мало ли что может случиться, возможно, он вам больше будет не нужен?
- LIST — это список полей, входящих во внешний ключ. Список разделяется запятыми.
- ANOTHER_TABLE — это другая таблица, по которой устанавливается не внешний ключ, а необязательный элемент.
- LIST2 — это список полей этой таблицы. Типы полей в списке LIST должны совпадать с типами полей в списке LIST2.

Предположим, что в первой таблице у нас есть два поля — NO и NAME — целого и символьного типов, соответственно. Во второй таблице у нас есть поля с одинаковыми именами и типами. Определение внешнего ключа:

```
FOREIGN KEY KEY_NAME (NO, NAME)
REFERENCES ANOTHER_TABLE (NAME, NO)
```

Это определение некорректно, потому что типы полей NO и NAME не совпадают. Нужно использовать такое определение:

```
FOREIGN KEY KEY_NAME (NO, NAME)
REFERENCES ANOTHER_TABLE (NO, NAME)
[ON DELETE <OPTION>]
[ON UPDATE <OPTION>]
```

Если же поля имеют одинаковые имена, как в нашем случае, список LIST2 лучше вообще не указывать, чтобы не создавать себе лишних проблем.

Необязательные параметры ON DELETE <OPTION> и ON UPDATE <OPTION> определяют действие по обновлению информации в базе данных, при удалении информации из таблицы и при ее обновлении. А действия могут быть следующими:

- CASCADE — удаление или обновление значения везде, где оно встречается. Например, у нас есть таблица клиентов и заказов. Мы хотим удалить клиента с номером C_NO = 1. Из таблицы заказов будут удалены сведения обо всех заказах, сделанных клиентом.
- NOACTION — вы не сможете удалить информацию из таблицы клиентов до тех пор, пока вы не удалите все заказы, сделанные этим клиентом. То есть действие NOACTION запрещает удалять запись из основной таблицы, если она используется в дочерней таблице.
- SETNULL — все значения в дочерней таблице будут заменены на NULL (если значения NULL допускается).
- С помощью параметра SET_DEFAULT вы можете указать значение по умолчанию. Например, если вы укажете SET DEFAULT 1, то при удалении клиента с любым номеров его заказы будут приписываться клиенту с номером 1, который, разумеется, всегда есть в таблице CLIENTS.

Какое действие выбрать? Несомненно, действие NOACTION является самым безопасным, а CASCADE — самым удобным для оператора. Действие SETNULL я вообще не рекомендую использовать: во-первых, оно небезопасно, а во-вторых — неудобно: из таблицы клиентов (основная таблица) запись удалится, а потом вам вручную придется удалять записи в таблице заказов (дочерняя таблица). Действие SET_DEFAULT иногда очень даже полезно использовать, но я сказал иногда.

13.2.14. Удаление полей и таблиц

А как удалить поле? Стандартом SQL не предусмотрено удаление столбцов, но в MySQL мы все же можем это сделать:

```
ALTER TABLE CLIENTS
DROP ZIP;
```

Удалить таблицу еще проще:

```
DROP ORDERS;
```

13.3. Функции PHP для работы с MySQL

13.3.1. Соединение PHP-сценариев с таблицами MySQL

Основной функцией является **mysql_connect()**, которая подключает наш сценарий к серверу баз данных и выполняет авторизацию пользователя.

```
mysql_connect([string $hostname] [, string $user] [, string
$password])
```

Все параметры функции являются необязательными, поскольку значения по умолчанию можно прописать в конфигурационном файле `php.ini`. Если вы хотите указать другие имя MySQL-узла, имя пользователя и пароль, вы всегда можете это сделать. Параметр `$hostname` может быть указан в виде узел: порт.

Функция возвращает идентификатор (типа `int`) соединения, вся дальнейшая работа осуществляется только через этот идентификатор. При следующем вызове функции **mysql_connect()** с теми же параметрами новое соединение не будет открыто, а функция возвратит идентификатор существующего соединения.

Для закрытия соединения предназначена функция **mysql_close(int $connection_id)**. Соединение можно не закрывать — оно будет закрыто автоматически при завершении сценария. Если вы используете более одного соединения, при вызове **mysql_close()** нужно указать идентификатор соединения, которое вы хотите закрыть. Опять же таки, вы можете вовсе не закрывать соединения — они будут закрыты автоматически при завершении сценария, но помните, что это плохой стиль, поэтому лучше всего закрыть соединение самостоятельно.

Если вы планируете использовать только одно соединение с базой данных за все время работы сценария, можно не сохранять его идентификатор и не указывать идентификатор при вызове остальных функций.

Функция **mysql_connect()** устанавливает обыкновенное соединение с базой данных. Однако PHP позволяет создавать так называемые постоянные соединения — для этого используется функция **mysql_pconnect()**. Параметры у этой функции такие же, как и у **mysql_connect()**.

В чем же разница между обыкновенным и постоянным соединением? Постоянное соединение не закрывается после завершения работы сценария, даже если сценарий вызвал функцию **mysql_close()**. Соединение «привязывается» к PID потомка Apache (от имени которого оно и работает) и закрывается лишь тогда, когда удаляется процесс-владелец (например, перезагрузка или завершение работы сервера Apache).

PHP работает с постоянными соединениями примерно так: при вызове функции **mysql_pconnect()** PHP проверяет, было ли раньше установлено соединение. Если да, то возвращается его идентификатор, а если нет, открывается новое соединение и также возвращается идентификатор.

Недостатков у постоянных соединений практически нет. Постоянные соединения позволяют значительно снизить нагрузку на сервер, а также повысить скорость работы сценариев, использующих базу данных.

При работе с постоянными соединениями нужно следить, чтобы максимальное число клиентов Apache не превышало максимального числа клиентов MySQL, то есть параметр **MaxClient** (в файле httpd.conf) должен быть меньше или равен параметру **max_user_connection** (параметр MySQL).

13.3.5. Обработка результата запроса

Если запрос с помощью функции **mysql_query()** успешно выполнился, то в результате клиент получает набор записей, который может быть обработан следующими функциями:

mysql_result() — получить нужный элемент из набора записей;
mysql_fetch_array() — занести запись в массив;
mysql_fetch_row() — занести запись в массив;
mysql_fetch_assoc() — занести запись в ассоциативный массив;

13.3.2. Выбор базы данных

Функция **mysql_select_db(string $db [, int $id])** выбирает базу данных, с которой будет работать сценарий. Если открыто не более одного соединения, можно не указывать параметр `$id`.

Пример подключения к базе данных:

```
// Пытаемся установить соединение
if(!mysql_connect($SERVER,$USER,$PASSWD))
{
echo "Не могу подключиться к серверу";
exit;
}
// Выбираем базу данных
mysql_select_db($DB);
```

13.3.3. Обработка ошибок

Если произойдет ошибка, мы получим сообщение "Не могу подключиться к серверу" и все: сценарий завершит свою работу. Не очень-то удобно. Прежде всего, для себя, то есть для отладки своего сценария, можно использовать две функции:

- **mysql_errno(int $id)**
- **mysql_error(int $id)**

Первая функция возвращает номер ошибки, а вторая — сообщение об ошибке. Вместо тривиального сообщения удобнее использовать следующее:

```
echo "ERROR ".mysql_errno()." ".mysql_error()."\n";
```

Теперь вам не нужно гадать, из-за чего произошла ошибка — вы сразу увидите сообщение об ошибке.

13.3.4. Выполнение запросов к базе данных

Все запросы к текущей базе данных отправляются функцией **mysql_query()**. Этой функции нужно передать один параметр — текст запроса. Текст запроса может содержать пробельные символы и символы новой строки (`\n`). При этом текст должен быть составлен по правилам SQL.

Пример запроса:

```
$r=mysql_query("select * from hbd");
```

Следующий запрос должен вернуть содержимое таблицы `hbd`. Результат запроса присваивается переменной `$r`. Результат — это набор данных, который после выполнения запроса нужно обработать определенным образом.

Стоит отметить, что функция **mysql_result()** универсальная: зная количество записей и количество полей, можно «обойти» весь результат, но в тоже время эта функция достаточно медленная. Поэтому для обработки больших наборов записей рекомендуется использовать функции, имеющие в середине **_fetch_**, а именно, **mysql_fetch_row(), mysql_fetch_array()**, и т.д.

Функция **mysql_fetch_row(int $res)** получает сразу всю строку, соответствующую текущей записи результата `$res`. Каждый следующий вызов функции перемещает указатель запроса на следующую позицию (как при работе с файлами) и получает следующую запись. Если больше нет записей,

13.3.5. Обработка результата запроса

Если запрос с помощью функции **mysql_query()** успешно выполнился, то в результате клиент получает набор записей, который может быть обработан следующими функциями:

mysql_result() — получить нужный элемент из набора записей;
mysql_fetch_array() — занести запись в массив;
mysql_fetch_row() — занести запись в массив;
mysql_fetch_assoc() — занести запись в ассоциативный массив;
mysql_fetch_object() — занести запись в объект.

Перед тем как начинать разбор полученного результата, обычно бывает необходимо определить количество содержащихся в нем записей и полей. Функция **mysql_num_rows()** позволяет узнать, сколько записей содержит наш результат:

```
$r=mysql_query("select * from hbd");
echo "В таблице hbd ".mysql_num_rows($r)." записей";
```

Запись состоит из полей (грубо говоря — колонок). Узнать, сколько полей содержит каждая запись результата, позволяет функция **mysql_num_fields()**:

```
$r=mysql_query("select * from hbd");
echo "В таблице hbd ".mysql_num_fields($r)." полей";
```

Мы уже можем получить количество записей и количество полей таблицы. Нам осталось узнать значение каждого поля. Это можно сделать с помощью **mysql_result(int $result, int $row, mixed $field)**. Параметр `$row` задает номер записи, а `$field` — имя или порядковый номер поля.

Предположим, что наш запрос вернул следующий набор данных:

```
Email           FirstName   LastName
-----------------------------------------------
ivan@mail.ru    Ivan        Ivanov
sid@mail.ru     Petr        Sidorov
```

Вывести его в браузер позволяет следующий код:

```
$rows = mysql_num_rows($r);
$cols = mysql_num_fields($r);

echo "<PRE>";
for ($i=0; $i<$rows; $i++)
 {
  for (j=0; $j<$cols; $j++)
     echo mysql_result($r, $i, $j)."\t";
     echo "\n";
 }
echo "</PRE>";
```

Стоит отметить, что функция **mysql_result()** универсальная: зная количество записей и количество полей, можно «обойти» весь результат, но в тоже время эта функция достаточно медленная. Поэтому для обработки больших наборов записей рекомендуется использовать функции, имеющие в середине **_fetch_**, а именно, **mysql_fetch_row(), mysql_fetch_array()**, и т.д.

Функция **mysql_fetch_row(int $res)** получает сразу всю строку, соответствующую текущей записи результата `$res`. Каждый следующий вызов функции перемещает указатель запроса на следующую позицию (как при работе с файлами) и получает следующую запись. Если больше нет записей, то возвращается значение `false`.

Пример использования:

```
$r=mysql_query("select * from hbd where month=\"$bd_m\" and
day=\"$bd_d\");
for ($i=0; $i<mysql_num_rows($r); $i++)
{
$f=mysql_fetch_row($r);
echo $f;
}
```

Примечание.

Обратите внимание на строку запроса — текстовые значения полей обязательно должны быть в кавычках — это требование MySQL.

Использовать функцию **mysql_fetch_row()** не всегда очень удобно, поскольку значения всех полей одной записи находятся все в одной строке. Намного удобнее использовать функцию **mysql_fetch_array()**, возвращающую ассоциативный массив, ключами которого будут имена полей.

Примечание.

Функция **mysql_fetch_array(int $res [, int $result_type])** возвращает не ассоциативный массив, а массив, заданный необязательным параметром `$result_type`. Параметр `$resuly_type` может принимать значения:

- MYSQL_ASSOC — возвращается ассоциативный массив;
- MYSQL_NUM — возвращается массив с числовыми индексами (как в функции mysql_fetch_row());
- MYSQL_BOTH — возвращается массив с двойными индексами, то есть вы можете работать с ним, как с ассоциативным массивом и как со списком. MYSQL_BOTH — это значение по умолчанию параметра $result_type.

В PHP есть одна функция, которая возвращает «чистый» (с одним индексом) ассоциативный массив — **mysql_fetch_assoc(int $res)**. Эта функция — просто синоним для **mysql_fetch_array($res, MYSQL_ASSOC)**.

Пример использования функции **mysql_fetch_array()**:

```
$r=mysql_query("select * from hbd where month=\"$bd_m\" and
day=\"$bd_d\");
for ($i=0; $i<mysql_num_rows($r); $i++)
{
$f=mysql_fetch_array($r);
echo "$f[email] $f[name] $f[month] $f[day] $f[sex] <br>";
}
```

Как видите, использовать функцию **mysql_fetch_array()** намного удобнее, чем **mysql_fetch_row()**.

13.3.6. Получение информации о результате

Мы рассмотрели практически все необходимые вам для работы функции. Однако PHP предоставляет еще несколько полезных функций, которые позволяют узнать информацию о результате.

Функция **mysql_field_name(int $result, int $offset)** возвращает имя поля, находящегося в результате `$result` с номером `$offset` (нумерация начинается с 0). Другими словами, функция возвращает имя поля с номером `$offset`.

Функция **mysql_field_type(int $result, int $offset)** возвращает тип поля с номером `$offset` в результате `$result` (номер задается относительно результата, а не таблицы).

Функция **mysql_field_flags(int $result, int $offset)** возвращает перечисленные через пробел флаги (модификаторы), которые имеются у поля с номером `$offset`. Все поддерживаемые MySQL флаги представлены в табл. 13.7.

Флаги полей MySQL *Таблица 13.7*

Флаг	Описание
not_Null	Поле не может содержать неопределенного значения (NULL), то есть поле должно быть явно инициализировано
Primary_Key	Поле будет первичным ключом — идентификатором записи, по которому можно однозначно идентифицировать запись
auto_increment	При вставке новой записи значение этого поля будет автоматически увеличено на единицу, поэтому в таблице никогда не будет двух записей с одинаковым значением этого поля
Unique_Key	Поле должно содержать уникальное значение
Multiple_Key	Индекс
Blob	Поле может содержать бинарный блок данных
Unsigned	Поле содержит беззнаковые числа
Zerofill	Вместо пробелов используются символы с кодом \0
Binary	Поле содержит двоичные данные
enum	Поле может содержать один элемент из нескольких возможных (элемент перечисления)
timestamp	В поле автоматически заносится текущая дата и время при его модификации

Функция **mysql_field_flags()** возвращает флаги в виде строки, в которой флаги разделяются пробелами.

Сейчас рассмотрим полноценный пример вывода содержимого таблицы и оформления в виде таблицы HTML (см. листинг 13.1):

Листинг 13.1. Вывод содержимого таблицы БД в виде HTML-таблицы

```
<?
$SERVER = "localhost";
$USER = "user";
$PASSWD = "12345";
$DB = "my_db";

// Подключаемся
if(!mysql_connect($SERVER,$USER,$PASSWD))
{
echo $HEAD;
echo $CP;
echo $BODY;
echo "<h2>$ERROR</h2>";
echo "</body></html>";
exit;
}

// Выбираем базу данных
mysql_select_db($DB);

// Выводим заголовок таблицы
echo "<table border=1 width=100% bgcolor=gold>";
echo "<tr><td>E-mail</td><td>Имя</td><td>Месяц</td>";
echo "<td>Число</td><td>Пол</td></tr>";

// Запрос  select * from hbd
$r=mysql_query("select * from hbd");

// Выводим таблицу
for ($i=0; $i<mysql_num_rows($r); $i++)
{
echo "<tr>";

$f=mysql_fetch_array($r);
echo "<td>$f[email]</td><td>$f[name]</td><td>$f[month]</td>";
echo "<td>$f[day]</td><td>$f[sex]</td>";

echo "</tr>";
}
echo "</table></body></html>"
?>
```

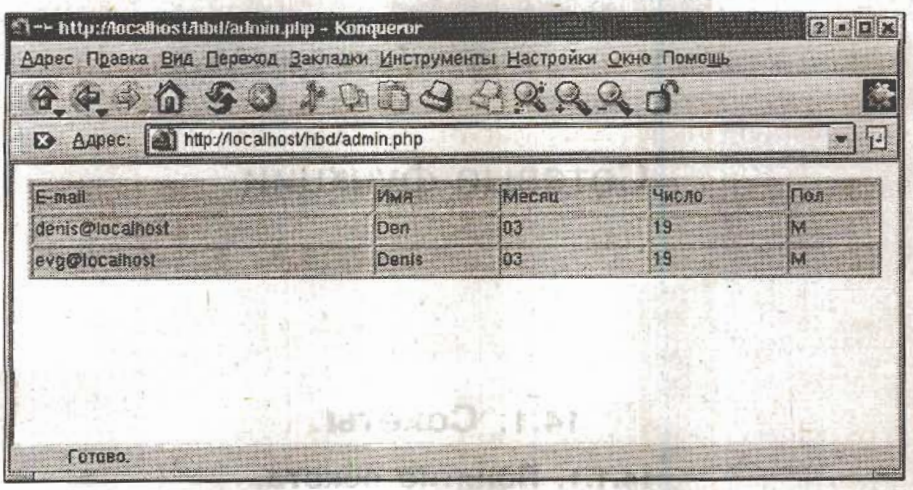


Рис. 13.1. *Вывод сценария, отображающего таблицу*

Еще одна полезная функция, которая может пригодиться вам при написании ваших сценариев — это функция **mysql_list_tables(string $db [, int connect_id])**. Функция возвращает идентификатор результата (одна колонка), содержащего имена всех таблиц в указанной базе данных. Второй параметр, как обычно, это идентификатор соединения:

```
$r = mysql_list_tables("hbd");
while($row = mysql_fetch_row($r))
{
echo $row."\n";
}
```

Сетевые функции

14.1. Сокеты

14.1.1. Понятие сокета. Функции для работы с сокетами

Открытие и использование сокетов

Сокет во многом похож на дескриптор файла (то есть с сетевым соединением можно работать, как с обычным файлом), но служит для нескольких иных целей, а именно, для передачи информации по сети. Помните, функция **fopen()** позволяла открывать удаленные файлы, которые находились на Web-серверах. Но для многих задач этого явно недостаточно. Получить файл по протоколу HTTP мы можем, но как реализовать отправку электронных сообщений на протоколе SMTP или любую другую сетевую функцию на каком-либо другом, отличном от HTTP протоколе? Конечно, в случае отправки письма можно вызвать программу **sendmail** (или **mail**) и передать ей письмо, чтобы она, в свою очередь, его отправила. Но это не очень удобно, потому что:

- Эта программа должна быть установлена на этом компьютере (соответственно, компьютер должен работать под управлением Linux).
- Мы должны знать точный путь к программе.

А что делать, если программа не установлена или мы используем Windows? Как отправить сообщение? PHP позволяет использовать более гибкий инструмент — сокеты, которые позволяют установить соединение по любому протоколу. Это значит, что мы можем «общаться» с любой сетевой программой (нам только нужно знать протокол обмена) и реализовать любую сетевую задачу.

Пример использования SMTP я выбрал неслучайно — в этой главе мы научимся отправлять письма с помощью сокетов, а потом — в следующей главе — будем использовать функцию **mail()** — она проще, но иногда она работает некорректно. Поэтому все равно рекомендуется использовать сокеты.

Для открытия сокета используется функция **fsockopen ()**:

```
fsockopen(string $host, int $port [, int $err] [, string $err_msg])
```

Первый параметр функции — это имя узла, к которому мы хотим подключиться, или его IP-адрес. Второй параметр — это порт службы. В нашем случае с отправкой письма — это 25 (SMTP). При возникновении ошибки в переменные `$err` и `$err_msg`, если они заданы (они являются необязательными), будут записаны, соответственно, код ошибки и текст сообщения об ошибке.

После открытия сокета мы можем работать с ним, как с обыкновенным файлом, то есть использовать функции **fwrite()**, **fread()**, **fgets()**, **fputs()**, и другие. В отличие от канала, сокет открывается, как для чтения, так и для записи.

Работу с протоколом SMTP через сокет мы рассмотрим в следующем пункте (п. 14.1.2), а сейчас приведем пример, демонстрирующий работу через сокет с протоколом http (см. листинг 14.1).

Листинг 14.1. Работа через сокет с протоколом HTTP

```
<?
// Открываем сокет
$s = fsockopen("localhost",80);
// Отправляем команду GET - получить index.html (/)
fputs($s, "GET / HTTP/1.0\n\n");
// Читаем ответ
while (!feof($s)) echo fgets($s,1000);
// Закрываем сокет
fclose($s);
?>
```

Блокирующий/неблокирующий режимы. Функция socket_set_blocking

Функция **socket_set_blocking(int $socket_descriptor, int $mode)** является псевдонимом для функции **stream_set_blocking(resource $stream, int $mode)**. Если параметр `$mode` принимает значение FALSE, поток переводится в неблокирующий режим, если TRUE — выполняется блокировка потока. Функция **stream_set_blocking()** влияет на работу функций **fread()** и **fgets()**. В блокирующем режиме функция **fgets()** будет ждать до тех пор, пока не прочитает данные из потока (а этот момент может и не настать). В неблокирующем режиме функция **fgets()**, не прочитав данные, завершает свою работу.

Примечание.

Функция **socket_set_blocking** (как и **stream_set_blocking()**) появилась в версии PHP 4.3.0. Если у вас версия PHP младше, вы не сможете использовать режим блокировки потока.

14.1.2. Отправка электронных писем с помощью сокетов

Чтобы отправить письмо, мы должны знать, как сделать это вручную. Нет, не нажать кнопку «Новое сообщение», а затем — «Отправить». Мы должны знать сам протокол SMTP, то есть, что почтовая программа отправляет серверу и какие данные от него получает. Если вы не хотите вникать в некоторые детали SMTP, то пропустите этот раздел. Об отправке писем с помощью соответствующих функций PHP мы будем говорить в следующей главе.

Итак, если вы все-таки решились — запустите программу **telnet**:

```
telnet SMTP-узел 25
```

Если у вас Linux и программа **sendmail** (или другой агент MTA) установлена, можно использовать локальный хост — так будет даже лучше. Введите:

```
telnet localhost 25
```

Вы должны увидеть приветствие программы MTA (Mail Transfer Agent — Агент передачи почты).

```
220 localhost.localdomain ESMTP Sendmail 8.11.6/8.11.6; Wed, 16
Apr 2003 13:24:56 +0300
```

Программа поприветствовала нас и ждет нашей команды. Поздороваемся с программой:

```
HELO  localhost
```

Обратите внимание, что слово «привет» пишется `HELO`, а не `HELLO`. Вместо `localhost` нужно указать ваш узел, если у вас установлен агент MTA.

На наше приветствие программа ответит:

```
250 localhost.localdomain Hello localhost.localdomain
[127.0.0.1], pleased to meet you
```

Данное сообщение означает, что нам разрешена отправка почты.

Примечание.

Что делать, если вы работаете в Windows и у вас не установлен MTA? Тогда запустите программу **telnet** (она есть в любой операционной системе) и подключитесь к серверу провайдера: `telnet mail.isp.ru 25`. В команде `helo` нужно ввести имя своей машины, узнать которое поможет программа **ipconfig**.

После взаимного приветствия нужно представиться:

```
MAIL FROM:  den@localhost
```

В команде MAIL FROM нужно указать свой адрес электронной почты. Ответ программы выглядит так:

```
250 2.1.0 den@localhost... Sender ok
```

Программа сообщает, что отправитель допустим. Потом нужно указать адрес реципиента:

```
RCPT  TO:  evg@localhost
```

Ответ программы:

```
250 2.1.5 evg... Recipient ok
```

Вот теперь можно послать команду DATA, что означает, что мы будем передавать письмо.

```
DATA
```

Программа сообщит нам, что для окончания ввода нужно нажать «Enter. Enter»:

```
354 Enter mail, end with "." on a line by itself
```

Введите сообщение, например, `Test Message`. Потом нажмите «Enter. Enter» и увидите сообщение о том, что наше письмо поставлено в очередь на отправку:

```
250 2.0.0 h3GAOup01391 Message accepted for delivery
```

Сейчас можно заново повторить команды MAIL FROM, RCPT TO, DATA для создания нового сообщения, но мы этого делать не будем, поэтому введите команду QUIT.

Приведенный пример демонстрирует обмен почтовой программой и сервером по протоколу SMTP. Конечно, пример очень тривиален, потому что мы не использовали ни заголовок, ни аутентификации, но сейчас речь не о них. Главное, чтобы вы уловили идею работы с сокетом: нужно отправить команду, прочитать результат, отправить следующую команду, опять прочитать результат, и т.д.

Сценарий, приведенный в листинге 14.2, автоматизирует проделанную выше работу.

Листинг 14.2. Работа через сокет с протоколом SMTP

```
<?
$CRLF = "\r\n";
$r=$loops = 0;
$m =$line = "";

// Открываем сокет

$s = fsockopen("localhost","25", $r, $m);

// Функция чтения ответа
function get_data()
```

- **gethostbynamel(string $name)** — с одним именем может ассоциироваться несколько IP-адресов: такая организация позволяет снизить нагрузку на сервер, то есть «разбросать» запросы клиентов по разным серверам. Данная функция возвращает список IP-адресов, которые ассоциируются с именем.

Первая функция возвращает доменное имя узла по IP-адресу:

```
echo gethostbyaddr("192.168.1.1");
echo gethostbyaddr($REMOTE_ADDR);
```

Функция **gethostbyname()** является обратной к первой: она возвращает IP-адрес по имени:

```
echo gethostbyname("www.mail.ru");
```

Одному доменному имени могут соответствовать несколько IP-адресов. Для чего это нужно? Если нагрузка на сервер большая, то правильно настроенный сервер DNS будет каждый раз возвращать другой IP-адрес узла — IP-адрес зеркала. Выходит, что один клиент работает с сайтом `www.big_site.net`, подключившись по адресу 192.168.1.1, а второй работает с зеркалом этого сайта (но для клиента это совершенно прозрачно), но подключившись по адресу 192.168.1.2. Нагрузка на серверы распределяется равномерно, вследствие чего повышается производительность работы сайта.

Третья функция **gethostbynamel()** возвращает список (массив) IP-адресов, которые ассоциируются с определенным доменным именем. Вот пример использования данной функции:

```
$ips = gethostbynamel("www.big_site.net");
foreach ($ips as $ip) echo "$ip <br>";
```

В качестве вывода мы получим длинный список IP-адресов. Не знаете, какое имя передать функции **gethostbynamel()**? В целях эксперимента укажите www.yahoo.com. Вывод я не привожу умышленно, чтобы вам было интереснее.

14.2.2. Корректное разрешение IP-адреса в доменное имя

Попробуем разрешить IP-адрес 111.111.111.333 (я знаю, что такой IP не существует, но я специально использую несуществующий IP, чтобы не произошло недоразумений с владельцем этого IP) в доменное имя.

```
<?
echo gethostbyaddr("111.111.111.333");
?>
```

Функция **gethostbyaddr()** работает следующим образом: она обращается к узлу 111.111.111.333 и запрашивает *у него* доменное имя. Плохой дядька-владелец компьютера с адресом 111.111.111.333 настроил свой компьютер так, чтобы он сообщал функции **gethostbyaddr()**, что имя его компьютера `www.yahoo.com`. В результате вы получите вывод `www.yahoo.com`, что не является истиной. Вот вам и подмена доменного имени!

Для корректного преобразования IP-адреса в имя функции **gethostbyaddr()** и **gethostbyname()** должны работать в паре:

```
$ip = "192.168.1.1";
$domain_name = gethostbyaddr($ip);

// произошла ошибка и в результате мы получили, что
$domain_name = $ip
if ($domain_name == $ip) die("Неверный IP!");

$ip2 = gethostbyaddr($domain_name);

// Если была ошибка, $ip2 = $domain_name
if ($ip2 === $domain_name) die("Неверное доменное имя");

if ($ip === $ip2) echo "$ip => $domain_name";
else "по адресу $ip находится хакер!!!";
```

Создание своих проектов на PHP

Глава 15. Отправление и прием писем. Делаем почтовую рассылку на сайте

Глава 16. Делаем чат для своего сайта

Глава 17. Голосование на сайте

Глава 18. Лента новостей

Глава 19. Гостевая книга и Форум

Глава 20. Счетчик и анализ серверной статистики

Глава 21. Сессии в PHP

Глава 22. Работа с WWW

Глава 23. Создание и использование шаблонов. Модули

Глава 24. Система PHP-Nuke — система управления содержимым сайта (контентом). Система PHP-Nuke

Отправление и прием писем. Делаем почтовую рассылку на сайте

15.1. Использование функции mail() для отправки сообщений

В предыдущей главе мы рассмотрели, как отправлять электронные сообщения через сокеты. На примере мы убедились, что отправлять сообщения таким способом не очень удобно, потому что нужно знать особенности протокола SMTP.

В этой главе мы рассмотрим функцию **mail()**, позволяющую быстро отправить сообщение. Формат этой функции таков:

```
mail(string $email, string $subject, $string $message [,string
$haders]);
```

Предположим, что нам нужно отправить письмо по адресу ivan@ivanov.org, тема сообщения «Привет» и само сообщение — «Как дела?» Воспользуемся функцией **mail()**:

```
mail("ivan@ivanov.org","Привет","Как дела?");
```

Если ваша система правильно настроена, сообщение будет отправлено.

Примечание.

Для правильной настройкой системы нужно отредактировать секцию mail function файла конфигурации php.ini. В нем нужно указать имя SMTP-сервера, если сервер работает под Windows, или путь к программе sendmail, если на сервере установлена ОС Linux.

В функции **mail()**, как вы уже догадались, первым параметром задается адрес электронной почты получателя, а вторым параметром — само сообщение. Теперь разберемся с необязательным параметром **Header**. Он содержит различные служебные заголовки письма. *Все заголовки знать* вам не нужно, вот самые необходимые:

- **From** — содержит имя и адрес отправителя;

- **Reply-To** — задает адрес для ответа;
- **X-Mailer** — позволяет указать название почтовой программы;
- **Content-type** — определяет тип содержимого;
- **Content-Transfer-Encoding** — сколько бит используется для передачи символа. Если вы планируете отправлять сообщения на русском языке, установите значение 8 bit.

Расширенный пример использования функции:

```
mail("ivan@ivanov.org","Привет","Как дела", "From: sidorov@ukr.net\n",
"Reply-to: root@ukr.net\n",
"Content-Type: text/plain; charset=\"koi8-r\"\n",
"Content-Transfer-Encoding: 8bit");
```

Обратите внимание, что мы с помощью заголовков **Content-type** и **Content-Transfer-Encoding** установили кодировку KOI8-R, а также 8-битное кодирование символов при передаче сообщения.

Если не указать кодировку, то пользователь вместо русских букв увидит непонятные иероглифы, а если указать кодировку, но забыть указать параметр **Content-Transfer-Encoding**, то вместо русского текста он увидит так называемый транслитерационный **tekst**. Заголовок **Content-type** также указывает формат сообщения — `text/plain`. О форматах мы поговорим чуть позже.

Если вы хотите преобразовать заголовки сообщения и само сообщение из одной кодировки в другую, например, из koi8-r в windows-1251, используйте функцию **convert_cyr_string()**, которая уже рассматривалась в этой книге.

Предположим, что нам нужно сделать небольшую рассылку. Список адресов находится в файле `users.txt`, а само сообщение — в файле `message.txt`. Сценарий из листинга 15.1 автоматизирует рутинную работу.

Листинг 15.1. Простой список рассылки

```
<?
// Список-массив электронных адресов
$Users = file("users.txt");
// Сообщение
$Message = join(' ',file("message.txt"));

// Отправляем сообщение каждому пользователю
foreach ($Users as $user) mail($user,"Spam",$Message);

?>
```

15.2. Если сообщения не отправляются

Иногда функция **mail()** не может отправить сообщение. В большинстве случаев это происходит потому, что она не смогла правильно определить SMTP-сервер. Если у вас установлена и настроена программа **Sendmail** (или другой MTA), у вас не будет проблем с отправлением писем. А вот если нет... Функция просто не сможет определить адрес SMTP-сервера. Поэтому вам ничего другого не остается, как использовать сокеты (или класс **HtmlMimeMail**, который мы рассмотрим в следующем пункте).

Напомню, что пример отправки сообщения с использованием сокетов был рассмотрен в предыдущей главе.

15.3. Отправление писем в формате HTML и вложений

15.3.1. Общие положения

MIME-формат **plain/text** идеально подходит для электронных сообщений, но иногда нам нужно послать сообщение с форматированием текста и/или картинкой. Поэтому нам просто не обойтись без формата **text/html**. Обычно в формате HTML отправляются рекламные сообщения.

Если вы хотите отправить простое HTML-сообщение без картинок, то можно воспользоваться и функцией **mail()**. При этом можно даже отправить сообщение с картинками, но сами картинки отправлены не будут — в HTML-документе нужно будет сделать ссылку `<img src=http://...>`. Если почтовая программа пользователя поддерживает HTTP (если она поддерживает HTML, то HTTP она уж точно поддерживает), она отобразит картинки.

Вот пример отправления сообщения в формате HTML:

```
$msg = "<html><body><h1>Привет!!!</h1></body></html>";
mail("ivan@ivanov.org","Привет",$msg,"Content-Type: text/html;
charset=\"koi8-r\"\n");
```

Теперь предположим, что нам нужно передать вложение или картинку, чтобы она отображалась прямо в сообщении — тогда пользователю не нужно будет ждать, пока загрузится картинка — он сразу ее скачает вместе с сообщением. В этом случае формат HTML нам не поможет. Нужно отправить сообщение в так называемом MIME-формате. Что представляет собой этот формат, говорить не стану — если вам интересно, создайте письмо с вложением в почтовой программе (например, в Outlook), сохраните его как файл, а потом просмотрите в текстовом редакторе.

15.3.2. Использование класса HtmlMimeMail

Класс **HtmlMimeMail**, разработанный Ричардом Гейесом (Richard Heyes), позволяет отправлять сообщение в MIME-формате. Скачать данный класс можно по адресу `http://www.phpguru.org/downloads/html.mime.mail/`.

Примечание.

Класс можно рассматривать как тип переменной. Пока вам о нем больше ничего не нужно знать. Подробнее классам и вообще объектно-ориентированному программированию посвящена отдельная глава. Забегая наперед, скажу, что класс – это набор данных и методов для их обработки – такой себе черный ящик, предназначенный для решения какой-нибудь задачи. Вы не знаете алгоритм решения этой задачи, вы просто вызываете методы класса – вот по такому принципу построена работа с классами. Конечно, можно открыть исходный код класса, но зачем? Ведь класс сам знает, что и как делать. Правда, иногда нужно немного адаптировать готовый класс для решения именно нашей задачи. Вот этим мы и займемся чуть ниже в этой главе.

Распакуйте архив в каталог `/var/www/html/mime`. После распаковки класс пока еще не готов к работе. Чтобы каждый раз не указывать кодировку и параметры SMTP-сервера, пропишем их один раз. Откройте в текстовом редакторе файл `HtmlMimeMail.php` и найдите следующие строки:

```
$this->build_params['html_encoding'] = 'quoted-printable';
$this->build_params['text_encoding'] = '7bit';
$this->build_params['html_charset']  = 'ISO-8859-1';
$this->build_params['text_charset']  = 'ISO-8859-1';
$this->build_params['head_charset']  = 'ISO-8859-1';
$this->build_params['text_wrap']     = 998;
```

Кодировка ISO-8859-1 не поддерживает кириллицу. Для отправки сообщений на русском языке нужно использовать кодировку KOI8-R (или CP1251 — как кому больше нравится). Также для корректного отображения кириллицы нужно включить 8-битное кодирование символов. Разработчик класса **HtmlMimeMail** не предвидел, что класс будут использовать на территории бывшего СНГ, поэтому он указал кодировку ISO-8859-1 и 7-битное кодирование. Сами понимаете, нас эти параметры никак не устраивают. Поэтому без всякого зазрения совести заменяем их следующими:

```
$this->build_params['html_encoding'] = 'quoted-printable';
$this->build_params['text_encoding'] = '8bit';
$this->build_params['html_charset']  = 'koi8-r';
$this->build_params['text_charset']  = 'koi8-r';
$this->build_params['head_charset']  = 'koi8-r';
$this->build_params['text_wrap']     = 998;
```

Мы включили кодировку **koi8-r** и 8-битовое кодирование символов. Можно было бы и не изменять исходные тексты класса, но не хочется устанавливать эти параметры в каждой программе, использующей класс. Исходя из тех же самых соображений, укажем имя SMTP-сервера по умолчанию и порт:

```
$this->smtp_params['host'] = 'localhost';
$this->smtp_params['port'] = 25;
```

Если агент MTA не установлен на вашей машине, то имя localhost нужно заменить именем SMTP-сервера вашего провайдера.

Затем откройте файл `smtp.php` и найдите следующие строки:

```
$this->authenticated   = FALSE;
$this->timeout     = 5;
$this->status      = SMTP_STATUS_NOT_CONNECTED;
$this->host      = 'localhost';
$this->port      = 25;
$this->helo      = 'localhost';
$this->auth      = FALSE;
$this->user      = '';
$this->pass      = '';
```

Если ваш SMTP-сервер требует аутентификации, то установите значение TRUE для параметра `authenticated`. Также нужно указать имя пользователя (user) и пароль (pass). Также нужно указать имя SMTP-сервера и его порт. Если у вас медленный канал связи, будет полезным увеличение тайм-аута, установите, например, 10 или даже 30 секунд. Тайм-аут устанавливается для функции **fsockopen()**. Можно пробежаться по остальным файлам пакета и изменить значение параметра по умолчанию **charset** некоторых функций на **koi8-r**.

Вот теперь ваш класс полностью готов к работе. Следующий пример (листинг 15.2) демонстрирует, как отправить письмо в формате HTML с картинками, а так же, как прикрепить к письму файл (в нашем случае — zip-архив).

Листинг 15.2. Пример использование класса HtmlMimeMail

```
<?
//  отображаем все ошибки - только для отладки
  error_reporting(E_ALL);

// подключаем класс htmlMimeMail. Файл htmlMimeMail.php
// должен находиться в одном каталоге с нашим сценарием
  include('htmlMimeMail.php');

// создаем новый объект класса htmlMimeMail
```

```
  $mail = new htmlMimeMail();
// Читаем картинку. Эта картинка нужна для нашего HTML-файла.
// В данном случае она является фоном.
  $background = $mail->getFile('bg.gif');

// Читаем вложение - файл docs.zip
  $attachment = $mail->getFile('docs.zip');

// Читаем текст сообщение: сначала в текстовом формате -
// для почтовых программы, которые
// не поддерживают HTML, а затем - в формате HTML.
// Желательно, чтобы имя файла было message.html
  $text = $mail->getFile('message.txt');
  $html = $mail->getFile('message.html');

// Говорим, что наше письмо будет в HTML и указываем
// просто текст сообщения
  $mail->setHtml($html, $text);
// Связываем картинку с HTML
  $mail->addHtmlImage($background, 'background.gif', 'image/
gif');

// Добавляем вложение. MIME-тип: application/zip
  $mail->addAttachment($attachment, 'docs.zip', 'application/
zip');

// Адрес отравителя
  $mail->setReturnPath('ivan@ivanov.com');

// А это - некоторые заголовки
  $mail->setFrom('"Иван" <ivan@ivanov.com');
  $mail->setSubject('Привет');
  $mail->setHeader('X-Mailer', 'HTML Mime mail class (http://
www.phpguru.org)');

// Отправляем письмо по адресу: sidorov@ukr.ne
  $result = $mail->send(array('sidorov@ukr.ne'), 'smtp');

// Если есть ошибки, выводим их, а если нет -
// отображаем "Отправлено"
  if (!$result) {
    print_r($mail->errors);
    } else {
      echo 'Отправлено';
      }
?>
```

15.3.3. Сводный алгоритм отправки сообщений в различных MIME-форматах

Я старался писать понятные комментарии, но, на всякий случай, прокомментирую еще раз. Чтобы отправить сообщение в формате MIME, нужно:

1. Подключить класс и создать объект класса **HtmlMimeMail**.
2. В обыкновенные переменные прочитать все файлы (и вложения, и картинки, которые использует HTML, а также сам текст письма) с помощью метода `GetFile()`.
3. Используя метод `SetHtml`, установить текст письма. Вы можете прикрепить к письму несколько HTML-файлом, но только один будет основным, все остальные будут обыкновенными вложениями.
4. Связать картинки с HTML-файлом сообщения (метод **addHtmlImage**).
5. Добавить все вложения методом **addAttachment()**, указав при этом переменную вложения, название вложения (обычно оно соответствует названию файла), а также MIME-тип.
6. Установить обратный адрес, некоторые заголовки письма.
7. Отправить письмо методом **Send()**.

Теперь рассмотрим еще один пример (листинг 15.3), как отправить с помощью класса обыкновенное текстовое (не HTML) сообщение и вложение, то есть обыкновенное письмо.

Листинг 15.3. Отправление сообщения с вложением

```
<?
   error_reporting(E_ALL);
// подключаем класс htmlMimeMail
   include('htmlMimeMail.php');

// создаем новый объект класса htmlMimeMail
   $mail = new htmlMimeMail();

   // Читаем вложение
   $attachment = $mail->getFile('docs.zip');

   // Читаем текст сообщения
   $text = $mail->getFile('msg.txt');

   // Используем метод SetText для создания
   // текстового сообщения
   $mail->setText($text);

   // Добавляем вложение
   $mail->addAttachment($attachment, 'docs.zip', 'application/
zip');
```

```
  // Отправляем сообщение
  $mail->setFrom('Иван <ivan@localhost>');
  $result = $mail->send(array('"Сидоров" <sidorov@localhost>'));

  echo $result ? 'Отправлено' : 'Ошибка';
?>
```

Как видите, ничего сложного нет, просто вместо метода **setHtml()** используется метод **SetText()**.

15.3.4. Автоматическая отправка копий сообщений

Возможно, вы захотите отправить кому-то копию сообщения. Для этого используются методы **SetBcc()** и **SetCC()**:

```
  // "слепая" копия
$mail->setBcc('bcc@example.com');
// простая копия
  $mail->setCc('Carbon Copy <cc@example.com>');
```

Надеюсь, что приведенного материала вполне достаточно для работы с классом. Если вас интересует дополнительная информация — изучите исходные тексты класса — они достаточно информативны.

15.4. Прием писем по протоколу POP3

15.4.1. Общие сведения о механизме приема писем и протоколе POP3

В предыдущем пункте мы рассмотрели класс **HtmlMimeMail**, позволяющий легко отправлять письма в формате MIME. Перед этим мы узнали, как можно отправить письмо: используя функцию **Mail()** и напрямую — через сокеты. Теперь осталось разобраться, каким образом в PHP можно организовать получение писем.

Встроенной функции, насколько я знаю, нет, но зато можно написать сценарий, который, напрямую — через сокеты, «свяжется» с сервером POP3 и получит сообщения. Конечно, для этого нужно знать протокол POP3, но его рассмотрение выходит за рамки этой книги. Поэтому мы пойдем по пути наименьшего сопротивления: будем использовать уже готовый класс **POP3 (разработчик Manuel Lemos [mlemos@acm.org])**. С одной стороны, это правильно: зачем заново изобретать велосипед? А с другой — ведь никто не мешает вам разобраться с протоколом POP3 и самому написать аналог класса POP3, ведь так?

Примечание.

Протокол POP (Post Office Protocol) ver. 3 используется для получения электронных писем с их загрузкой на компьютер клиента. Служба POP3 использует порт 110.

Минимальные знания о протоколе POP3 нам все-таки просто необходимы. Программа-клиент, например, The Bat!, Outllok, kmail или любая другая, поддерживающая POP3, подключается к почтовому серверу, который также должен поддерживать протокол POP3. После подключения программа-клиент отправляет серверу имя пользователя и его пароль. Если эти данные правильны, почтовый клиент получает список сообщений. Список возвращается в виде:

```
Номер сообщения / Размер сообщения
```

Каждое сообщение, кроме своего номера, имеет уникальный идентификатор, например, 3ebcc2a500000001.

Почтовый клиент, получив список сообщений, может:

- Начать получение каждого сообщения.
- Получить какое-нибудь определенное сообщение.
- Удалить все сообщения или сообщения, удовлетворяющие условию, например, размер которых превышает 1 Мб.

Вообще, действия почтового клиента зависят от того, как он запрограммирован изначально, а также от того, как он был впоследствии настроен пользователем.

Получив сообщение, почтовый клиент может оставить его на сервере или же пометить для удаления, другими словами, удалить полученное сообщение.

15.4.2. Использование класса POP3

Вот теперь мы можем приступить к написанию собственного сценария, эмулирующего работу почтового клиента.

Класс POP3 доступен по адресу `http://dkws.narod.ru/php`.

Рассмотрим важнейшие методы класса (табл. 15.1).

Методы класса POP3 — *Таблица 15.1*

Метод	Описание
Open()	Открывает новое соединение. До вызова этого метода нужно указать имя сервера (свойство hostname): в противном случае нечего будет открывать – ведь вы не указали самое главное — имя сервера, к которому нужно подключиться

Методы класса POP3 *Таблица 15.1 (продолжение)*

Метод	Описание
Login(string $user, string $pass, int $apop)	Регистрирует пользователя $user с использованием пароля $pass. Этот метод нужно вызвать после установления соединения, то есть после успешного вызова **Open()**. Параметр $apop определяет должна (1) ли использоваться авторизация APOP или нет (0)
Statistics (&$messages,&$size)	Метод **Statistics()** записывает в переменную $messages количество сообщений, а в переменную $size – общий объем всех сообщений
ListMessages (string $message, int $id)	Возвращает список сообщений. **Следует отметить, что нумерация сообщений начинается с единицы — это важно.** Первый параметр задает количество возвращаемых сообщений. Если задана пустая строка, будут возвращены все сообщения. Если второй параметр ($id) = 1, то будет возвращен уникальный идентификатор каждого сообщения, иначе будет возвращен размер сообщения
RetrieveMessage ($N,&$headers, &$body,int lines)	Получает сообщение с номером $N. В переменные $headers и $body записываются заголовки и тело сообщения, соответственно. Последний параметр определяет, сколько строк сообщения будет возвращено. Если хотите получить все сообщение, укажите отрицательное значение
DeleteMessage(int $N)	Удаляет сообщение с номером $N
ResetDeletedMessages()	Удаляет список сообщений, отмеченных для удаления. Если вы отметите какие-нибудь сообщения методом DeleteMessage(), а потом вызовете ResetDeletedMessages(), то ни одно сообщение не будет удалено. Этот метод полезно использовать при отладке сценария, чтобы каждый раз не отправлять тестовые сообщения

15.4.3. Пишем сценарий получения сообщений

Вот теперь наших знаний вполне достаточно, чтобы написать сценарий получения сообщений (листинг 15.4).

Листинг 15.4. Получение всех сообщений

```
<?
// Подключаем класс pop3. Файл pop3.php должен находиться
// в одном каталоге
// с нашим сценарием
require('pop3.php');
```

```
// Задаем имя пользователя и пароль
$user="denis";
$password="123456";
// Не использовать APOP
$apop=0;

// Создаем объект класса POP3
$pop3_connection=new pop3_class;

// Почтовый сервер - localhost. Если вы не настроили
// POP3 сервер на своей
// машине, можете попробовать сервер провайдера или
// какой-нибудь другой,
// например, pop.mail.ru

$pop3_connection->hostname="localhost";

// Открываем соединение
if(($error=$pop3_connection->Open())==="")
{
// Мы только что подключились к почтовому серверу
  echo "<PRE>Connected to the POP3 server &quot;
        $pop3_connection->hostname&quot;.</PRE>\n";

// Регистрируемся - передаем имя и пароль
  if(($error=$pop3_connection->Login($user,$password,$apop))==="")
  {

   // Имя и пароль приняты.
   echo "<PRE>User &quot;$user&quot; logged in.</PRE>\n";

   // Получаем статистику
   if(($error=$pop3_connection->Statistics(&$messages,&$size))==="")
   {
 echo "<PRE>There are $messages messages in the mail box with
a total of $size bytes.</PRE>\n";

        // Получаем список сообщений
       $result=$pop3_connection->ListMessages("",0);

       // Если предыдущий вызов возвратил
       // переменную-массив, значит
      // в почтовом ящике есть сообщения
```

```
     if(GetType($result)=="array")
     {

// Выводим номера сообщений и размер
for(Reset($result),$message=0;$message<count($result);
Next($result),$message++)

     echo "<PRE>Message ",Key($result)," -
",$result[Key($result)],"
 bytes</PRE>\n";

   // Получаем идентификаторы сообщений
     $result=$pop3_connection->ListMessages("",0);

     // Если предыдущий вызов возвратил переменную-массив,
     // значит
     // в почтовом ящике есть сообщения
     if(GetType($result)=="array")
     {

// Выводим номера сообщений и размер
for(Reset($result),$message=0;$message<count($result);
Next($result),$message++)

     echo "<PRE>Message ",Key($result)," -
",$result[Key($result)],"
 bytes.</PRE>\n";

   // Получаем идентификаторы сообщений
      $result=$pop3_connection->ListMessages("",1);

      if(GetType($result)=="array")
      {
for(Reset($result),$message=0;$message<count($result);
Next($result),$message++)
       echo "<PRE>Message ",Key($result),", Unique ID -
 \"",$result[Key($result)],"\"</PRE>\n";

// Количество сообщений > 0
       if($messages>0)
       {
// Получаем каждое сообщение. Нумерация - с единицы

for($M=1; $M<=$messages; $M++)
{
```

```
 // Получаем сообщение с номером $M
 if(($error=$pop3_connection->RetrieveMessage(
$M,&$headers,&$body,2)) ==="")

 // Выводим заголовки сообщения
 {
 echo "<PRE>Message $M :\n-Message headers starts
   below-</PRE>\n";
 for($line=0;$line<count($headers);$line++)
       echo "<PRE>",HtmlSpecialChars($headers[$line]),"</PRE>\n";
echo "<PRE>-Message headers ends above-\n-Message body starts
below- </PRE>\n";

 // Выводим тело сообщения.
// Мы получили только 2 строки (см. RetrieveMessage)
 for($line=0;$line<count($body);$line++)
  echo "<PRE>",HtmlSpecialChars($body[$line]),"</PRE>\n";
 echo "<PRE>-Message body ends above-</PRE>\n";
 // Помечаем сообщения на удаление
 if(($error=$pop3_connection->DeleteMessage($M))==="")
 {
  echo "<PRE>Marked message $M for deletion.</PRE>\n";

  // Сбрасываем список сообщений для удаления
  if(($error=$pop3_connection->ResetDeletedMessages())==="")
    {
echo "<PRE>Resetted the list of messages to be deleted.</PRE>\n";
    }
  }
       } } // заканчивается цикл for
// заканчивается блок if ($messages>0)

// Отключаемся
if($error=="" && ($error=$pop3_connection->Close())==="")
  echo "<PRE>Disconnected from the POP3 server
&quot;$pop3_connection->hostname&quot;.</PRE>\n";

      }
      else
        $error=$result;
     }
     else
      $error=$result;
    }
  }
}
if($error!="")
  echo "<H2>Error: ",HtmlSpecialChars($error),"</H2>";
?>
```

На рис. 15.1 и 15.2 вы видите вывод сценария, если сообщений нет, и если есть одно сообщение.

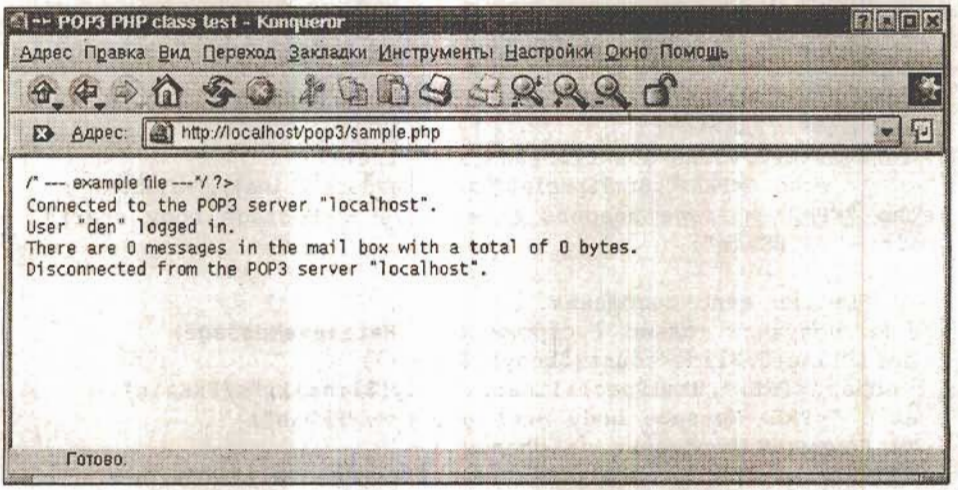


Рис. 15.1. *Нет сообщений*

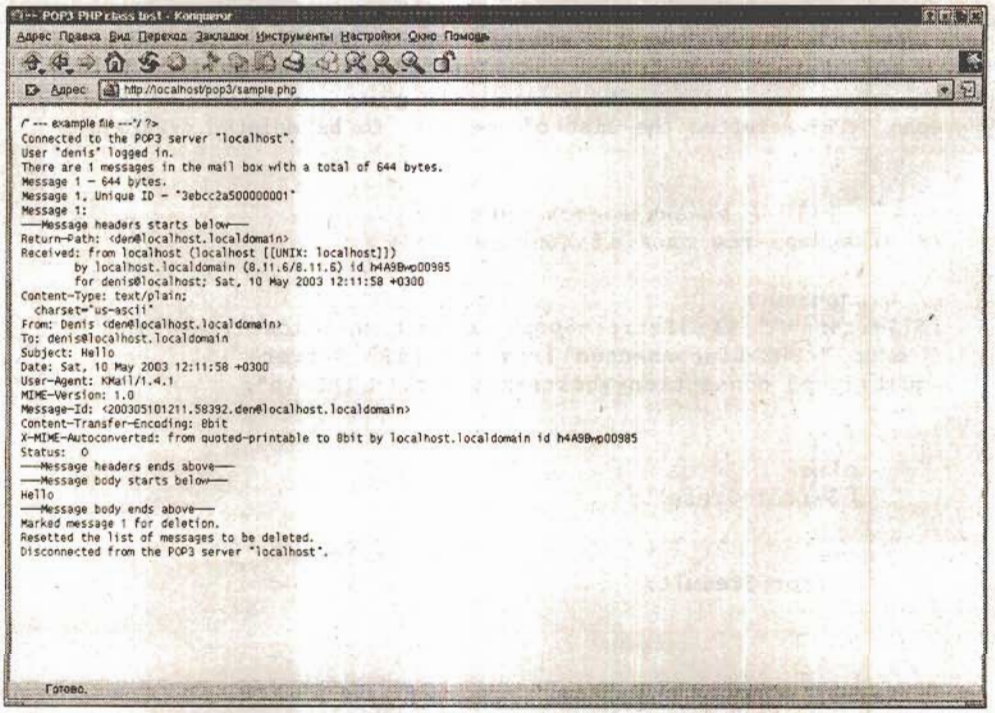


Рис. 15.2. *Есть сообщение*

Рассмотрим вывод сценарий подробнее — с моими комментариями:

// Мы подключились к серверу localhost и пользователь denis успешно зарегистрирован

```
Connected to the POP3 server "localhost".
User "denis" logged in.
```

// В ящике – 2 сообщения, суммарных объем 1316 байтов

```
There are 2 messages in the mail box with a total of 1316 bytes.
```

// Размер каждого сообщения

```
Message 1 - 644 bytes.
Message 2 - 672 bytes.
```

// Уникальные идентификаторы каждого сообщения

```
Message 1, Unique ID - "3ebcc2a500000001"
Message 2, Unique ID - "3ebcc2a500000002"
```

// Сообщение 1, его заголовки и тело

```
Message 1 :
-Message headers starts below--
Return-Path: <den@localhost.localdomain>
Received: from localhost (localhost [[UNIX: localhost]])
  by localhost.localdomain (8.11.6/8.11.6) id h4A9Bwp00985
  for denis@localhost; Sat, 10 May 2003 12:11:58 +0300
Content-Type: text/plain;
  charset="us-ascii"
From: Denis <den@localhost.localdomain>
To: denis@localhost.localdomain
Subject: Hello
Date: Sat, 10 May 2003 12:11:58 +0300
User-Agent: KMail/1.4.1
MIME-Version: 1.0
Message-Id: <200305101211.58392.den@localhost.localdomain>
Content-Transfer-Encoding: 8bit
X-MIME-Autoconverted: from quoted-printable to 8bit by
localhost.localdomain id h4A9Bwp00985
Status:  O
-Message headers ends above--
-Message body starts below--
Hello
-Message body ends above--
```

// Отмечаем сообщение 1 для удаления

```
Marked message 1 for deletion.
```

// Сбрасываем список сообщения для удаления

```
Resetted the list of messages to be deleted.
```

// Сообщение 2, его заголовки и тело

```
Message 2 :
--Message headers starts below--
Return-Path: <den@localhost.localdomain>
Received: from localhost (localhost [[UNIX: localhost]])
   by localhost.localdomain (8.11.6/8.11.6) id h4A9HTd01035
   for denis@localhost; Sat, 10 May 2003 12:17:29 +0300
Content-Type: text/plain;
   charset="us-ascii"
From: Denis <den@localhost.localdomain>
To: denis@localhost.localdomain
Subject: How are youe?
Date: Sat, 10 May 2003 12:17:29 +0300
User-Agent: KMail/1.4.1
MIME-Version: 1.0
Message-Id: <200305101217.29233.den@localhost.localdomain>
Content-Transfer-Encoding: 8bit
X-MIME-Autoconverted: from quoted-printable to 8bit by
localhost.localdomain id h4A9HTd01035
Status:  O
--Message headers ends above--
--Message body starts below--
Hi Denis!
--Message body ends above--
```

// Помечаем для удаления и опять сбрасываем список, чтобы

// не удалить сообщения — они

// нам еще пригодятся для дальнейшего тестирования

```
Marked message 2 for deletion.
Resetted the list of messages to be deleted.
```

// Отключаемся

```
Disconnected from the POP3 server "localhost".
```

Немного подытожим всю эту главу. В ней мы научились отправлять и принимать сообщения. Теперь наших знаний вполне хватит, чтобы написать свой собственный Web-клиент для работы с электронной почтой. Конечно, вам еще пригодится материал всех последующих глав, например, п. 22.4, в котором рассматривается загрузка файлов на сервер.

Глава 16

Делаем чат для своего сайта

16.1. Принцип создания чата

Нужен ли вашему сайту чат? Если ваш сайт постоянно обновляется, значит его должны помещать постоянные пользователи. Пусть их немного — пять, десять или пятьдесят человек. Ваш сайт посвящен теме, которая объединяет эту группу людей. Так почему бы не предусмотреть возможность общения постоянных посетителей? Думаю, что я убедил вас — чат нужен.

Как сделать свой чат? Мы сейчас рассмотрим идею создания простого чата — на одну комнату. В любом чате должен быть модуль регистрации, который проверят имя пользователя и пароль посетителя, а также позволяет зарегистрировать нового посетителя.

Если введенные имя пользователя и пароль правильны, пользователь попадет в комнату общения. Ну, а дальше, думаю, все ясно: пользователь вводит сообщение, оно передается на сервер и записывается в файл. Причем записываться оно должно в начало файла (мы рассматривали пример записи в начало файла в этой книге), чтобы пользователи видели новое сообщение первым. Модуль, который записывает информацию в файл, после записи должен выводить содержимое этого файла, причем не весь файл, а только первые n записей.

Формат файла может быть таким:

```
<login>:Сообщение
```

Писать чат мы не будем. Почему? Да потому, что я не вижу смысла заново изобретать велосипед — уже написано очень много чатов, причем доступен исходный код.

Если у вас есть время и желание написать самый лучший чат — возьмите исходники профессионального чата, например, чата CHAT.PHP.SPB.RU, посмотрите, как он написан, и сделайте свой. Неплохой учебник, в котором описаны основы создания простенького чата, можно найти по адресу `http://dushik.iatp.org.ua/lessons/1`.

Далее мы рассмотрим два чата: SimpleChat, который соответствует своему названию — он очень простой, и профессиональный чат CHAT.PHP.SPB.RU. Если вам не понравятся оба чата, посетите страницу `http://www.softlinks.ru/scripts/s34p1.php` — вы найдете много ссылок на различные чаты — от самых простых, до самых сложных.

16.2. Простой чат – SimpleChat

Если вы заинтересованы в посещаемости вашего сайта, то кроме интересной информации, привлекательного дизайна, желательно наличие еще двух атрибутов:

- Чата, в котором постоянные посетители смогут поделиться опытом, обсудить какие-нибудь связанные с тематикой сайта проблемы или просто пообщаться.
- Гостевой книги, которая представляет собой кладезь мнений о вашем сайте — такая себе книга жалоб и предложений.

Перепробовав десяток различных чатов, я остановил свой выбор на сценарии SimpleChat! версии 1.3 (`http://hot-things.net`).

SimpleChat — это простой HTML-чат, написанный на PHP 4. Чат работает довольно быстро, а вывод сценария небольшой, поэтому большой нагрузки на трафик не будет. Желательно, чтобы ваш браузер поддерживал HTML4 и JavaScript 1.2 — в этом случае чат будет выглядеть привлекательнее.

Основными преимуществами **SimpleChat** являются:

- Быстрая загрузка (всего 3...9 Кб за каждое обновление).
- Простая установка.
- Не требует MySQL.
- Поддерживается форматирование текста — жирный, курсив, а также выделение текста цветом.
- 20 цветовых схем.
- Различные пиктограммы пользователей — пользователь сам выбирает свою пиктограмму.
- Поддержка смайликов и гиперссылок.
- Фильтр HTML-тегов.
- Автоматическое обновление через 5, 10, 20, 30 или 60 секунд.
- 150 строк истории (предыдущих сообщений).
- Список online-пользователей.

- Звуковое сопровождение — при получении нового сообщения вы услышите звуковой сигнал.
- Переведен на разные языки, в том числе и на русский.
- Оптимизирован под браузеры MSIE 4+, Netscape 4+, Opera 4+, Mozilla.

Такое обилие функций для чата, который в архиве занимает 34 Кб — довольно неплохо. В то же время, можно выделить сразу бросающиеся в глаза недостатки:

- Отсутствие аутентификации пользователей.
- HTML-чат довольно легко взломать — получить возможность писать сообщения от имени другого пользователя.
- Отсутствие приватной комнаты.

Но, с другой стороны, эти недостатки не кажутся такими уж значительными, чат — это ведь не место для ведения серьезных переговоров. Для этого рекомендуется использовать SSL-соединения.

Что мне еще понравилось в этом чате — это простота установки. Чат работает сразу после установки — не нужно ничего изменять или переписывать заново, что часто бывает при использовании сценариев такого рода. Единственное, что я изменил — это настройки языка.

16.2.1. Установка, настройка и работа с чатом

Распакуйте архив schat.zip в каталог `/var/www/html/schat`. Сразу измените права доступа к каталогу data:

```
chmod 777 /var/www/html/schat/data
```

Примечание.
Если вы работаете под Windows, этого делать не нужно.

Затем откройте файл `incl/hdr.inc` и измените последнюю строку `include "lang/lang-en.inc";` строкой `include "lang/lang-ru.inc";`. Этим мы подключим русский язык.

После этого чат уже должен работать. Для этого в строке браузера введите следующий URL:

```
http://localhost/schat/index.php
```

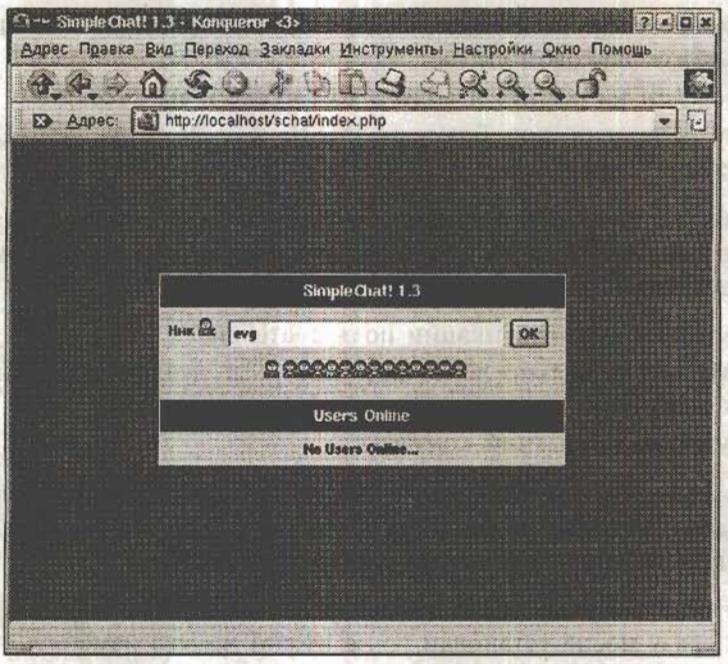


Рис. 16.1. *Регистрация*

Сейчас введите свой ник и выберите пиктограмму (см. рис. 16.2).

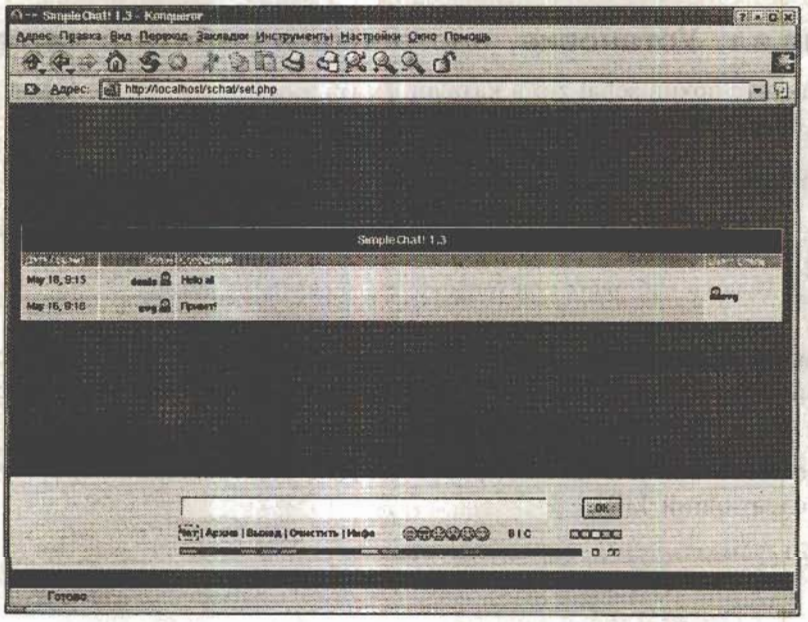


Рис. 16.2. *Чат-комната*

SimpleChat отображает сообщения в формате «Дата/Время», «Пользователь», «Сообщения». Справа вы можете видеть список online-пользователей — тех, кто работает с чатом в данный момент.

В нижней части окна находится панель управления чатом. Здесь вы можете:

- Отправить простое сообщение.
- Отправить форматированное сообщение (допускается использование цвета, изменение начертания, вставка смайликов и гиперссылок).
- Изменить цветовую гамму (в нижней части панели находятся 20 цветных прямоугольников — это кнопки выбора цветовой схемы).
- Включить/выключить звуковое сопровождение.
- Изменить режим работы чата: Чат/Архив/Выход/Очистить/Инфо.

Наиболее часто используемыми вами режимами будут «Архив» и «Выход». С выходом-то все ясно. А вот «Архив» (см. рис. 16.3) придется немного модифицировать.

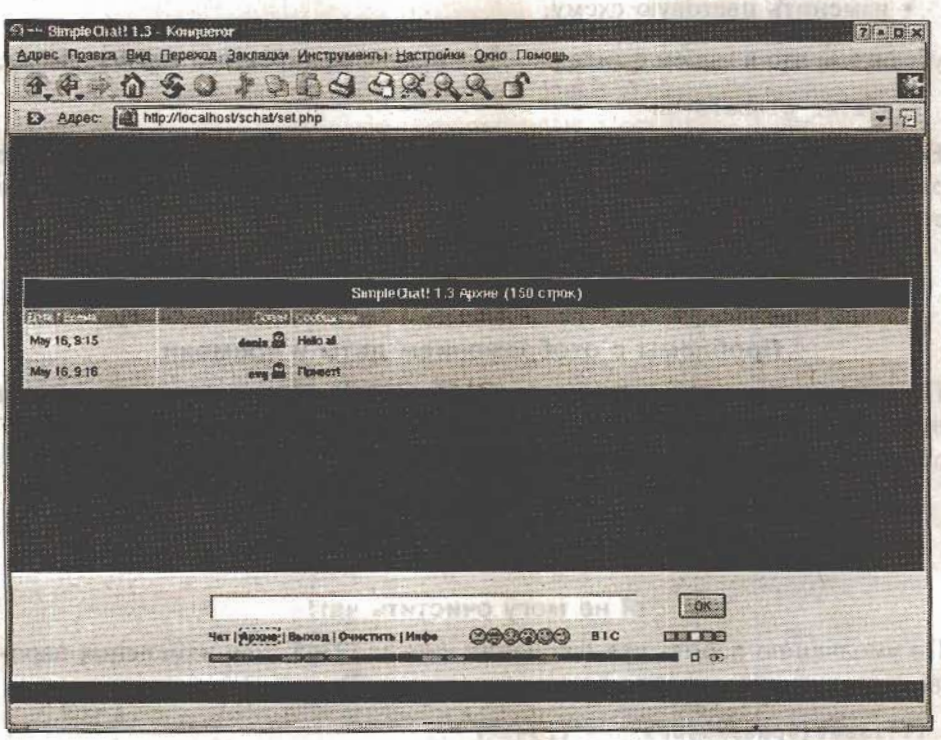


Рис. 16.3. Архив сообщений

По умолчанию в этом режиме отображаются 150 последний сообщений. Если вы хотите это число увеличить (или уменьшить), придется отредактировать файл index.php, строка 45:

```
for($i=0;$i<150;$i++){if(isset($myar[$i])){$nr=$nr.$myar[$i]."\n";}}wrl($nr);
```

Просто замените константу в выражении цикла нужным вам числом.

Вот, собственно, и все. Правда, не совсем. У нас все очень хорошо и плавно — по принципу «пришел, увидел, победил». Может случиться и такое, что чат у вас работать не будет или будет работать некорректно.

16.2.2. Проблемы при установке

Некорректная работа

Если вы не можете:

- войти;
- отправить сообщение;
- изменить цветовую схему,

убедитесь, что в вашем файле php.ini включена директива **register_globals**:

```
register_globals=On
```

Если же изменить файл php.ini невозможно — обычно провайдеры не разрешают редактировать файлы конфигурации, то добавьте следующую строку в начало файла incl/hdr.inc:

```
ini_set('error_reporting',7);import_request_variables("GPC");
```

Проблемы с отображением даты и времени

По умолчанию чат настроен на GMT (UTC). Для настройки чата на нужный вам часовой пояс, измените 0 в первой строке файла incl/hdr.inc нужным вам значением (+/– Число):

```
$datetime=gmdate('M j, G:i',time()+0*3600);
$datetime=gmdate('M j, G:i',time()+3*3600);
```

Я не могу очистить чат!

По умолчанию пароль для очищения чата – 12345. Для изменения пароля отредактируйте файл clear.php (строка 7):

```
if(isset($pass)&&$pass=="12345") ...
```

16.2.3. Полезные советы

Модульная структура чата открывает довольно интересные особенности. Например, вы можете легко перевести его на другой язык, вставить печать списка online-пользователей в любую страницу вашего сайта и многое другое.

Начнем с самого простого — перевода чата на другой язык. Если вы хотите перевести чат на какой-нибудь язык, скопируйте файл `lang/lang-ru.inc` в файл `lang/lang-XX.inc` и переведите сообщения на нужный вам язык, например, украинский.

Вот фрагмент файла `lang-ru.inc`:

```
<?php
$crl[0]='10';
$crl[1]='<meta http-equiv="Content-Type" content="text/html;
charset=windows-1251">';
$crl[2]='SimpleChat! 1.3';
$crl[3]='Sound: On/Off';
$crl[4]=': OK :';
$crl[5]='Отмена';
$crl[6]='Пароль';
$crl[7]='Поменять цвет...';
$crl[8]='Злость';
$crl[9]='Здорово';
$crl[10]='Огорчение';
$crl[11]='Кричать';
```

Тут же можно изменить кодировку сообщений.

Возможно, вы хотите просмотреть полный протокол чата без вызова самого чата. Откройте файл `data/log` — в нем вы найдете все, что вам нужно:

```
1053076834:|:May 16, 9:20:|:evg:|:w1:|: <b></b>
1053076782:|:May 16, 9:19:|:evg:|:w2:|:Как дела? <img
src="pics/m5.gif" width="15" height="15" hspace="2" alt="">
1053076582:|:May 16, 9:16:|:evg:|:w2:|:Привет!
1053076536:|:May 16, 9:15:|:denis:|:w1:|:Hello all
```

Вы можете модифицировать сценарий, добавив в него запись в протокол IP-адреса, если вам это нужно.

Используя возможности чата, вы можете отображать список online-пользователей на любой странице вашего сайта.

Для этого подключите файл `user.inc` к вашей странице:

```
<?php $pth='xxx';include "$pth/incl/users.inc"; ?>
```

XXX — это относительный путь к чату.

После этого вызовите функцию **whos_online()**, передав ей строку-разделитель пользователей.

```
<?php whos_online('<br>'); ?>
```

Пример файла `index.php` с использованием функции **whos_online()** приведен в листиге 16.1.

Листинг 16.1. Вывод списка пользователей

```
<?php $pth='friends/simple-chat';include "$pth/incl/users.inc";
?>

<html><head>
<title>Основная страница</title>
</head><body>

<!-- список в строку -->
<div style="background-color:#cccccc">
<?php whos_online(' '); ?>
</div><br>

<!-- список - в колонку -->
<div style="background-color:#aaaaaa">
<?php whos_online('<br>'); ?>
</div><br>


</body></html>
```

16.3. Профессиональный чат CHAT.PHP.SPB.RU

Чат CHAT.PHP.SPB.RU разработан Дмитрием Бородиным (dima@php.spb.ru) и доступен по адресу `http://php.spb.ru/chat`. Почему данный чат называется профессиональным или, как его называет разработчик, «правильным»? Достаточно взглянуть на особенности данного чата:

- Сайт не использует MySQL. Почему это хорошо? Во-первых, вы можете не покупать хостинг с поддержкой MySQL и будете экономить до $10 в месяц. Во-вторых, отсутствие поддержки MySQL упрощает установку чата.
- Сайт не использует Java-апплетов и Flash-анимации.
- Отсутствие многооконности.
- Плавная загрузка сообщений.
- Подсветка сообщений при обращении к вам.
- Индикаторы связи вашего компьютера с Интернетом и с чатом.

- Постоянные соединения с сервером. Вы можете «сидеть» в чате сутками.
- Дополнительные скины.
- Минимальные требования к памяти: чату нужно всего 5 Мб для обслуживания 100 пользователей.
- Чат рассчитан на большое количество пользователей — у вас не возникнут проблемы с 1000 пользователей, которые работают одновременно.
- Встроенная защита от флуда.
- Передача по сети только фраз пользователей, а не HTML-кода, что снижает нагрузку на трафик.
- Регистрация новых пользователей, причем вам не удастся выбрать ник, похожий на уже зарегистрированный.
- Чат позволяет изменять регистрационную анкету и даже удалять свой логин.
- Интеграция с базами пользователей от любого форума на MySQL — Phorum.org, phpBB, phpNuke и др.
- Настройка уровня безопасности, что актуально для Интернет-кафе.
- Чат позволяет создавать чат-порталы, то есть объединять несколько чатов.

И это далеко не все возможности чата. На сайте `http://php.spb.ru/chat` вы найдете не только исходники чата, но и подробную документацию с описанием всех возможностей.

Установка и настройка чата под Windows

Чат устанавливается полностью автоматически и очень быстро — за 10...15 секунд, в зависимости от производительности вашего компьютера. Скачайте файл `chat-php-spb-ru-04_60.exe (http://php.spb.ru/chat/)`, запустите его. Это обыкновенный самораспаковывающийся RAR-архив. Нажмите кнопку "Извлечь" и подождите, пока чат будет распакован в каталог `c:\www` (каталог можно изменить).

После этого должны запуститься два черных окошка, которые не нужно закрывать. Через пару секунд запустится Internet Explorer и откроет страницу `http://127.0.0.1/`. Все! Чат готов к работе.

Если автоматическая установка у вас не прошла (как у меня), значит, сейчас самое время приступить к настройке чата вручную. Откройте файл `c:\WWW\chat-php-spb-ru-04\www\__dima.php`, найдите в нем строку:

```
"$os=0";
```

Измените значение переменной `$os` на:

- 3, если у вас Windows 9x/ME;
- 2, если у вас Windows NT/2000/XP.

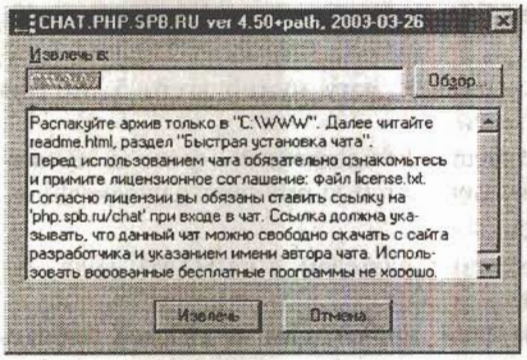


Рис. 16.4. *Распаковка чата*

Потом найдите строки `"$rand_password=.."` и `"$cookie_pass=.."` и впишите в них два любых слова, желательно вписать не слова, а абсолютно случайный набор символов, например, `XudfjHdtujn7dkjfd87`.

После этого найдите переменную `$adminfirst` и присвойте ей значение — ник администратора чата:

```
$adminfirst = "admin";
```

Потом запустите файл `c:\www\chat-php-spb-ru-04\apache\apache.exe` — это обыкновенный сервер Apache для Windows. Если сервер не запустился, то вам придется его настроить. Как это делать, вы уже знаете.

Если Apache запустился нормально, запустите файл `c:\WWW\chat-php-spb-ru-04\daemon\run_this.bat`. Откроется еще одно черное окошко, а затем — Internet Explorer. Если последний не запустился, запустите его вручную (http://127.0.0.1).

Теперь вы можете приступить к настройке чата «под себя». Если у вас возникнут какие-нибудь проблемы, обратитесь к разработчику чата.

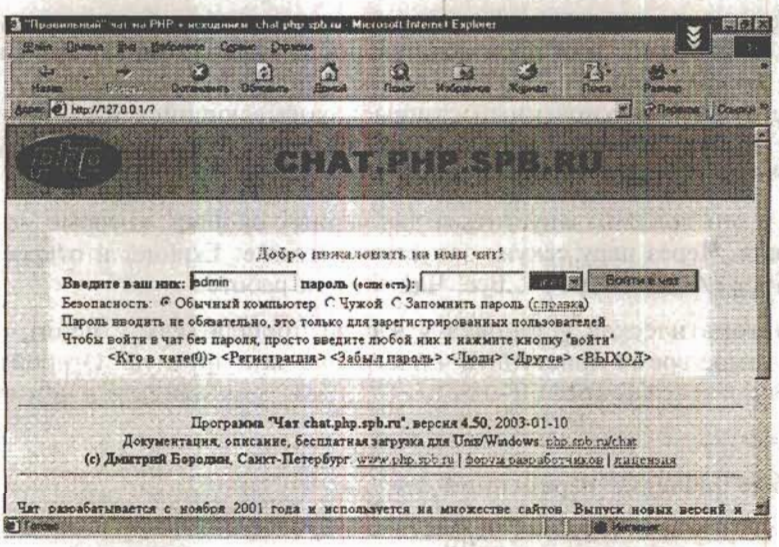


Рис. 16.5. *Чат в окне Internet Explorer*

Установка и настройка чата под Linux

Скачайте файл chat-php-spb-ru-04_60.exe (это SFX RAR) с сайта php.spb.ru. Да, это тот же самый файл, который используется для установки под Windows. Вам понадобится Windows для распаковки RAR-архива. Просто распакуйте его в какой-нибудь каталог, а потом я скажу, что с ним делать.

Также вам еще понадобится дополнительный файл — php.spb.ru/files/chat-php-spb-ru-04_06_unix.zip — он содержит специфические для Linux настройки.

Создайте нового пользователя — wwwchat (группа — тоже wwwchat). Все дальнейшие операции будем выполнять от имени этого пользователя. Если вы не root, тогда выполняйте все от своего имени.

Пусть ваш чат будет расположен в каталоге /home/wwwchat/chat. Вот будущая структура этого каталога (пока ничего создавать не нужно):

```
/home/wwwchat                - домашний каталог пользователя wwwchat
/home/wwwchat/tmp            - каталог для временных файлов
/home/wwwchat/chat           - это корень чата
/home/wwwchat/chat/www       - PHP-скрипты
/home/wwwchat/chat/daemon    - чат-сервер
/home/wwwchat/chat/php_sess - файлы сессий PHP-скриптов
/home/wwwchat/chat/readme.html - этот файл вы сейчас читаете (копию)
```

Структура каталога **daemon**:

```
...daemon/            - чат-сервер
...daemon/db          - база данных юзеров: пароли не шифруются!
...daemon/lib         - библиотеки windows-версии и чат-сервера
...daemon/logs        - протоколы чат-сервера
...daemon/www-chat    - временные файлы PHP-скриптов
...daemon/www-conf    - конфигурация чата
...daemon/www-server  - временные файлы чат-сервера
...daemon/www-sess    - временные файлы чат-сервера и PHP-скриптов
```

Если вы root, то измените права доступа (из соображений безопасности):

```
chmod -R wwwchat:group_chat /home/wwwchat
chown 750 /home/wwwchat
```

Теперь проверим версию библиотеки TCL (чат использует не только PHP, а TCL, что значительно снижает загрузку памяти и увеличивает производительность). Введите команду **tclsh** или **tclsh83**. Если у вас нет этой команды, тогда вам нужно установить библиотеку TCL. У вас есть библиотека TCL? Тогда узнаем ее версию — введите команду **tclversion**. Для работы чата нужен TCL версии 8.2 или выше. Скорее всего, у вас будет

установлена подходящая версия. Если же нет, тогда обновите версию TCL или попросите сделать это администратора хостинга. Новую версию TCL можно взять здесь — `http://prdownloads.sourceforge.net/tcl/tcl8.3.4.tar.gz`.

Мы подготовили нашу систему к установке чата. Распакуйте скачанный файл (SFX архив) в каталог `/home/www/chat/www`. Ясное дело, что вам сначала нужно будет распаковать его в Windows, а затем перезагрузиться в Linux и скопировать его из Windows-раздела. Напомню, что команда для монтирования Windows-раздела выглядит так:

```
mount  /dev/hdaN /mnt/c  (N - это номер вашего раздела)
```

Если же вы не root, а устанавливаете чат на сервер хостинг-провайдера, тогда просто распакуйте его в ваш домашний каталог (точнее в подкаталог chat).

После этого откройте файл `__dima.php` (`/home/wwwchat/chat/www/__dima.php`) и укажите следующие параметры:

- `$wwwhost` — этот параметр должен совпадать с директивой **ServerName** сервера Apache;
- `$os = 1` — мы используем Linux.

Откройте файл `/home/wwwchat/www/daemon/dima_chat.conf`. Найдите переменную `$wwwhost` — ее значение должно совпадать со значением переменной `$wwwhost` в файле `__dima.php`.

Изменим права доступа:

```
chmod 777 daemon/db
chmod 777 daemon/logs
chmod 777 daemon/www-chat
chmod 777 daemon/www-conf
chmod 777 daemon/www-conf/*
chmod 777 daemon/www-server
chmod 777 daemon/www-sess
chmod 755 daemon/down
chmod 755 daemon/libtbcload1.3.so
chmod 755 daemon/ps
chmod 755 daemon/run_background
chmod 755 daemon/run_this
```

Запустите файл `/home/wwwchat/www/run_this`. Если вы увидите следующий вывод, значит чат настроен правильно:

```
Config loaded. Your TCL version: 8.3. Wait please...
[11:25] ____________________________________
[11:25]          CHAT.PHP.SPB.RU, ver 4.06, 06 September 2002
[11:25] ____________________________________
```

```
[11:25]   Chat software (c) Dmitry Borodin, Russia, St.-Petersburg
[11:25] ------------------------------------
[11:25] Before using, please read licence on http://php.spb.ru/chat!
[11:25] ------------------------------------
[11:25] PID: 10495, TCL Version: 8.3.4, hostname: localhost
[11:25] Listing localhost:8888 ...
[11:25] Encoding iso8859-1 ...
[11:25] Found charset for IRC-frame: windows-1251.
[11:25] Searching old session file...
[11:25] Opening main chat file: www-chat/chat.txt ...
[11:25] ok, file clean
[11:25] *** LOADED (0.35 sec) ***
```

А если не увидите, тогда повторите настройку, возможно, вы где-то допустили ошибку. Также рекомендую прочитать файл `unix.html`, который вы найдете в архиве вместе с исходниками чата. В этом файле установка под Linux описана очень подробно.

Глава 17

Голосование на сайте

17.1. Делаем голосование сами

Вы хотите знать, какой дизайн сайта больше нравится вашим пользователям — старый, новый или вообще ни тот, ни этот? А может вам интересно знать, какую операционную систему предпочитают пользователи? Тогда вам не обойтись без голосования — нужно ведь знать глас народа.

Сейчас мы напишем небольшой сценарий, позволяющий оценить дизайн нашего сайта. Вариантами ответов будут:

- Новый.
- Старый.
- Никакой.
- Мне все равно.

Наша «система» обработки голосования будет состоять из трех файлов:

- `vote.php` — обрабатывает ответ пользователя и записывает его в файл `vote.txt`.
- `vote.txt` — здесь будут храниться ответы пользователей.
- `vote.html` — а это форма, содержащая варианты ответов.

Начнем с самого простого — напишем форму с вариантами ответов (см. листинг 17.1).

Листинг 17.1. Файл vote.html

```
<FORM METHOD="POST" action="vote.php">
<TABLE border="2" cellspacing="0" cellpadding="3"
bordercolor="#336699">
<TR><TD><TABLE BORDER=0>
<TR><TD bgcolor="#336699"><font color=white>Какой дизайн вам
больше нравится?</font></TD></TR>

<TR><TD><INPUT TYPE=radio NAME=answer VALUE=1>Новый</TD></TR>
<TR><TD><INPUT TYPE=radio NAME=answer VALUE=2>Старый</TD></TR>
```

```
<TR><TD><INPUT TYPE=radio NAME=answer VALUE=3>Никакой</TD></TR>
<TR><TD><INPUT TYPE=radio NAME=answer VALUE=4>Мне все равно</
TD></TR>

<TR><TD><INPUT TYPE=Submit NAME=vote VALUE=" Голосовать! "></
TD></TR>
<TR><TD><INPUT TYPE=Submit NAME=result VALUE=" Результат "></
TD></TR>
</TABLE></TD></TR></TABLE></FORM>
```

Теперь приступим к созданию файла `vote.txt` (см. листинг 17.2).

Листинг 17.2. Файл vote.txt

```
Результаты
0
0
0
0
```

В качестве первой строки вы можете указать все, что угодно. Нули — это начальные результаты — ведь пока еще никто не голосовал. Первый нуль — это количество голосов за новый дизайн сайта, второй — за старый, и т.д.

Вот теперь мы готовы написать сценарий, обрабатывающий ответы пользователей (см. листинг 17.3).

Листинг 17.3. Сценарий vote.php

```
<?
// Это файл, содержащий результаты
$file = "vote.txt";
$data = file($file);
$i = 1;
// Количество строк в файле
$qty = Count($data);
$n = 0;

while ($i <= $qty):
$data[$i] = trim(str_replace ("\n","", $data[$i]));
$n = $n+$data[$i];
$i++;
endwhile;

if ($answer != "") {
echo "<br>Спасибо!";
```

```
$data[$answer]++; $n++;

$res = "Результаты голосования\n".$data[1].
"\n".$data[2]."\n".$data[3]."\n".$data[4];
$fp = @fopen($file,"w");

if ($fp) { $counter=fputs($fp,$res); fclose($fp); }
else { echo "Ошибка записи в файл"; }

} else { echo "<br>Результаты голосования"; }

echo "<br>Новый - <b>".$data[1]."</b>";
echo "<br>Старый - <b>".$data[2]."</b>";
echo "<br>Никакой - <b>".$data[3]."</b>";
echo "<br>Мне все равно - <b>".$data[4]."</b>";
echo "<br><br>Всего голосов: ".$n;

?>
```

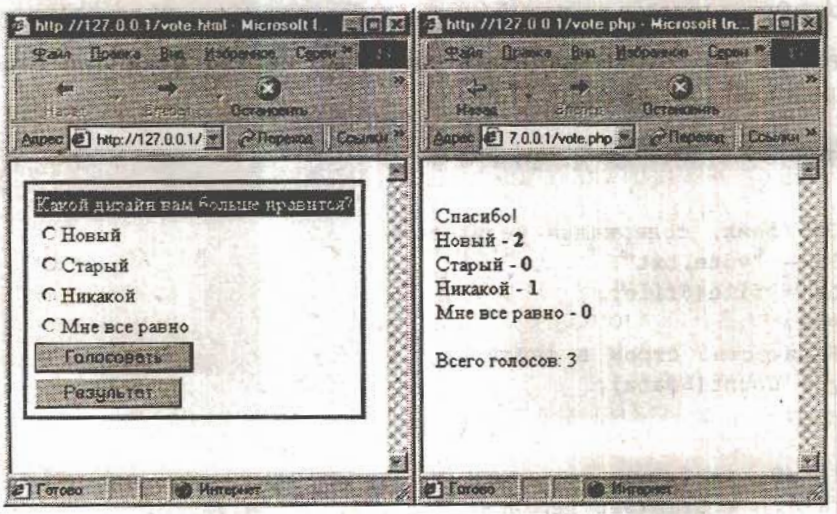


Рис. 17.1. *Сценарий голосавания vote в работе*

В следующем пункте мы рассмотрим целую систему голосований, позволяющую обрабатывать не одно голосование, а несколько.

17.2. Используем профессиональный бесплатный скрипт Advanced Poll

Общее описание и установка Advanced Poll

Лучшим скриптом для голосования, на мой взгляд, является Advanced Poll. Данный сценарий позволяет создавать голосования и централизованно управлять ними. Поддерживается работа как с базой данных MySQL, так и с обыкновенным текстовым файлом, что чрезвычайно удобно, если ваш хостинг-провайдер по тем или иным причинам не позволяет вам использовать MySQL.

Скачайте скрипт по адресу `http://www.proxy2.de/scripts.php` (скрипт называется Advanced Poll, имя файла `pollphp.zip`). Распакуйте его в какой-нибудь локальный каталог. В архиве будет два каталога — `db` и `textfile`. Первый каталог содержит систему голосования с поддержкой MySQL, а второй каталог содержит версию системы, использующую текстовый файл. В этой главе мы рассмотрим именно второй вариант, так как он немного проще в установке на сервер и не требует наличия сервера баз данных.

Скопируйте содержимое каталога `textfile` в каталог `$DocumentRoot/poll (/var/www/html/poll)`. Структура каталогов будет выглядеть следующим образом:

```
/var/www/html/poll
```

admin — средства управления системой;
image — картинки;
include — файлы вложений;
lang — настройки языка;
misc — дополнительные возможности (*);
polldata — данные голосований;
templates — шаблоны.

Примечание.
Лично у меня в каталоге **misc** находится один пустой (!) файл info.php. Для чего он используется, не спрашивайте — я сам не знаю.

После этого от вас потребуется выполнить пару несложных трюков, чтобы заставить систему работать на вашем сервере. Первое, что нужно сделать — изменить права доступа к файлам и каталогам.

Установите права доступа для каталога `/var/www/html/poll/polldata` равные 777 (разрешает чтение, запись и просмотр каталога всем пользователям). Для всех файлов, находящихся в этом каталоге, установите права доступа 666 (разрешает чтение и запись, но запрещает выполнение). После

этого, установите права 777 для каталога `templates` (и для всех его подкаталогов. Все файлы, которые содержат подкаталоги каталога `templates` и сам каталог, должны иметь права доступа 666.

Примечание.
Право чтения для каталога означает право чтения файлов из каталога, право записи – право записывать файлы (модифицировать, создавать новые), а право выполнения (или просмотра) для каталога означает просмотр оглавления. Более подробно о правах доступа вы сможете прочитать в гл. 11.

Права доступа для каталога `include` — 755 (владельцу, то есть нам, разрешено чтение, запись и просмотр каталога, всем остальным — только чтение и просмотр), а для файла `inlude/config.inc.php` — 666 (читаем, изменяем и запускаем). Вот и все! Теперь можно приступать к настройке голосования.

Настройка Advanced Poll. Центр управления

Запускаем «центр управления»: `http://узел/poll/admin/index.php`. Параметры для входа в систему:

- Имя пользователя admin.
- Пароль poll.

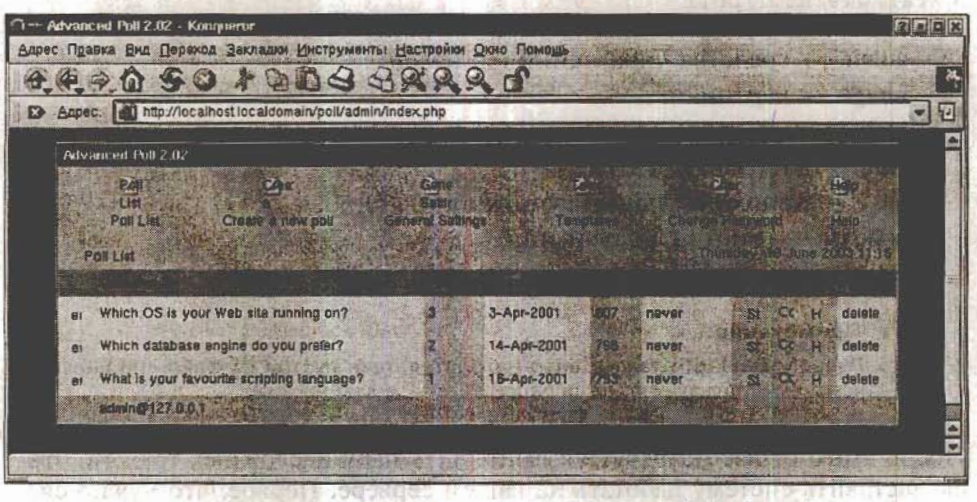


Рис. 17.2. Центр управления системой Poll

В верхней части окна вы увидите шесть импровизированных кнопок:

- **Poll List** — список голосований;
- **Create a new poll** — создать новое голосование;
- **General Settings** — параметры;

- **Templates** — управлене шаблонами;
- **Change password** — изменение пароля администратора;
- **Help** — помощь.

Первое, что нужно сделать — это изменить пароль по умолчанию (см. рис. 17.3). Ведь если кто-нибудь узнает, что у вас установлена система Poll, а вы не изменили стандартный пароль, это вполне может закончиться или удалением ваших анкет, или их изменением.

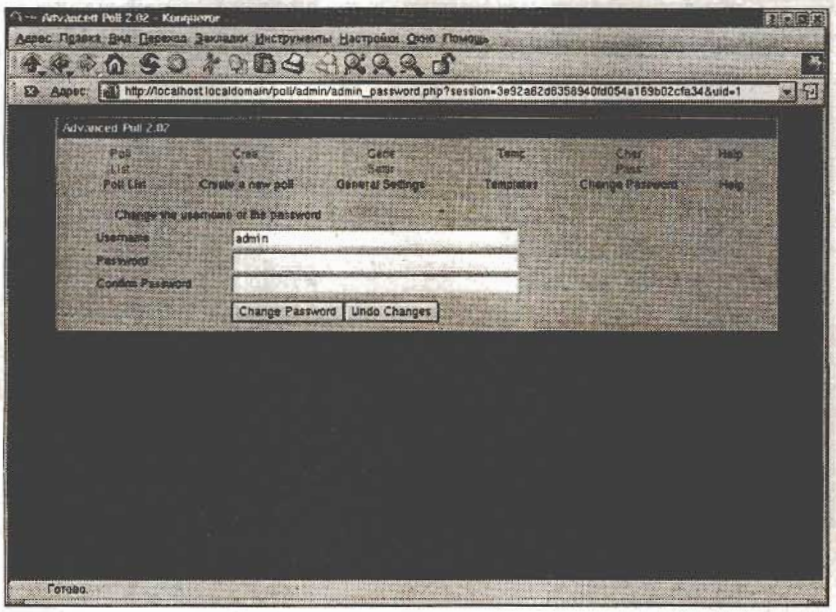


Рис. 17.3. *Изменение пароля*

Теперь изменим параметры самой системы (см. рис. 17.4). Нам нужно изменить язык системы, а также некоторые другие параметры. Зайдите в раздел `General Settings`.

Если при нажатии кнопки **Submit** вы получаете сообщение `Invalid value`, значит вам не повезло, и придется редактировать файлы конфигурации вручную. Можно, конечно, искать ошибку в самом коде сценария, но гораздо проще открыть в любом текстовом редакторе файл `english.php`, который находится в каталоге `lang`. В нем нужно сделать следующие изменения:

```
/* Установим русскую кодировку - koi8-r */
$lang["charset"]    = "koi8-r";
```

```
/* По своему желанию вы можете перевести сообщения на русский
язык, но если у вас
     есть файл russian.php, вы можете его просто скопировать в
english.php  и все будет работать нормально
*/
...
```

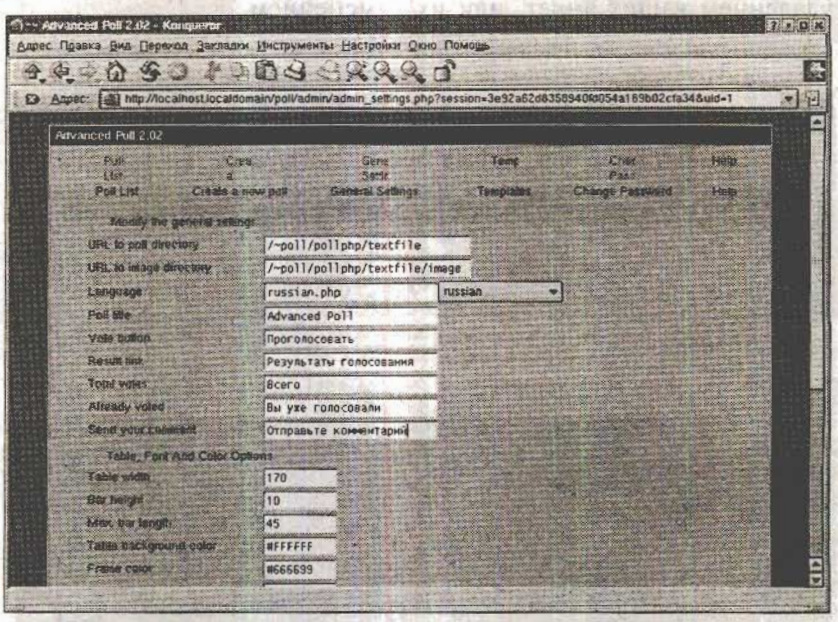


Рис. 17.4. *Изменение параметров системы Poll*

Затем откройте файл `include/config.inc.php` и измените его, как показано в листинге 17.4.

Листинг 17.4. Файл include/config.inc.php

```
<?php
$pollvars['base_url'] = "/~poll/pollphp/textfile";
$pollvars['base_gif'] = "/~poll/pollphp/textfile/image";
$pollvars['lang'] = "english.php";
$pollvars['title'] = "Advanced Poll";
$pollvars['vote_button'] = "Проголосовать";
$pollvars['result_text'] = "Результаты";
$pollvars['total_text'] = "Всего";
$pollvars['voted'] = "Вы уже голосовали!";
$pollvars['send_com'] = "Ваш комментарий";
```

```
$pollvars['table_width'] = "170";
$pollvars['img_height'] = "10";
$pollvars['img_length'] = "45";
$pollvars['bgcolor_tab'] = "#FFFFFF";
$pollvars['bgcolor_fr'] = "#666699";
$pollvars['font_color'] = "#000000";
$pollvars['font_face'] = "Verdana, Arial, Helvetica, sans-
serif";
$pollvars['result_order'] = "desc";
$pollvars['check_ip'] = "0";
$pollvars['lock_timeout'] = "2";
$pollvars['type'] = "percent";
$pollvars['time_offset'] = "0";
$pollvars['polls_pp'] = "12";
$pollvars['entry_pp'] = "5";
$pollvars['def_options'] = "10";
/* следующие параметры изменять не нужно */
$pollvars['poll_username'] = "admin";
$pollvars['poll_password'] = "b0f6dfb42fa80caee6825bfecd30f094";
$pollvars['poll_version'] = "2.02";
?>
```

Создание голосования

Вот только теперь мы можем приступить к созданию голосования. Но перед этим рассмотрим список параметров для каждого голосования (Poll List, рис. 17.2):

- **Question** — вопрос голосования (тем обсуждения).
- **Poll ID** — идентификатор голосования. Данный идентификатор нужно указать при написании сценария голосования.
- **Created** — дата создания.
- **Days** — сколько дней прошло с момента создания.
- **Expiration** — дата окончания голосования.
- **Stats** — всевозможная статистика.
- **Action** — действие. Обычно в этой колонке стоит действие **delete**, позволяющее удалить голосование.

Вот теперь нажмите кнопку **Create a new poll** — сейчас будем создавать новое голосование (см. рис. 17.5).

Все предельно просто: пишите ваш вопрос (**Questions**) и варианты ответа на него (**Option X**). Можно включить протоколирование (**logging on**), установить начальное значение для каждой опции, а также выб-

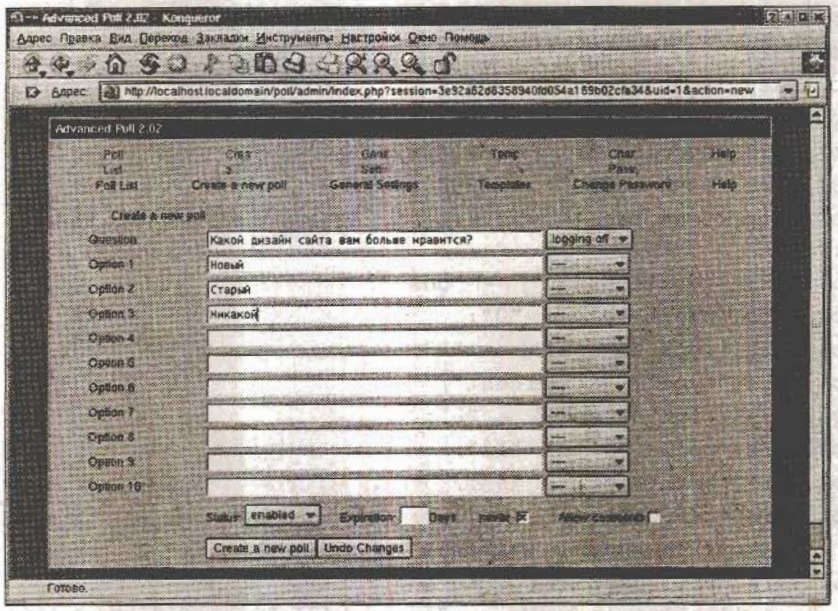


Рис. 17.5. *Создание нового голосования*

рать цвет опции, который будет использоваться при отображении результатов голосования.

В нижней части окна можно установить следующие параметры голосования:

- Активно ли оно (**enabled**) или пассивно (**disabled**), то есть будет ли это голосование отображено в списке голосований или нет.
- Через сколько дней оно станет пассивным (**Expiration Days**) или никогда (**never**).
- Разрешается ли пользователям оставлять комментарии (**Allow comments**).

Все, теперь можно смело нажать кнопку **Create a new poll,** и голосование будет создано. В этом можно убедиться, посмотрев в список голосований.

Редактирование голосования

Для редактирования уже созданного голосования просто щелкните по нему мышью (см. рис. 17.6). Если вы хотите добавить дополнительные варианты ответа, нажмите кнопку **Add options**. А если вам нужно удалить один из вариантов, просто удалите текст-описание варианта.

Наше голосование создано. Что делать дальше? Как добавить форму голосования на наш сайт? Я советую вам подробно изучить сценарии `demo*.php`.

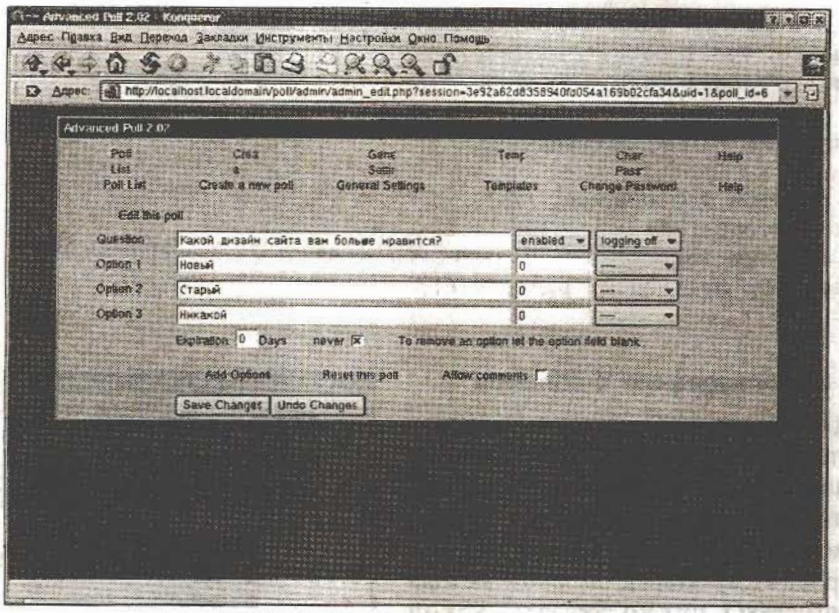


Рис. 17.6. Редактирование голосования

В них вы найдете нужные фрагменты кода. Сейчас мы напишем простенький сценарий (см. листинг 17.5), который будет отображать одно из созданных нами голосований. Загляните в центр управления, чтобы узнать его номер (Poll ID) — он нам пригодится для написания сценария. Результат работы нижеприведенного сценария показан на рис. 17.7.

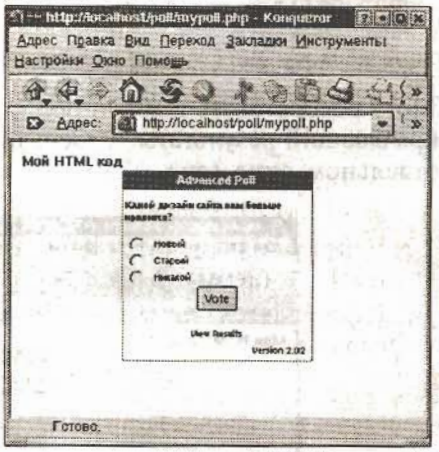


Рис. 17.7. Голосование

Листинг 17.5. Включение голосования в Web-страницу

```
<?php
/* Подключаем систему Poll  ПЕРЕД  вашим HTML-кодом*/
 include "./poll_cookie.php";
?>
 <b>Мой HTML код</b>
<?
/* путь к каталогу, в котором установлена система */
$poll_path = "/var/www/html/poll";
```

```
// подключаем файл конфигурации и файл класса
require $poll_path."/include/config.inc.php";
require $poll_path."/include/class_poll.php";

$php_poll = new poll();

/* простое голосование */
echo "<center>";
// Выводим голосование 6 (Poll ID = 6)
echo $php_poll->poll_process(6);

?>
```

Использование шаблонов

Предположим, что мы хотим использовать какой-нибудь стандартный шаблон. Установленные шаблоны находятся в каталоге `templates`. Сейчас будем использовать шаблон `simple` (см. рис. 17.8). Перед вызовом метода **poll_process()** нужно указать, какой шаблон мы хотим использовать:

```
$php_poll-
>set_template_set("simple");
echo $php_poll->poll_process(6);
```

Могу посоветовать шаблон `popup`, открывающий результаты голосования в отдельном окне (рис. 17.9).

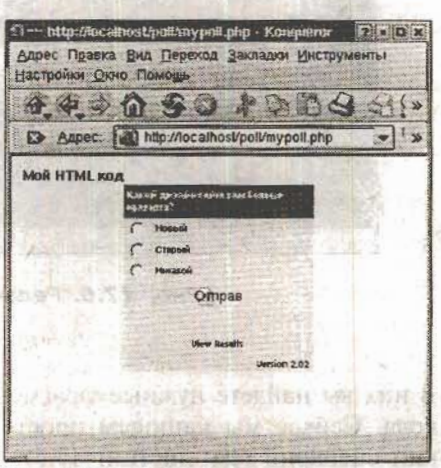


Рис. 17.8. Шаблон "Simple"

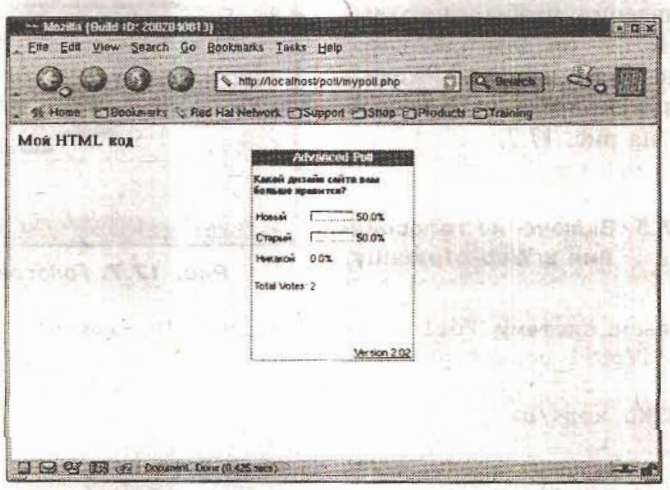


Рис. 17.9. Результаты голосования

Несколько голосований на одной Web-странице

Если вы хотите отобразить на одной странице несколько голосований — нет проблем: нужно просто несколько раз вызвать метод **poll_process()**, например:

```
$php_poll->set_template_set("simple");
echo $php_poll->poll_process(6);
echo $php_poll->poll_process(4);
echo $php_poll->poll_process(3);
```

Теперь попробуем вывести список доступных голосований. Для этого нужно использовать объект класса **plist()**, а не **poll()**, как в предыдущем примере.

```
require $poll_path."/include/config.inc.php";
require $poll_path."/include/class_poll.php";
require $poll_path."/include/class_pollcomment.php";
require $poll_path."/include/class_plist.php";

$php_poll = new plist();

$php_poll->set_template("poll_list");
$php_poll->set_date_format("m/d/Y");
echo $php_poll->view_poll_list();
echo $php_poll->get_list_pages();
```

Сценарий в листинге 17.6 выводит три голосования рядом друг с другом, а после них — список доступных голосований. Результат показан на рис. 17.10.

Листинг 17.6. Вывод нескольких голосований друг возле друга

```
<?php
 include "./poll_cookie.php";
?>

<?php

$poll_path = "/var/www/html/poll";

require $poll_path."/include/config.inc.php";
require $poll_path."/include/class_poll.php";
require $poll_path."/include/class_pollcomment.php";
require $poll_path."/include/class_plist.php";

$php_poll = new plist();

/* первое */
echo "<center><TABLE><TR><TD>";
echo $php_poll->poll_process(6);
echo "</TD><TD>";

/* второе */
```

```
  $php_poll->set_max_bar_length(80);
 echo $php_poll->poll_process(2);
 echo "</TD><TD>";

 /* третье */

  $php_poll->set_template_set("popup");
  if ($php_poll->is_valid_poll_id(3)) {
       echo $php_poll->display_poll(3);
  }
 echo "</TD></TR></TABLE>";

 // шаблон оформления
 $php_poll->set_template("poll_list");
 // формат даты
 $php_poll->set_date_format("m/d/Y");
 echo $php_poll->view_poll_list();
 echo $php_poll->get_list_pages();

 ?>
```

Наверное, вам не нравится надпись Advanced Poll и вам хотелось заменить ее на что-нибудь родное, например, My Poll. Откройте файл `include/config.inc.php` и найдите строку:

```
$pollvars['title'] = "Advanced Poll";
```

Вместо заголовка 'Advanced Poll' укажите нужный вам заголовок.

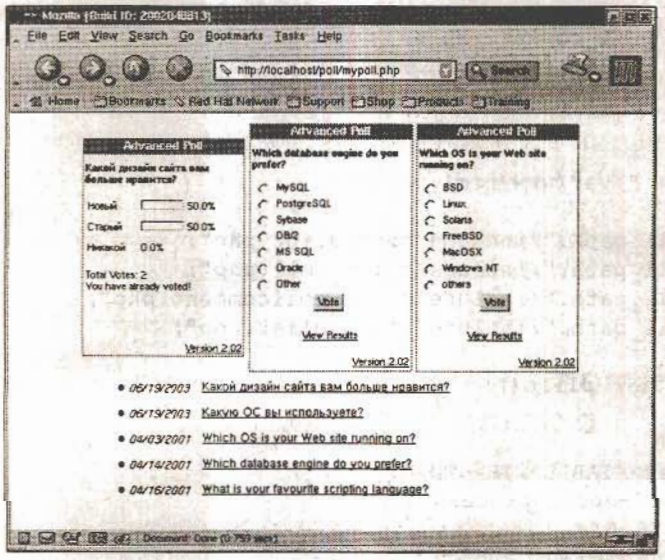


Рис. 17.10. *Вывод нескольких голосований на одной Web-странице*

Лента новостей

18.1. Принцип организации новостной ленты

Информационные сайты, порталы просто не могут обойтись без новостной ленты. Примеры новостных лент вы можете увидеть на многих информационных сайтах: `www.computerra.ru`, `www.linux.ru`, `www.linuxsrp.ru`, `dkws.narod.ru`. Обычно на главной странице отображается содержание новости «в двух словах» и «прилагается» ссылочка на страницу, где можно прочитать об этом подробнее, если эта новость вас заинтересовала.

Что же такое новостная лента? Это своеобразный симбиоз форума и системы голосования. Отличие от форума заключается в том, что новое сообщение (то есть новость) может добавить не кто угодно, а только администратор или лицо, отвечающее за наполнение странички с новостями. Обыкновенным пользователям (даже если они являются зарегистрированными пользователями сайта) разрешается только оставлять свои комментарии к новостям. Фактически, любой форум можно «трансформировать» в новостную ленту, разрешив создавать новые темы только модераторам.

Кроме функций форума новостная лента выполняет функции системы голосования, так как обычно предоставляется возможность оценить материал.

Писать собственными руками новостную ленту мы не будем — зачем заново изобретать велосипед. Если же вас не устраивает готовая лента новостей, можно отредактировать «под себя» готовый продукт, а если и этот вариант вас не устраивает, тогда вам остается только одно — запастись терпением и писать систему самому.

18.2. Используем профессиональный скрипт (FusionPHP)

18.2.1. Устанавливаем скрипт

Если вы создаете информационный сайт, портал или же ваш сайт часто обновляется и вы хотите сообщить пользователям, что же появилось нового, вам не обойтись без ленты новостей. Как утверждает команда

разработчиков FusionPHP, система Fusion News Management соответствует последним стандартам в области такого рода систем. Так почему же не использовать лучшее?

Система Fusion News Management предельно проста в эксплуатации, но, как оказалось, сложна в установке. Главным образом, сложность установки заключается в непродуманности сценария установки, предоставленного разработчиками системы. В принципе, ничего сложного в установке нет, но поработать руками (и головой!) все же придется. Систему можно взять на сайте `http://www.fusionphp.net/`.

Итак, приступим к установке системы. Для начала распакуем архив `fusionnews361.zip` в каталог `/var/www/html/news`. В этом каталоге появятся два подкаталога — images и upload. Как сказано в руководстве по установке, нужно установить права 777 для каталога `upload` и запустить сценарий `installer.php`, находящийся в каталоге `upload`. Скажу сразу, изменять права доступа ПОКА не нужно — вы еще успеете это сделать, а сейчас запустите `installer.php` (`http://localhost/news/upload/installer.php`).

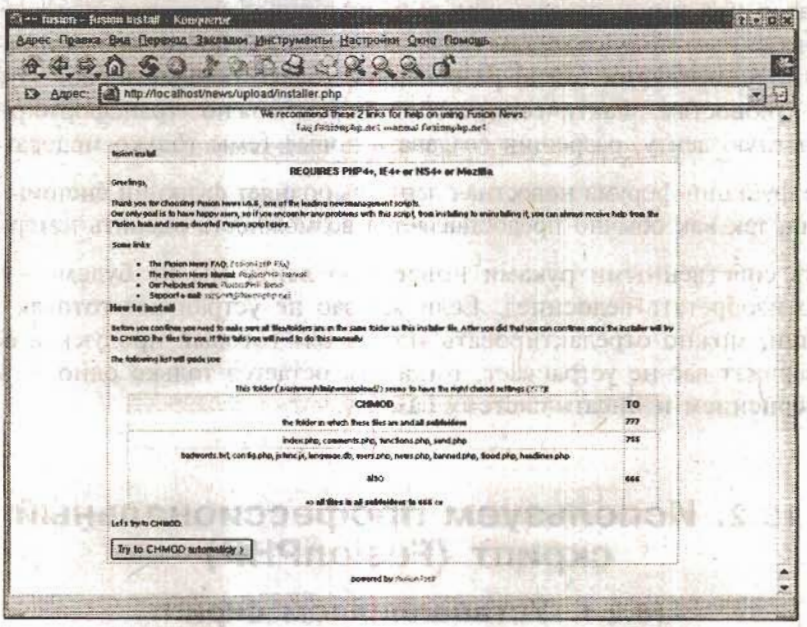


Рис. 18.1. *Первый шаг инсталлятора*

На первом этапе установки (см. рис. 18.1) сценарий предлагает нам изменить права доступа к подкаталогам каталога `upload` и файлам системы следующим образом:

- Каталог `upload` – 777.
- Подкаталога каталога `upload` – 777.
- Все файлы — 666.
- Файлы `index.php`, `send.php`, `functions.php`, `comments.php` – 755.

Примечание.

Напомню, права доступа 777 для каталога разрешают чтение и запись информации из/в каталог. Права доступа 666 для файла разрешают его чтение и запись, но запрещают выполнение. А права доступа 755 разрешают любые операции с файлом (в том числе и выполнение) владельцу файла, а также чтение и запуск файла (но не изменение!) другими пользователями. Честно говоря, сам не пойму зачем нужно устанавливать право на выполнение для PHP-сценария, но без этого сценарий установки отказывается продолжить саму установку, поэтому нужно выполнить его требования.

Попробуйте нажать на кнопку «Try to CHMOD automatically». Если ваша система настроена неправильно, с точки зрения безопасности, то сценарий сам изменит права доступа должным образом. Скорее всего, у вас ничего не выйдет, то есть вы не сможете установить систему автоматически, поскольку у вас нет полномочий для изменения прав доступа файлов, и вы увидите приблизительно то, что показано на рис. 18.2, — сообщение о необходимости изменения прав доступа.

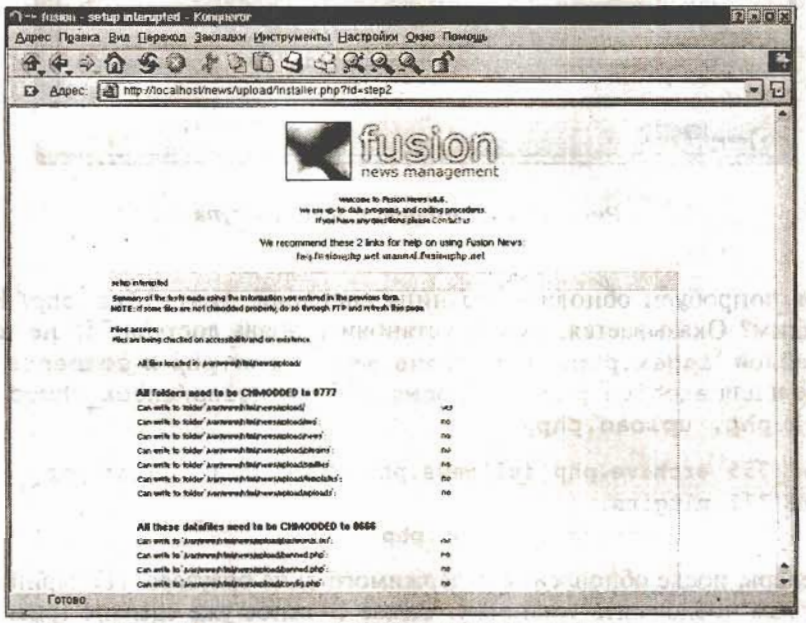


Рис. 18.2. Неправильные права доступа – установка прервана

Будем изменять права доступа вручную, в соответствии с указанной выше последовательностью действий. Должен сказать, что указанная последовательность неправильна. Мы будем все делать с точностью до наоборот. Сначала мы изменим права доступа для всех файлом и каталогов каталога upload:

```
chmod -R 666 upload
```

Затем изменим права доступа (755) для файлов index.php, functions.php, send.php и comments.php. После этого установим права доступа 777 для всех подкаталогов каталога upload и для него самого. Правильная последовательность действий указана на рис. 18.3.

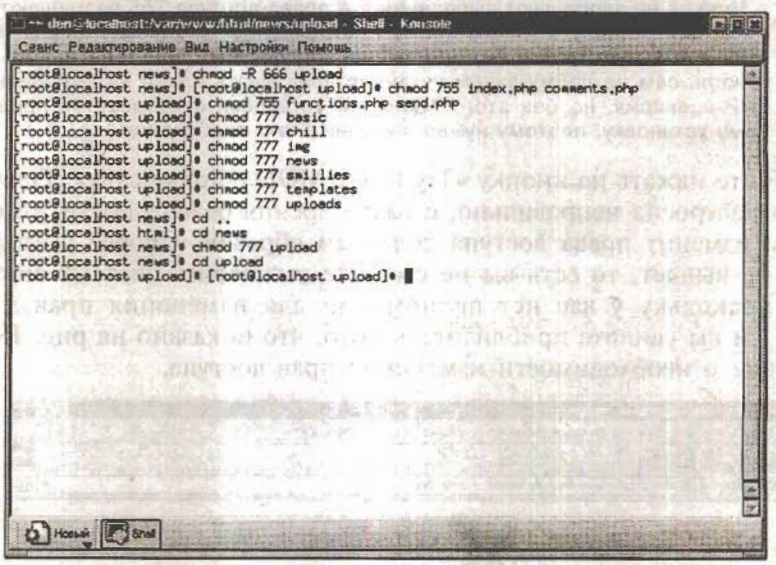


Рис. 18.3. *Изменение прав доступа*

Теперь попробуем обновить страницу сценария installer.php. И что мы видим? Оказывается, нужно установить права доступа 755 не только для файлов index.php, functions.php, send.php и comments.php, но еще и для archive.php, fullnews.php, plugins/check_chmod.php, search.php, upload.php:

```
chmod 755 archive.php fullnews.php search.php upload.php
chmod 777 plugins
chmod 755 plugins/check_chmod.php
```

Вот теперь, после обновления содержимого окна браузера, сценарий предложит вам продолжить установку. Самое сложное уже сделано (рис. 18.4)!

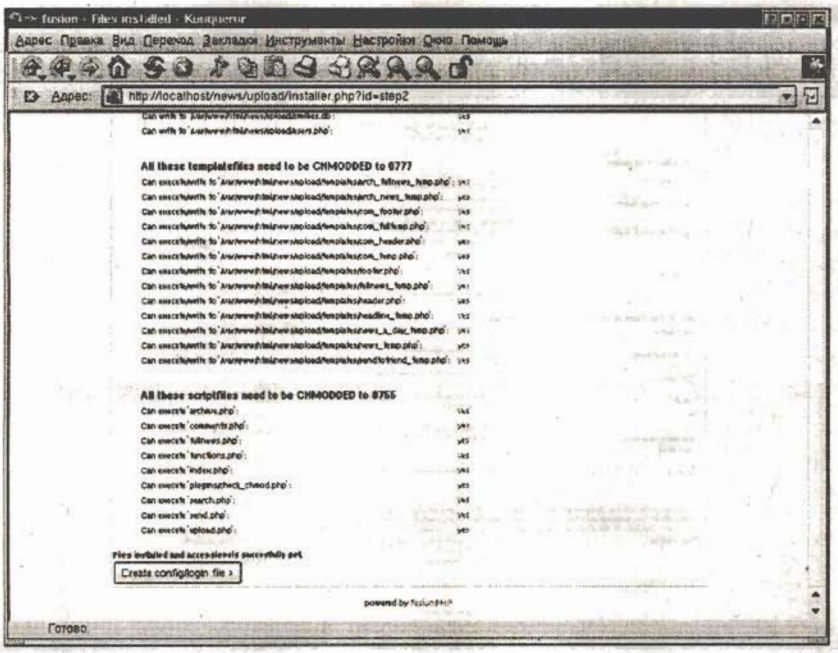


Рис. 18.4. *Права доступа установлены правильно*

Сейчас вам нужно ввести следующую информацию (рис. 18.5):

- URL вашего сайта, например, `http://www.my_site.ru`.
- URL к каталогу Fusion News. В нашем случае, это `http://www.my_site.ru/news/upload/`.
- Путь к каталогу Fusion News. У нас это `/var/www/html/news/upload/`.
- Логин администратора, например, `admin`.
- Пароль и его подтверждение. Осторожно! Пароль восстановить нельзя.
- Ник администратора.
- E-mail администратора.

Рекомендую использовать ник, отличный от логина администратора, а также установить режим **Hide email**, чтобы никто не узнал ваш адрес.

Последнее, что нужно сделать, так это выбрать режим работы системы — с использованием Сессий или Cookies. Рекомендую использовать сессии, так как они более практичны. В этой книге мы еще поговорим о разнице между сессиями и Cookies.

Вот теперь установка завершена (рис. 18.6). Не забудьте удалить файл `installer.php`, чтобы злобные хакеры не взломали вашу систему путем изменения пароля администратора.

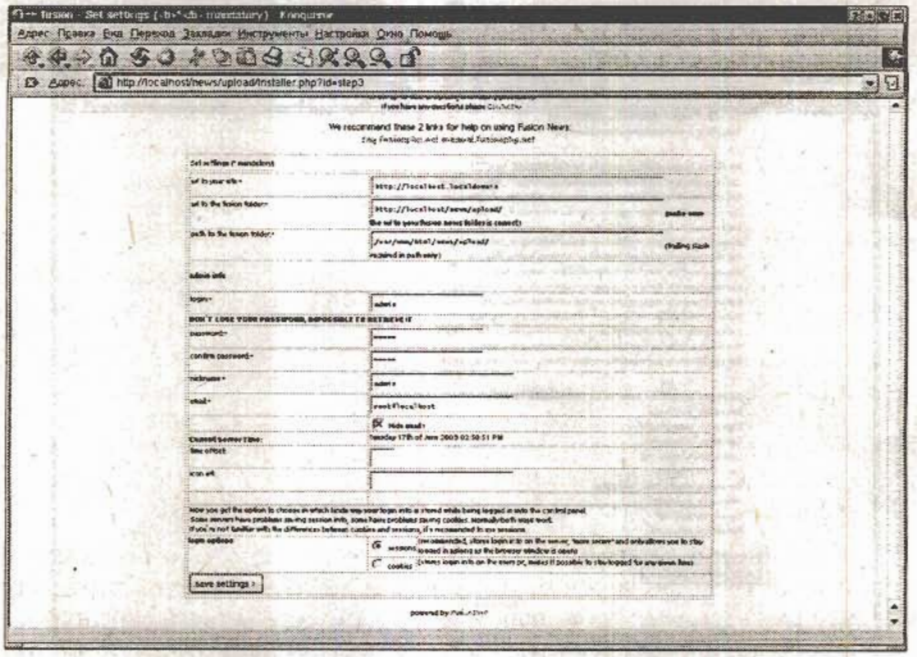


Рис. 18.5. *Ввод параметров системы*

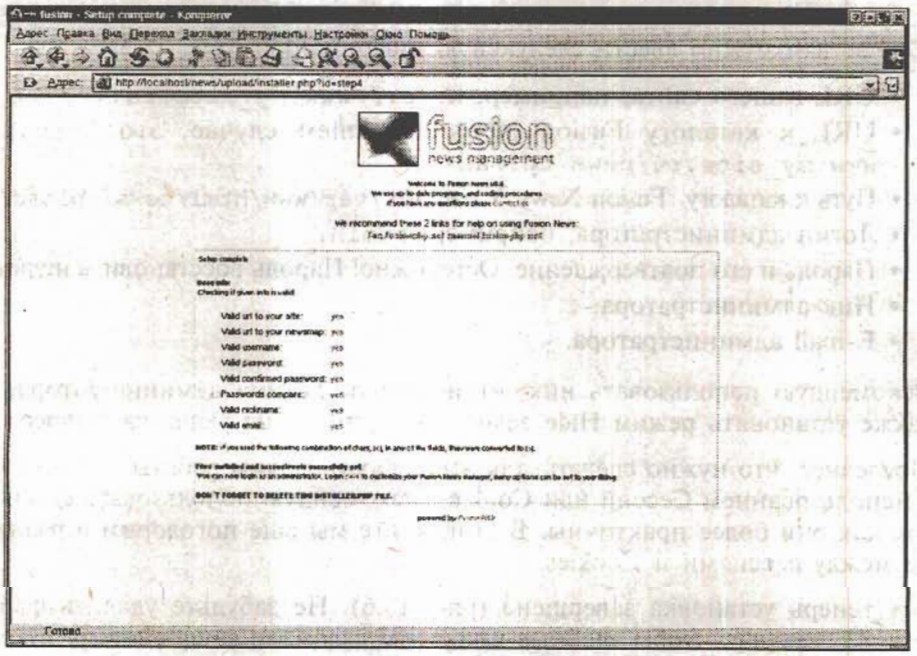


Рис. 18.6. *Установка завершена*

18.2.2. Работа со скриптом

Как я же писал, работать с системой чрезвычайно просто. В верхней части окна вы увидите восемь ссылок:

- main
- post news
- edit posts
- view news
- my webpage
- plugins
- help/update
- login

Для просмотра новостей нужно перейти по ссылке **view news**. Для добавления или редактирования новостей вам нужно сперва зарегистрироваться, а только потом щелкнуть по ссылке **post news** или **edit posts**.

Что вам еще нужно знать о системе? Во-первых, ее можно перевести на русский язык (или на любой другой) — для этого достаточно отредактировать файл `language.db` в любом текстовом редакторе и вручную перевести сообщения, выдаваемые системой, на русский язык. Во-вторых, все параметры, указанные вами при установке, находятся в файле `config.php`. У меня он выглядит так:

```
<?php
$fusion_id = "6LO9UY4844pN43fZb0vJ44fYF21WTcl9";
$cookies = "";
$site = "http://localhost.localdomain";
$furl= "http://localhost/news/upload/";
$hurl = "http://localhost.localdomain";
$fpath = "/var/www/html/news/upload/";
$datefor = "m-d-Y H:i";
$numofposts = "10";
$numofh = "2";
$bb = "checked";
$ht = "";
$first = "no";
$post_per_day = "";
$wfpost = "checked";
$wfcom = "checked";

// Это - "скин". Параметры скина хранятся в одноименном
// каталоге (upload/chill)

$skin = "chill";
```

```
$cbwordwrap = "";
$wwwidth = "80";
$smilies = "checked";
$stfpop = "";
$comallowbr = "checked";
$stfwidth = "";
$stfheight = "";
$fslink = "full story";
$stflink = "send to friend";
$pclink = "comments";
$fsnw = "";
$cbflood = "";
$floodtime = "30";
$comlength = "300";
$fullnewsw = "";
$fullnewsh = "";
$fullnewss = "";
$stfresize = "";
$stfscrolls = "";
$fullnewsz = "";
$htc = "";
$smilcom = "checked";
$bbc= "";
$compop= "";
$comscrolls= "";
$comresize= "";
$comheight= "";
$comwidth= "";
$uploads_active= "checked";
$uploads_size= "1048576";
$uploads_ext= "gif&#124;jpg&#124;jpeg&#124;png";
$enable_rss = "checked";
$link_headline_fullstory = "";
$flip_news = "";
?>
```

Наверно, вам будет интересно, где же хранятся сами новости. Они находятся в каталоге `upload/news`, в файлах `toc.php` (table of contents) и `news.X.php`. При желании, можно редактировать эти файлы вручную — формат их предельно прост.

Пример использования новостной ленты Fusion News показан на рис. 18.7.

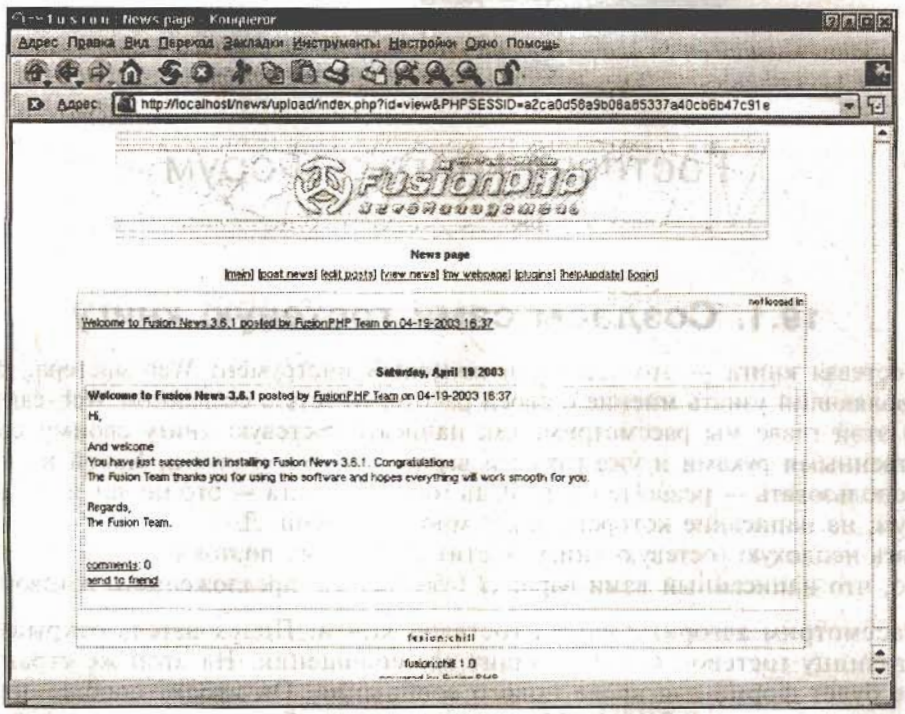


Рис. 18.7. *Пример использования новостной ленты Fusion News*

Глава 19

Гостевая книга и Форум

19.1. Создаем сами гостевую книгу

Гостевая книга — это еще один полезный инструмент Web-мастера, позволяющий узнать мнение о своей работе, то есть о созданном Web-сайте. В этой главе мы рассмотрим, как написать гостевую книгу своими собственными руками и уже готовый вариант гостевой книги. Какой из них использовать — решайте сами. Ведь гостевая книга — это не чат и не форум, на написание которого уйдет много времени. Для того, чтобы написать неплохую гостевую книгу, хватит 2...3 часов, поэтому вполне вероятно, что написанный вами вариант будет лучше предложенного готового.

Рассмотрим алгоритм нашей гостевой книги. Пользователь открывает страницу гостевой книги и видит все сообщения. На этой же странице будет форма для ввода нового сообщения. Он вводит сообщение и нажимает кнопку Submit. Мы записываем сообщение в начало файла, чтобы пользователи видели самые новые сообщения.

В файл гостевой книги будем записывать следующую информацию:

- Дату и время отправки сообщения.
- E-mail пользователя.
- Само сообщение.

Перед записью сообщения нужно удалить из него все HTML-теги. Это не только позволит немного сэкономить дисковое пространство, но и увеличит безопасность нашей гостевой книги. Формат файла гостевой книги будет следующий:

```
[Дата E-mail] Сообщение
```

Можно также записывать IP-адрес пользователя, но лучше этого не делать — гостевая книга вещь-то анонимная. Если вам все-таки интересен IP-адрес, то вы можете найти его в протоколах Web-сервера (по дате и времени) или же записывать его в файл, но не отображать в адресной книге. Данная манипуляция несколько усложнит наш сценарий, поэтому мы не будет записывать *IP-адреса в файл гостевой книги.*

Относительно записи E-mail могу сказать следующее: не записывайте и не выводите E-mail как ссылку, то есть в виде тега <a>: <a

href="mailto:user@email.ru">. Существуют программы, которые просматривают содержимое Web-страниц, находят теги <a>, содержащие электронные адреса, и вносят их в свои базы, которые потом будут проданы спамерам. Вы же не хотите, чтобы пользователям, оставившим сообщения в вашей гостевой книге, приходил спам?

Выводить сообщения будем с помощью функции `view()`. Ее исходный код представлен в листиге 19.1.

Листинг 19.1. Функция view()

```
function view()
{
$Messages=file('./gbook/gb.txt');
echo "<p><table width=100%>"; $i=0;

foreach($Messages as $v)
{
$i++;
if ($i % 2 == 0) echo "<tr><td>$v</td></tr>";
else echo "<tr><td bgcolor=gray>$v</td></tr>";
if ($i==$Max) break;
}
}
```

Сначала получаем весь файл `gb.txt` в массив `$Messages`. Потом обрабатываем каждый элемент этого массива — просто его выводим. Обратите внимание, если номер сообщения кратен двум, то фон ячейки (все сообщения выводятся как таблица, состоящая из одного столбца) станет серым. Если количество сообщений превысит лимит `$Max`, мы не станем выводить оставшиеся сообщения. Почему именной такой подход? Гостевая книга — это не чат и не форум. Согласитесь, если в вашей гостевой книге пользователи оставят 2-3 сообщения в день, то это уже хорошо (конечно, это при условии, что на сайте есть чат или форум — в противном случае все сообщения окажутся в гостевой книге). Поэтому, если мы будем выводить только первые 50 сообщений, это, во-первых, не создаст дополнительной нагрузки на трафик, а во-вторых, пользователи не увидят старые сообщения.

Если вас такой подход не устраивает, перепишите функцию `view()`, организовав постраничный вывод. Сделать это можно следующим образом: во-первых, заменить цикл **forach** циклом с параметром **for**, более гибко работающим с массивом (списком), а во-вторых, запоминать в Cookies номер последнего сообщения (`$last`), просмотренного пользователем. Вывод будет выглядеть следующим образом:

```
for($i=last; $i<count($Messages); $i++) echo
"<tr><td>$Messages[$i]</td></tr>";
```

Вот теперь можно приступить к написанию сценария gbook.php. Код сценария выглядит следующим образом:

Листинг 19.2. Код сценария гостевой книги

```
<?
echo "<html><head><title>Гостевая книга</title></head><body>";
echo "<h1>Гостевая книга</h1>";

$file_gb="./gbook/gb.txt";  // это файл гостевой книги
$file_tmp="./gbook/gb_tmp.txt";        // это временный файл,
// используемый сценарием
$Max=50;        // максимальное число выводимых сообщений

// Функция для вывода файла gb.txt
function view()
{
$Messages=file('./gbook/gb.txt');
echo "<p><table width=100%>"; $i=0;

foreach($Messages as $v)
{
$i++;
if ($i % 2 == 0) echo "<tr><td>$v</td></tr>";
else echo "<tr><td bgcolor=gray>$v</td></tr>";

// Если номер сообщения = 50 (то есть уже выведено 50
// сообщений), прерываем цикл
if ($i==$Max) break;
}
}

if (!isset($Post)) {

// Выводим форму для ввода нового сообщения
echo "<table width=100% border=1>";
echo "<tr><td><h2>Новое сообщение</h2><form method=post
action=$SCRIPT_NAME>";
echo " Email: <input type=text name=email>";
echo " Сообщение <input type=text name=mes> <input type=submit
name=Post></td></tr>";
echo "</table></form>";

// Выводим сообщения
view();

}
else
```

```
{
if(file_exists($file_tmp)) die("fatal error, call
administrator!");

// Добавляем новое сообщение в начало файла
if(copy($file_gb, $file_tmp))
{
 if($w=fopen($file_gb,"w"))
 {
  flock($w,2);   // блокируем доступ к основному файлу

  // $mes - это сообщение пользователя
  // перед внесением сообщения в файл удалим из него HTML-теги
 // strip_tags($mes);

  fwrite($w,"[".date("d.m.Y H:i:s").$email."] ".
strip_tags($mes)."\n");

  if(!$r=fopen($file_tmp,"r")) die("can't open file");
  flock($r,1);   // блокируем доступ к временному файлу

  while($mes=fgets($r,10240))
  {
   fputs($w,$mes);
  }

  // Снимаем блокировку и закрываем файлы
  flock($r,3);
  fclose($r);
  flock($w,3);
  fclose($w);

  // Удаляем временный файл
  unlink($file_tmp);
 }
}
echo "Сообщение успешно записано в файл";
// Выводим гостевую книгу
view();
}
?>
```

Сохраните этот сценарий под именем `gbook.php` в каталоге **$DocumentRoot** (по умолчанию, это `/var/www/html` — в Linux, и `c:\www\html` — в Windows). Теперь создайте каталог `gbook` в каталоге **$DocumentRoot**:

```
cd /var/www/html
mkdir gbook
```

Измените права доступа, позволяющие всем пользователям читать каталог, записывать в каталог файлы и просматривать его содержимое (там все равно будут только два файла):

```
chmod 777 gbook
```

В принципе, право просмотра содержимого каталога (его оглавления) совсем не обязательно — ну зачем пользователю знать, что в нем находится файл `gb.txt`? Поэтому прав 666 вполне хватит:

```
chmod 666 gbook
```

После этого нужно создать файл `gb.txt` и установить для него права доступа 666 (разрешить чтение, запись, но запретить выполнение):

```
cd /var/www/html/gbook
touch gb.txt
chmod 666 gb.txt
```

Примечание.

Если вы программируете под Windows 9x/ME, никакие манипуляции по изменению прав доступа производить не нужно.

Вот теперь можно запустить наш сценарий (см. рис. 19.1 и 19.2).

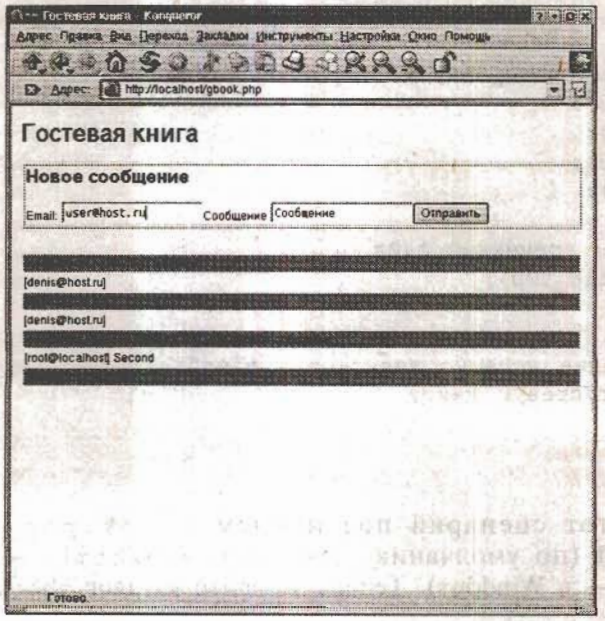


Рис. 19.1. *Просмотр гостевой книги*

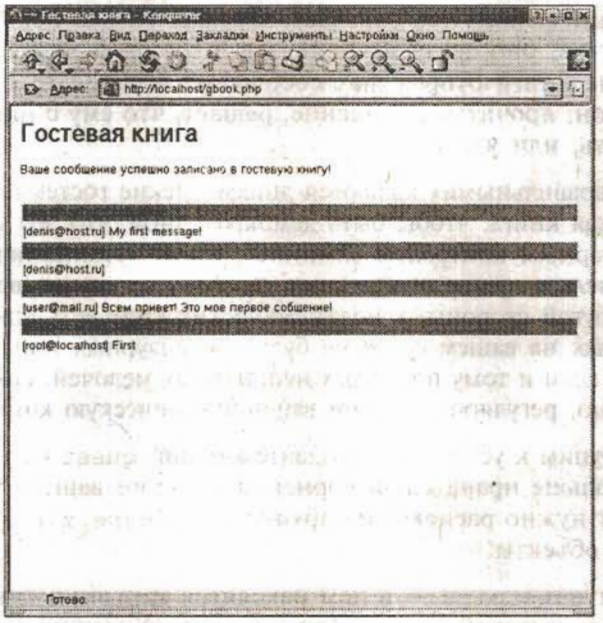


Рис. 19.2. *Сообщение добавлено*

Примечание.
Файл `gbook.php` доступен на моем сайте – `http://dkws.narod.ru/php`.

19.2. Используем готовый скрипт гостевой книги

19.2.1. Установка гостевой книги

Что такое «гостевая книга» — говорить, наверное, не нужно. Это своеобразная электронная книга записей, в которой пользователь может оставить свое сообщение на определенную тему и прочитать сообщения других. В предыдущем пункте мы с вами немножко научились тому, как можно сделать простенькую гостевую книгу. В этом и следующем пунктах мы с вами рассмотрим использование готовых скриптов. Сначала поработаем с относительно несложным скриптом, а затем, в следующем пункте (19.3), изучим, как «прикрутить» к сайту и использовать профессионалный форум PHPBB.

А пока начнем с малого собрата PHPBB — гостевой книги **GuestsHere**, доступную на сайте `http://www.pyroxpro.com/phpapps/index.php?download=GuestsHere.zip`. Гостевая книга **GuestsHere** яв-

ляется динамической. Это означает, что, как только пользователь отправит сообщение, оно сразу же появится в гостевой книге. Нединамические гостевые книги отправляют сообщение пользователя администратору сайта, а он, прочитав сообщение, решает, что ему с ним делать — или опубликовать, или удалить.

Наиболее «правильными» являются динамические гостевые книги — на то она и гостевая книга, чтобы быть демократичной: что это за «глас народа» под тоталитарным контролем администратора? Нединамические гостевые книги как нельзя лучше отображают «свободу слова» во времена бывшего Союза. С другой стороны, у нединамических книг есть одно неоспоримое преимущество: на вашем сайте не будет нецензурных выражений, нежелательной критики и тому подобных неприятных мелочей. Поэтому, если для вас это важно, регулярно читайте вашу динамическую книгу.

Итак, приступим к установке. Создайте каталог `guest` (или `guestbook` — как кому больше нравится) в корневом каталоге вашего Web-сервера. В этот каталог нужно распаковать архив `GuestsHere.zip`. Архив содержит следующие объекты:

- Каталог `guestpics` — в нем находятся картинки графического интерфейса гостевой книги. При желании их можно заменить своими собственными картинками.
- Файл `guest.php` — выводит форму для ввода нового сообщения адресной книги.
- Файл `guestlist.php` — выполняет непосредственную запись введенного пользователем сообщения в файл `guestshere.php`.
- Файл `guestshere.php` — самый обыкновенный HTML-файл, содержащий введенные пользователями сообщения, обрамленные в симпатичные рамки.
- Файл `loguests` — это протокол адресной книги.

После распаковки архива нужно правильно установить права доступа. Установите режим доступа, разрешающий запись для каталогов `guest` и `guestpics`:

```
chmod 777 guest
chmod 777 guest/guestpics
```

Если вы не можете ввести команду **chmod**, для изменения прав доступа используйте средство управления контентом, например, MonsterControls (а об этом я дальше буду рассказывать в. гл. 29). Также нужно разрешить запись в файл `guestshere.php`:

```
chmod 666 guestshere.php
```

В противном случае, вы получите сообщение о невозможности записи либо в каталог guets, или в файл `guestshere.php`.

Все! Ваша гостевая книга установлена и готова к работе. Забегая наперед, скажу, что у этой адресной книги два недостатка: она не русифицирована и нет средства, позволяющего удалить ненужное сообщение администратору. Однако, несмотря на то, что книга нерусифицирована, это никак не мешает оставлять русскоязычные сообщения. Избавиться от этих небольших недостатков достаточно просто: вам же никто не мешает открыть PHP-файл в любом текстовом редакторе и перевести все выводимые сообщения на русский язык? Тем более, что этих самых сообщений не так уж и много.

Со вторым недостатком тоже относительно легко можно справиться: можно просто удалять сообщения — вручную из файла `guestshere.php`, а также можно написать собственный сценарий, удаляющий ненужное вам сообщение.

К достоинствам гостевой книги можно отнести ее простоту: сколько времени вы потратили на ее установку? Минуты три-четыре. Если вам эта адресная книга покажется уж слишком простой, переходите к следующему пункту. В нем мы рассмотрим известный профессиональный скрипт форума PHPBB.

19.2.2. Работа с гостевой книгой

В строке браузера введите следующий URL: `http://localhost/guest/guest.php`. Вы увидите главную форму гостевой книги, позволяющую добавить новое сообщение.

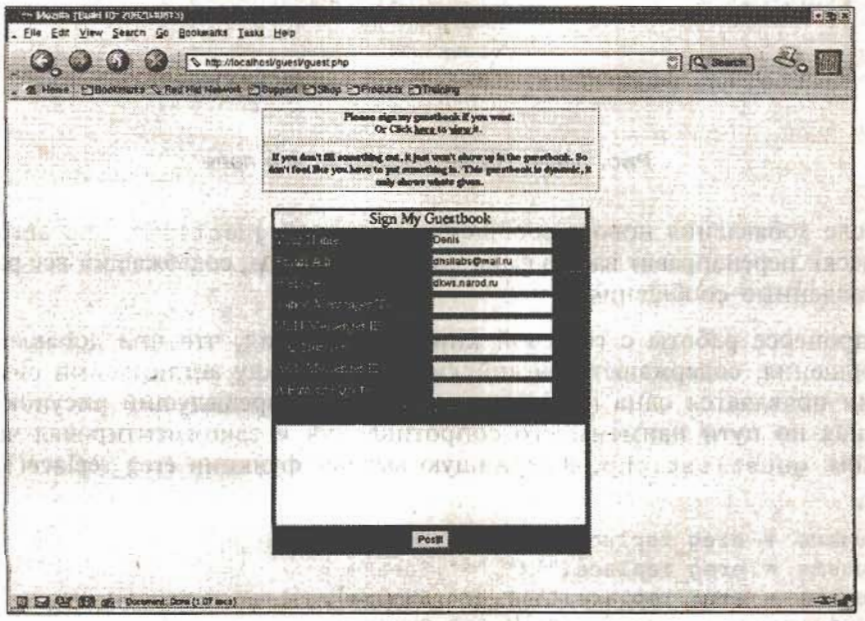


Рис. 19.3. Добавление нового отзыва

По ссылке «Click here to view it» можно просмотреть содержание гостевой книги. Заполните поля гостевой книги и нажмите кнопку «PostIt». Можно не заполнять все поля — в гостевой книге будут отображены только те поля, которые вы указали.

После нажатия кнопки «PostIt» вы увидите сообщение о том, что не все поля заполнены. При желании можно немного модифицировать файл `guestlist.php` так, чтобы это сообщение не выводилось.

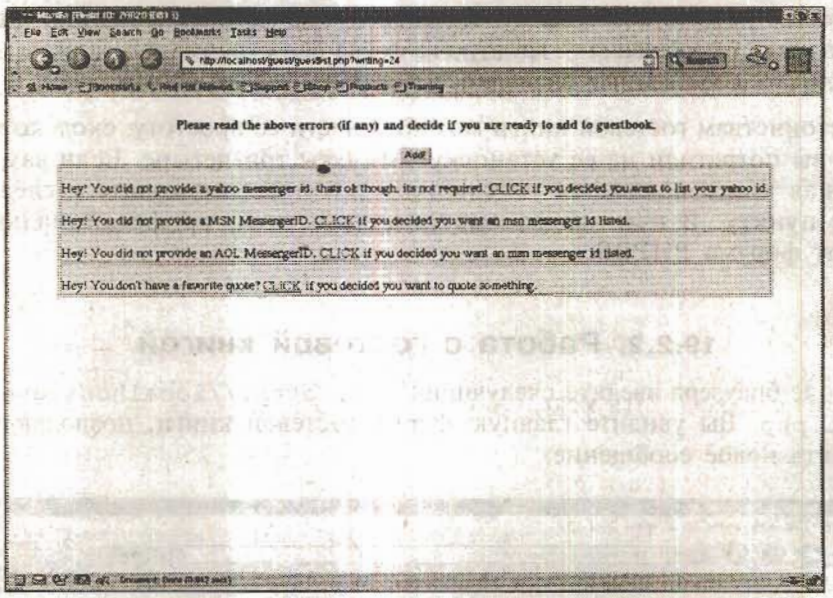


Рис. 19.4. *Вы не заполнили все поля*

После добавления нового сообщения сценарий `guestlist.php` автоматически перенаправит вас на файл `guestshere.php`, содержащий все ранее оставленные сообщения.

В процессе работы с гостевой книгой я заметил, что при добавлении сообщения, содержащего английский текст, между английскими символами появляется одна или две звездочки (см. предыдущий рисунок). Я пошел по пути наименьшего сопротивления и закомментировал часть файла `guestlist.php`, содержащую вызовы функции **ereg_replace()**:

```
/*
$uname = ereg_replace("\?","*",$uname);
$uname = ereg_replace("\(","*",$uname);
$uname = ereg_replace("\)","*",$uname);
$uname = ereg_replace("\<","*",$uname);
```

```
$uname = ereg_replace("\>","*",$uname);
$uname = ereg_replace("\{","*",$uname);
$uname = ereg_replace("\}","*",$uname);
$mail = ereg_replace("\?","*",$mail);
...
```

Если я правильно понял логику разработчика, вызовы **ereg_replace()** он использовал для удаления HTML-тегов, слэшей и другой потенциально опасной информации. Вместо многократных вызовов можно использовать одну из следующих функций:

- **strip_tags()** — удаляет HTML-теги;
- **HtmlSpecialChars()** — заменяет некоторые символы, например, больше или меньше, на их HTML-эквиваленты, то есть на **>** или **$lt**.

Например:

```
$uname = strip_tags($uname)
```

или

```
$uname = HtmlSpecialChars($uname)
```

Использовать регулярные выражения, как это сделал разработчик гостевой книги, вообще не рекомендуется, поскольку с их помощью вообще невозможно выделить некоторые теги.

После проведенного «оперативного вмешательства» ваша гостевая книга окончательно готова к работе.

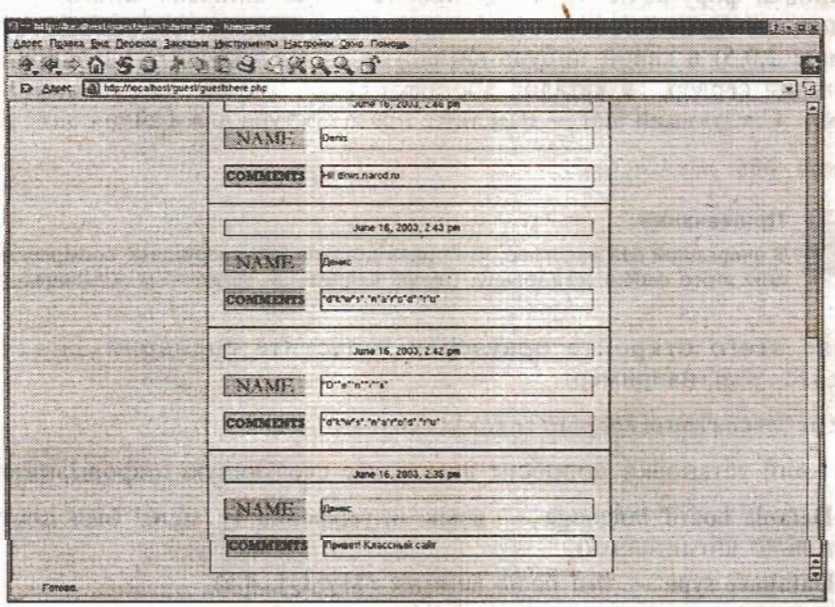


Рис. 19.5. Гостевая книга

19.3. Форум PHPBB

19.3.1. Установка форума PHPBB

В прошлом пункте мы рассмотрели простой форум, написанный Дмитрием Прониным. Если вы его уже установили и он вас устраивает, используйте его. Сейчас мы познакомимся с лучшим среди бесплатных форумов — форумом PHPBB.

Действительно, поработав с этим форумом, чувствуется веяние профессионализма разработчиков. Несмотря на то, что форум бесплатный, работает он не хуже любого коммерческого, например, **vbulletin** (www.vbulletin.com).

Форум PHPBB обеспечивает:

- Все функции форума.
- Ведение списка пользователей.
- Поддержку групп пользователей.
- Создание профилей пользователя, то есть каждый пользователь может определить свои собственные настройки.
- Поиск.
- Удобное администрирование форума.

Это только основные функции форума, со всеми остальными мы познакомимся в процессе работы с форумом.

Установка форума не вызывает проблем и не занимает много времени. Распакуйте архив `PHPbb-2.0.5.tar.gz` (не стоит устанавливать версию, младше 2.0.5) в какой-нибудь каталог, а затем закачайте распакованные файлы на сервер, в каталог `$DocumentRoot/phpbb` (`/var/www/html/phpbb`). Следующий шаг — измените права доступа для файла `config.php`:

```
chmod 666 config.php
```

Примечание.

В очередной раз напомню, что права доступа 666 разрешают всем модификацию этого файла (владельцу, группе владельца, остальным пользователям).

После этого откройте браузер и запустите сценарий `install/install.php`, например:

```
http://localhost/phpbb/install/install.php
```

Сценарий установки попросит ввести вас следующую информацию:

- **Default board language** — язык интерфейса (у меня был доступен только английский).
- **Database type** — тип базы данных (MySQL 4.x).
- **Instalation method** — метод установки — install (инсталляция).

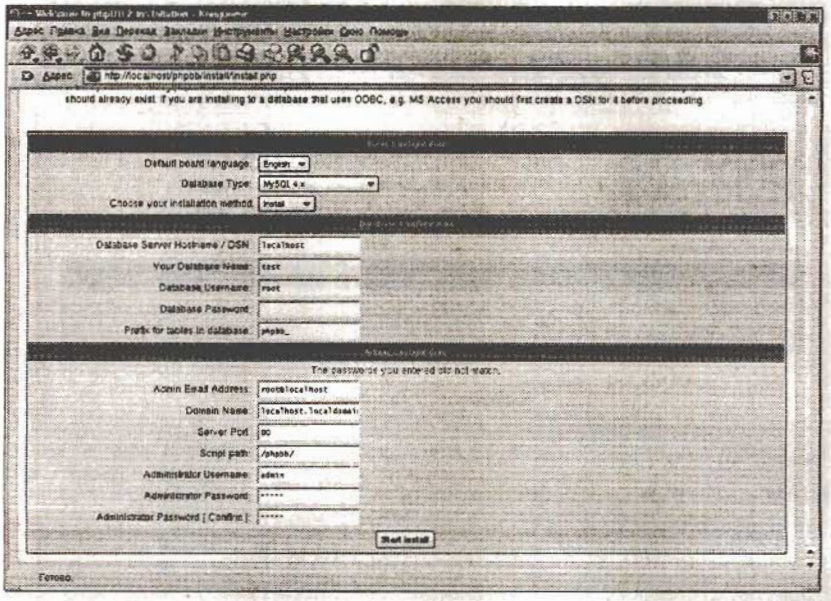


Рис. 19.6. *Установка форума phpBB*

- **Database Server Hostname** — имя сервера баз данных — обычно localhost.
- **Your database name** — имя базы данных (база данных уже должна существовать).
- **Database user** — имя пользователя созданной базы данных MySQL.
- **Database password** — пароль пользователя MySQL.
- **Prefix for tables in database** — префикс для таблиц форума в базе данных. Префикс нужно указать, если вы используете одну общую базу данных, чтобы имя таблиц различных программных продуктов не конфликтовали друг с другом.
- **Admin email** — адрес электронной почты администратора.
- **Domain name** — доменное имя.
- **Server port** — номер порта Web-сервера (обычно 80).
- **Script path** — путь к форуму от корневого каталога Web-сервера (от каталога `$DocumentRoot`). У нас это `/phpbb/`.
- Имя пользователя администратора и его пароль.

На этом установку можно считать завершенной. Осталось только удалить каталоги `install` и `contrib` (из соображений безопасности), иначе форум работать не будет.

После удаления этих каталогов измените права доступа к файлу `config.php` (теперь писать в этот файл имеете право только вы):

```
chmod 644 config.php
```

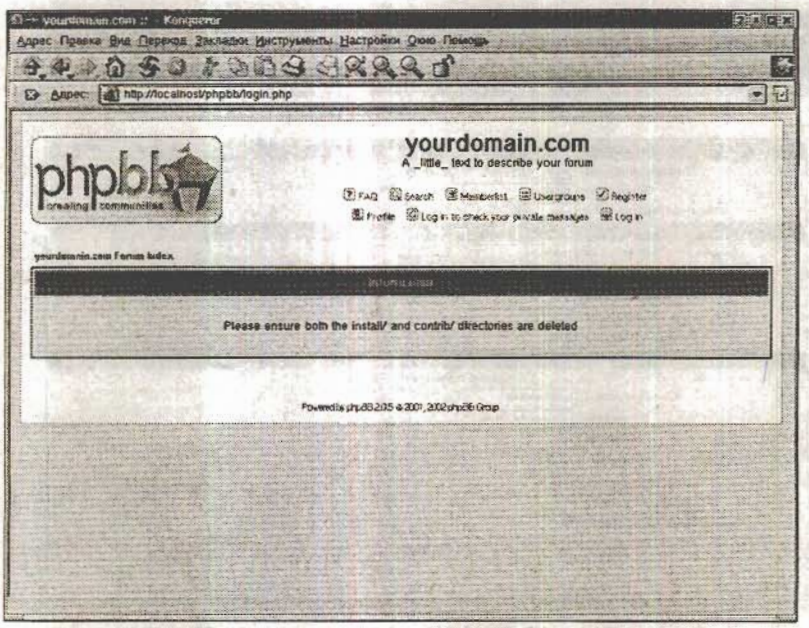


Рис. 19.7. *Удалите каталоги install и contrib*

Теперь русифицируем форум — как хорошо мы ни знаем английский, но русский язык созерцать на экране монитора гораздо приятнее. Зайдите на сайт `http://myphpbb.com.ru/` и скачайте файл `lang_russian.zip` (в разделе Русификация). Я рекомендую скачать первый файл — это самый удачный перевод форума.

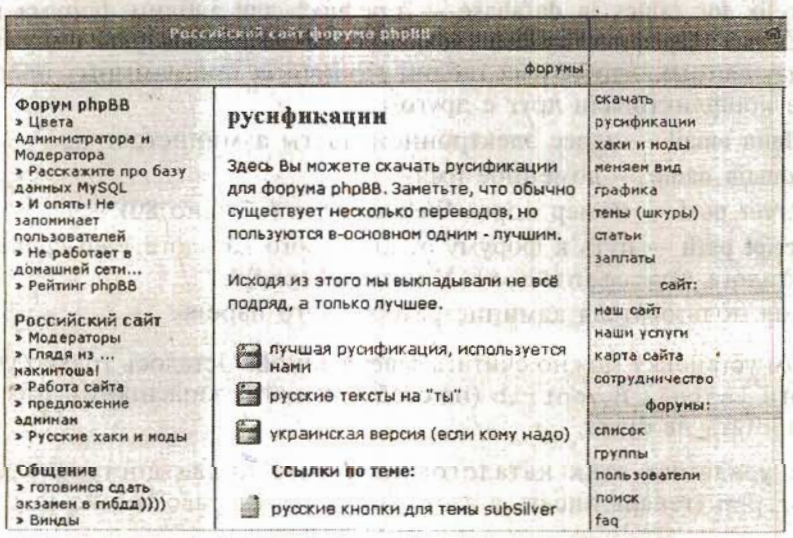


Рис. 19.8. *Русификация форума*

После того как архив закружен, распакуйте его в каталог `$DocumentRoot/phpbb/language/`. Все! Теперь вам в свойствах осталось только выбрать русский язык.

Работать с форумом просто — как обычно, нужно сперва зарегистрироваться (Register), затем войти (Log In), ну а дальше вы сами знаете, что делать.

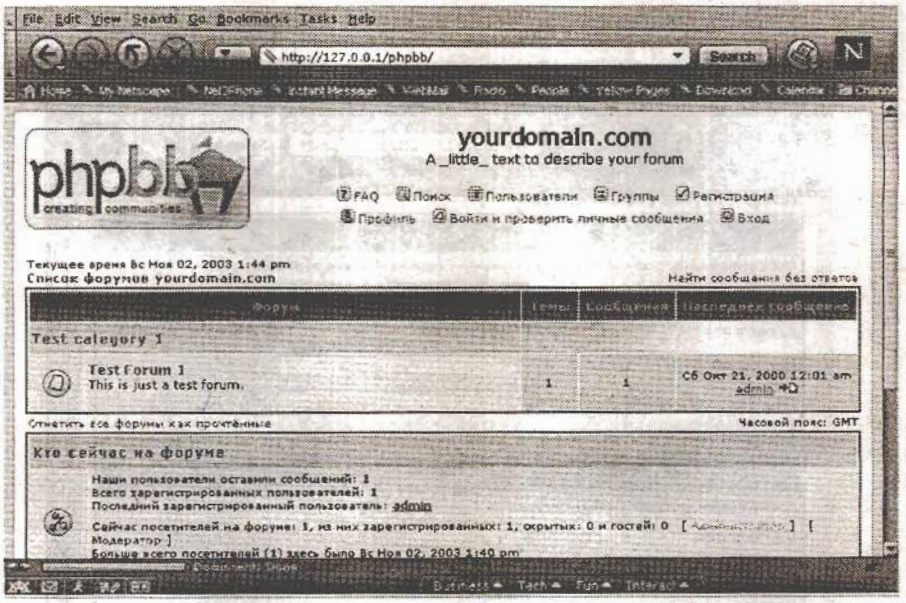


Рис. 19.9. *Форум в работе*

При регистрации нового пользователя форум запрашивает следующую информацию (ее объем впечатляет, что говорит о «серьезности» форума):

- Имя пользователя, пароль и его E-mail.
- Номер ICQ, адрес AIM, Yahoo ID, MSN ID.
- Домашнюю страничку.
- Подпись пользователя (можно включить теги HTML, BB, а также смайлики).
- Отображать ли адрес электронной почты.
- Скрывать ли состояние online.
- Уведомлять ли при ответе на сообщение в форуме.
- Уведомлять ли о личном сообщении.
- Открывать ли новое окно при получении личного сообщения.

- Всегда ли добавлять подпись.
- Разрешать ли использование тегов HTML, BB и смайликов.
- Язык форума (как я уже говорил, кроме английского в моем дистрибутиве ничего недоступно).
- Скин форума.
- Временную зону.
- Формат даты (синтаксис такой же, как и у функции date()).

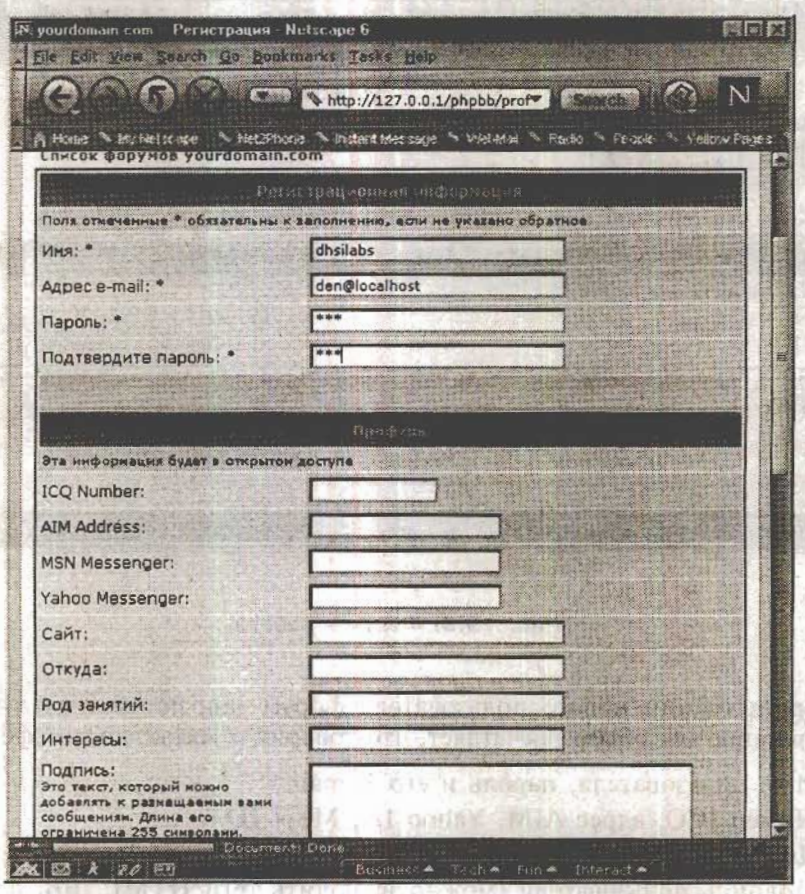


Рис. 19.10. Регистрация нового пользователя

19.3.2. Изменение внешнего вида форума

Изменить внешний вид форума можно, либо целиком сменив одно оформление на другое (один скин на другой), либо изменив лишь некоторые отдельные элементы. Первый вариант (полная смена оформления) лучше всего осуществлять с центра управления форумом. Об этом читайте ниже, в разделе «Администрирование форума».

Заменить отдельные же элементы можно вручную, заменив нужные графические файлы:

- Логотип форума — `форум/templates/SubSilver/images/logo_phpBB.gif`.
- Верх и низ форума меняются в следующих файлах:

  ```
  форум/templates/SubSilver/overall_header.tpl
  форум/templates/SubSilver/overall_footer.tpl
  ```

  Файлы `overall_header.tpl` и `overall_footer.tpl` являются файлами-темплейтами (.tpl) и отвечают за построение верхушки и низа каждой страницы форума. Код из этих файлов вставляется в каждую страницу. Настройку данных файлов можно осуществлять либо с центра управления (путем изменения соответствующих параметров оформления), либо вручную.
- Фоны-заливки:

  ```
  форум/templates/SubSilver/images/cellpic1.gif
  форум/templates/SubSilver/images/cellpic2.jpg
  форум/templates/SubSilver/images/cellpic3.gif
  форум/templates/SubSilver/images/cellpic.gif
  ```

19.3.3. Администрирование форума

Теперь поговорим об администрировании форума. Форум phpBB, подобно его коммерческим собратьям, обладает собственным центром управления. Вызвать центр управления можно по адресу:

```
http://имя_узла/phpbb/admin/index.php
```

Центр управления позволяет:

- Просмотреть статистику форума.
- Просмотреть список подключенных к форуму пользователей.
- Создавать новые форумы.
- Создавать различные разделы форума для обсуждения тех или иных вопросов.
- Установить уровень доступа к форуму.

- Создать резервную копию базы данных, а также восстановить заархивированную базу данных.
- Изменить общие параметры форума.
- Создавать стили форума.
- Организовать массовую рассылку почты зарегистрированным пользователям.
- Создать новую группу.
- Управлять учетными записями пользователей, назначать модератора той или иной группы.

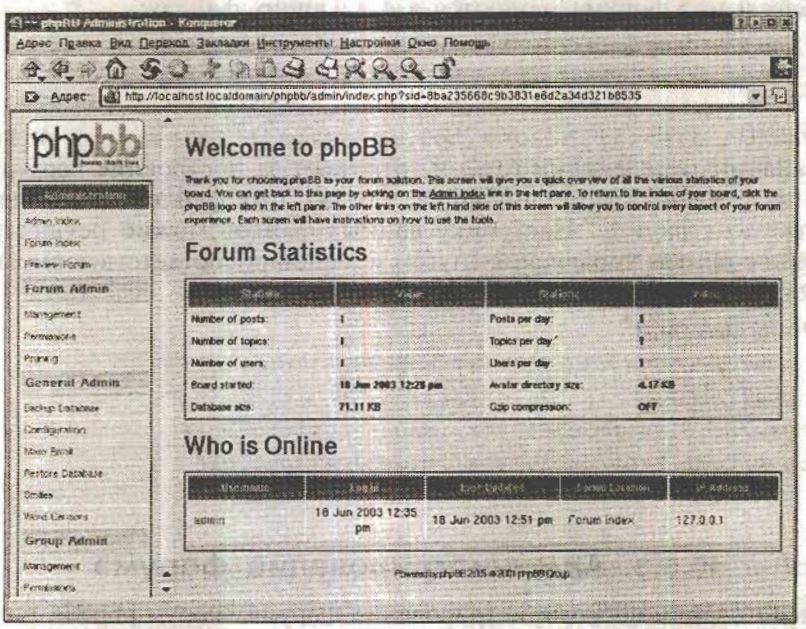


Рис. 19.11. *Центр управления*

Первым делом зайдите в **General Admin/Configuration** и измените общие параметры форума. Лично меня устраивают все параметры по умолчанию. Единственное, что я изменил, так это описание сайта `A little text to describe your forum` не совсем отвечало моим требованиям.

Остановимся на самых интересных возможностях центра управления форумом. В разделе **Форумы, Управление** (Forums, Management) можно создать или удалить форум, тут же можно изменить название форума.

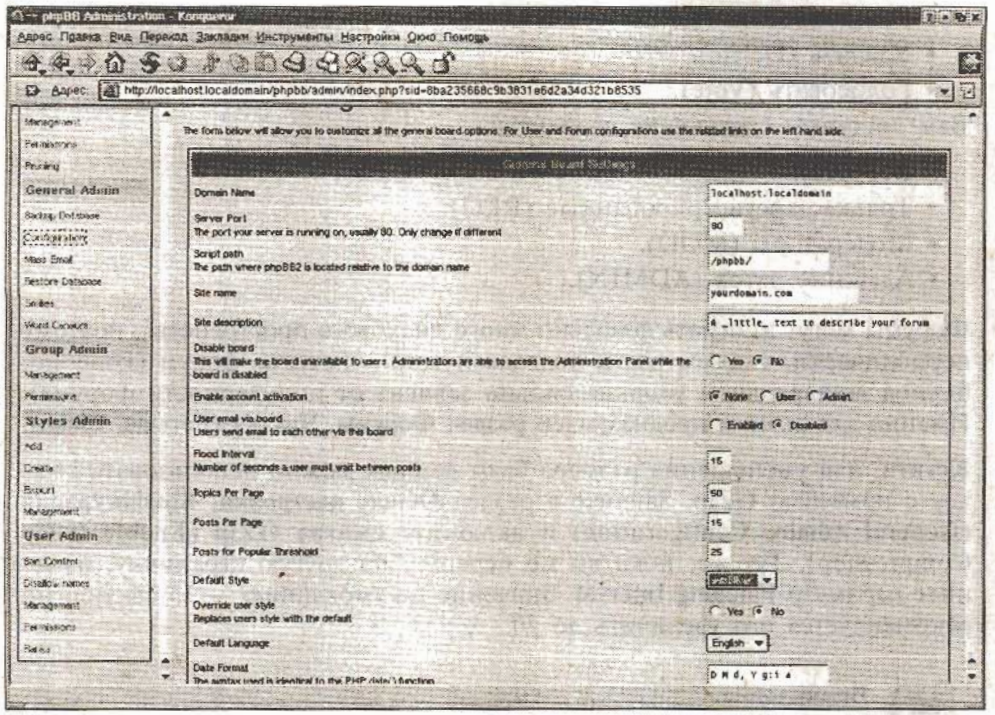


Рис. 19.12. *Общие параметры форума*

Раздел **Форумы, Права доступа** (Forums, Permissions) позволяет установить права доступа к конкретному форуму. Доступны следующие варианты:

- **Public** — все пользователи могут работать с форумом.
- **Registered** — форум только для зарегистрированных пользователей.
- **Registered [Hidden]** — этот форум будут «видеть» только зарегистрированные пользователи. Ясное дело, что кроме них, с этим форумом никто не будет работать.
- **Moderator** — форум только для модераторов.
- **Moderator [Hidden]** — скрытый форум — его будут видеть только модераторы.

В расширенном режиме (Advanced Mode) вы можете определить, кто имеет право:

- Просматривать сообщения (View).
- Читать сообщения (Read).
- Создавать новые сообщения (Post).
- Отвечать на сообщения (Reply).

- Редактировать (Edit).
- Удалять (Delete).
- Голосовать (Vote).

Вам доступны следующие варианты:

- Все (All).
- Только зарегистрированные (REG).
- Модераторы (MOD).
- Администратор (ADMIN).

Форумы могут занимать довольно много дискового пространства, поэтому периодически, скажем раз в полгода нужно «чистить» форум (конечно, период «генеральной уборки» сильно зависит от популярности форума). Именно для этого и предназначен раздел **Форумы, Чистка (Forums, Clean)**.

Кстати, для уменьшения размера базы данных можно использовать сжатие с помощью GZip. Зайдите в раздел **Общие настройки, Конфигурация (General Admin, Configuration)** и включите сжатие GZip (Enable GZip Compression). Тут же, пока вы не перешли на другую страничку, измените параметр **Flooding Interval**. Значение по умолчанию — 15 секунд, но рекомендуется его увеличить до 20.

Примечание.

Обычно интервал между сообщениями, поступившими от одного пользователя, составляет не менее 15 секунд. Если интервал меньше, это означает, что кто-то запустил специальную программу — "флудитель форума", которая будет "забивать" форум ненужными сообщениями. Параметр Flooding Interval позволяет хоть как-то защититься от этой напасти. Скажем, если интервал между сообщениями одного пользователя составляет 14 секунд (что меньше 15), эти сообщения будут удалены (точнее, вообще не будут добавляться в базу данных форума).

Вы надумали переезжать? В смысле, изменить хостинг-провайдера? Тогда выберите пункт **Общие настройки, Резервная копия БД (General Admin, Backup database)**. Вы можете создать резервную копию всей базы данных форума и записать на свой компьютер, чтобы при "переезде" на новый хостинг сохранить старые темы.

Не бойтесь экспериментировать с форумом — там все просто. Пока на форуме не зарегистрировалось много пользователей, и вы его официально не открыли, можно делать в центре управления все, что угодно — ведь всегда можно переустановить форум. А вот в процессе работы форума нужно быть предельно аккуратным — ведь если вы пару раз удалите какого-нибудь пользователя, вряд ли он захочет еще раз посетить ваш сайт. *И не забывайте* делать резервную копию базы данных!

Глава 20

Счетчик и анализ серверной статистики

20.1. Счетчик и протоколирование посещений на сайте

20.1.1. Простой счетчик

Разрабатываем механизм работы счетчика

Предлагаемое мною решение никак не претендует на универсальность, но наша цель сначала разработать простой, но «правильный» счетчик, то есть такой, который не реагировал бы на Reload (Обновление страницы). Позже также будет рассмотрен более сложный счетчик с протоколированием.

Сейчас попробуем разработать простой счетчик, который будет подсчитывать посещения только одного (нашего) сайта. Определим требования к нашему счетчику:

- Он должен подсчитывать только уникальных посетителей.
- Счетчик не должен реагировать на нажатие Reload в браузере.
- Счетчик не должен вести учет наших собственных посещений (с локального хоста).

Считать уникальных посетителей можно по-разному. Идентификация узлов в Internet осуществляется с помощью IP-адресов. Поэтому первое, что приходит в голову — записывать все уникальные IP-адреса в какой-нибудь файл (или базу данных). Но такой метод нам совершенно не подходит.

Во-первых, IP-адрес может быть динамический и пользователь может зайти на сайт во второй раз, но уже используя другой адрес. В этом случае не выполняется первое требование, предъявленное к нашему счетчику: уникальность посетителя. Во-вторых, в случае с файлом поиск в файле является довольно медленной операцией, не говоря уже об операции записи в файл с использованием блокировки, особенно учитывая, что к нашему сайту может в одно время обращаться несколько пользователей. К тому же рано или поздно, размер нашего файла станет неприлично большим. В случае с базой данной этот счетчик уже не будет простым.

Нам остается использовать только Cookies. Алгоритм работы счетчика в этом случае будет такой: когда пользователь заходит на ваш сайт, сценарий проверяет, установлены ли определенные (наши) Cookies. Если нет, то посетитель считается уникальным и счетчик увеличивается, в противном случае, возвращается предыдущее значение — то, которое было уже когда-то записано в файл. Естественно, после инкремента счетчика (увеличения на единицу) нужно установить Cookies, "пометив" этого пользователя, что он уже был на сайте.

Пишем сценарий счетчика

Счетчик действительно оправдывает свое название «простой»: всего 17 строк кода (см. листинг 20.1).

Листинг 20.1. Простой счетчик

```
<?
 $f=fopen("counter.dat","a+") or die("Cannot to open file");
 flock($f,2);
 $count=fread($f,100);
 if(!IsSet($Was))
  {
  $Was=1;
  SetCookie("Was",$Was,0x7FFFFFF);
  // Не засчитываем посещения со своего узла
  $REMOTE_HOST=gethostbyaddr($REMOTE_ADDR);
 if(!($HTTP_HOST===$REMOTE_HOST)) @$count=$count+1;
 ftruncate($f,0);
 fwrite($f,$count);
 }
 flock($f,3);
 fclose($f);
 echo $count;
?>
```

Теперь немного опишу принцип работы счетчика. В строке 2 открываем файл `counter.dat`. Причем используется режим **a+**. В этом режиме файл открывается, как для чтения, так и для записи. Указатель файла устанавливается на его конец. В отличие от режима «a», в режиме «a+», если файл не существует, он создается. Затем мы блокируем файл. Последний параметр функции **flock()** определяет вид блокировки. В данном случае — это режим 2 (LOCK_EX — исключительная блокировка): никто, кроме нас, не может производить с файлом никаких операций. Если еще какому-нибудь процессу нужно прочитать информацию из файла или записать что-

нибудь в него, он вынужден будет подождать, пока мы завершим работу с файлом. Возможно, при большой нагрузке на сервер, целесообразно использовать разделяемую блокировку, но пока нам это не нужно.

Затем мы читаем текущее значение из файла в переменную `$count`. Потом мы проверяем, установлен ли Cookie `$Was`. Для этого нам всего лишь нужно проверить существование переменной `$Was` с помощью функции **IsSet()**. Если Cookie `$Was` не установлен, мы его устанавливаем. Функция **SetCookie** устанавливает Cookie `$Was` и записывает в него значение одноименной переменной `$Was`. Третий параметр функции — это время существования Cookie.

Если значение, возвращенное функцией **gethostbyaddr($REMOTE_ADDR)**, совпадает со значением переменной окружения $HTTP_HOST, то сценарий запущен с локального хоста и мы не засчитываем это посещение. В принципе, проверку на запуск с локального хоста можно было бы и не производить, так как все равно устанавливаются Cookies и вы не увеличите значение счетчика более, чем на единицу.

В строке 12 (`fwrite($f,$count);`) мы записываем новое значение переменной `$count`, даже если оно не изменилось. Вы можете добавить какой-нибудь флаг изменения переменной `$count` и записывать новое значение в зависимости от состояния этого флага, например, 0 — изменений не было и, следовательно, записывать нечего, 1 — нужно обновить данные. Это довольно несложно и я оставляю вам это в качестве домашнего задания...

«Отладка» счетчика

Попробуйте запустить этой счетчик. Если вы запускаете его для отладки на своей домашней машине, закомментируйте строку 10 и замените строкой

```
@$count=$count+1;
```

Это нужно сделать, потому что, как я уже замечал, сценарий не учитывает посещения с локального хоста. При первом запуске вы должны увидеть на экране 1, при повторном посещении или нажатии на кнопку «Reload» (в т.ч. и на «Shift» + «Reload») значение счетчика не изменится. Если вы используете Netscape 6, для просмотра установленных Cookies выберите команду меню **Edit ⇒ Preferences**, а потом **Advanced ⇒ Cookies**.

К недостаткам нашего счетчика нужно отнести то, что если на сайт заходит пользователь с отключенными Cookies, счетчик воспринимает его каждый раз, как уникального посетителя, даже если он уже был на сайте. И это логично — Cookies ведь отключены.

20.1.2. Создание графического счетчика

Теперь можно немного и усложнить наш счетчик. Дело в том, что текстовые счетчики практически никто не использует — в основном выводится какое-нибудь графическое изображение. Вот сейчас и займемся созданием графического счетчика. Это увеличит объем исходного текста счетчика всего лишь на 7 строк. Вместо `echo $count`, то есть вывода результата, мы выведем рисунок (см. листинг 20.2).

Листинг 20.2. Графический счетчик

```
<?
 $f=fopen("counter.dat","a+") or die("Cannot to open file");
 flock($f,2);
 $count=fread($f,100);
 if(!IsSet($Was))
  {
  $Was=1;
  SetCookie("Was",$Was,0x7FFFFFFF);
  // Не засчитываем посещения со своего узла
  $REMOTE_HOST=gethostbyaddr($REMOTE_ADDR);
 if(!($HTTP_HOST===$REMOTE_HOST)) @$count=$count+1;
 ftruncate($f,0);
 fwrite($f,$count);
 }
 flock($f,3);
 fclose($f);
$img = ImageCreateFromPng("images/template.png"); // строка 16
$color = ImageColorAllocate($img, 220, 210, 60); // строка 17
$x = (ImageSx($img)-6.5*strlen("$count"))/2;      // строка 18
ImageString($img,3,$x,9,"Visits: $count",$color); // строка 19
Header("Content-type: image/png");                // строка 20
ImagePng($img);                                   // строка 21
ImageDestroy($img);
?>
```

Как видите, все очень просто. Строка 16 (для большего комфорта строки я пронумеровал) создает рисунок из файла `images/template.png` (должен существовать). То есть функция **ImageCreateFromPng** загружает фон для нашего счетчика. Потом (строка 17) мы определяем цвет букв — в данном случае должен получиться желтый (оранжевый) цвет. В строке 19 выводим в рисунок строку `"Visits: число"`. Строка будет расположена под наклоном. В строке 20 происходит самое главное: мы передаем браузеру тип рисунка — PNG. Потом сохраняем рисунок в стандартный вывод — в браузер. Последняя строка освобождает память, выделенную под рисунок.

Использовать этот счетчик можно так: <img src="counter.php">

Возможно, вы попытаетесь выполнить данный пример, и получаете сообщение об ошибке... Примерно такое: `Fatal error: Call to undefined function: imagecreatefrompng() in ...`

Оказывается, как всегда, есть одно НО. Для работы с графикой у вас должна быть установлена библиотека GD. Если вы работаете под Linux, вы должны установить PHP с поддержкой библиотеки GD. Даже если у вас эта библиотека и установлена, вы должны пересобрать PHP для поддержки библиотеки GD. При работе под Windows убедитесь, что у вас есть файл `php_gd.dll`. Если вы все настроили и протестировали сценарий на своей домашней машине, то это не значит, что он будет работать у вашего хостинг-провайдера. Некоторые хостинг-провайдеры не включают (или забывают включить) поддержку библиотеки GD.

Это еще пол беды. При работе с PHP (и включенной поддержкой GD) у вас просто может не быть функции **ImageCreateFromPng()**. В этом случае попробуйте функции **ImageCreateFromJpeg()** или **ImageCreateFromGif()**. Естественно, в зависимости от типа изображения, нужно корректно установить Content-type и использовать нужную функцию (**ImageJpeg()** — для Jpeg или **ImageGif()** — для Gif).

20.1.3. Протоколирование посещений

Методика протоколирования

Допустим, мы хотим знать, кто и когда посещал наш сайт, а также какой браузер был у пользователя. Потом эти данные можно будет использовать для формирования статистики посещений. Для этого мы будем использовать сервер баз данных MySQL.

Здесь можно пойти двумя путями: по пути наименьшего сопротивления и по немного аскетичному пути. Первый заключается в том, что, не усложняя себе жизнь, мы будем записывать в базу данных всех подряд — даже если пользователь уже был на сайте. Второй путь заключается в том, что мы будем вносить в базу только уникальных пользователей. Как вы догадались, мы пойдем по второму пути. Вариантов решения данной проблемы множество. Вы можете использовать предложенный мною метод, а можете написать свой.

Так как счетчик уже считает уникальных посетителей, то можно было бы просто добавить один запрос SQL, который бы вносил в базу нужную нам информацию. Но этот способ я считаю некорректным. Во-первых, мы будем использовать таблицу с индексами (такой подход в дальнейшем ускорит операции описка по таблице), а пользователь может отключить Cookies или попросту удалить старые Cookies и он уже не будет

уникальным. Таким образом, мы попытаемся внести еще одну запись, которая уже есть в таблице, выражаясь языком BDE (Borland Database Engine), мы получим "Key Violation", то есть нарушение индекса. Во-вторых, даже если мы будем проверять, есть ли пользователь в базе, мы получим довольно большую погрешность в подсчете уникальных посетителей: по статистике около 30% пользователей отключают Cookies.

Немного непонятно? Я так и знал. Сейчас все станет на свои места. Мы будем использовать таблицу с такими полями: `IP char(15), vdate char(8), browser char (127).`

Поле IP (IP-адрес клиента), естественно, должно быть уникальным, то есть по нему будем строить индекс. Поле `vdate` — дата *первого* визита (формат mm.dd.yy), поле `browser` — наименование клиента браузера. 127 символов для этого поля, конечно же много, но, а вдруг Microsoft выпустит IE8, включив в его описание имена всех разработчиков.

Реализация протоколирования

Базу данных мы создавать не будем (у нас ведь только одна таблица), а воспользуемся уже существующей стандартной базой `test`. Сейчас нужно создать таблицу. Для этого запустите программу **mysql test**.

Под Linux вам, скорее всего, нужно будет использовать пароль: **mysql test -p** Введите такой запрос (вы должны иметь право на создание таблицы):

```
create table stat(ip char(15) primary key, vdate char(8),
browser char(127));
```

После того, как мы создали таблицу нужной структуры, приступим к написанию самого сценария. Данный код нужно вставить после установки Cookies (то есть после строки 8, листинг 20.1).

```
    // подключаемся к базе данных
mysql_connect("localhost");
mysql_select_db("test");
    // считаем, что хост - уникальный
$Unique = 1;

$R=@mysql_query("SELECT * FROM stat") or die("Invalid SQL
query");

// Является ли хост уникальным
while($Rw=mysql_fetch_row($R))
{ if($Rw[0]===$REMOTE_ADDR)
  { $Unique = 0; break(1); }
}
```

```
    // если хост уникальный...
    if($Unique)
    {
    $vdate=date("d.m.y");
    @mysql_query("INSERT INTO stat
VALUES('$REMOTE_ADDR','$vdate','$HTTP_USER_AGENT')");
    }
    else
    {
    $vdate=date("d.m.y");
    mysql_query("UPDATE stat SET vdate='$vdate',
browser='$HTTP_USER_AGENT' WHERE ip='$REMOTE_ADDR'");
    }
```

Во второй строке происходит подключение к базе данных `test`, в которой мы создали таблицу `stat`. Мы подключаемся, используя имя пользователя по умолчанию. Скорее всего, на вашей домашней машине такой способ будет работать. А вот в случае с хостинг-провайдером вам так просто не подключиться. Для этого нужно использовать еще два дополнительных параметра: имя пользователя и пароль. Имя пользователя и пароль вы можете уточнить у вашего хостинг-провайдера, возможно оно будет совпадать с вашим логином при входе в систему:

```
mysql_connect("localhost","user","password");
```

В вышеприведенном примере я подключаюсь как пользователь `user` с паролем `password`. Еще раз напомню, что вы должны иметь право на создание таблицы (не говоря уже о целой базе данных — именно поэтому я предпочел использовать базу данных `test`).

Теперь рассмотрим алгоритм работы сценария. После подключения мы считаем, что хост уникальный. И это правильно, так как наши Cookie не установлены на машине клиента. Мы могли бы, как я уже писал, просто занести хост в базу данных, но пользователь может и отключить Cookies. Для этого нужно проверить есть ли он в нашей базе. Сначала мы получаем все уникальные хосты, которые есть в базе, потом обрабатываем их. Если хост не уникальный, мы сбрасываем флаг уникальности (переменная `$Unique`) и прерываем цикл. Если хост уникальный, мы вносим данные в базу, в противном случае мы обновляем информацию о последнем визите и браузере пользователя.

Теперь приведу полный исходный текст нашего, уже не простого, счетчика. Нужно сделать небольшое замечание: после того, как мы добавили операции с базой данных, потребность в строке 10 листинга 20.1 просто отпадает. Вместо нее строку `$count=$count+1` нужно добавить в блок обработки уникального хоста. Код счетчика приведен в листинге 20.3.

Листинг 20.3. Счетчик с протоколированием

```
<?
$f=fopen("counter.dat","a+") or die("Cannot to open file");
flock($f,2);
$count=fread($f,100);
if(!IsSet($Was))
 {
  $Was=1;
  SetCookie("Was",$Was,0x7FFFFFFF);
  // подключаемся к базе данных
  mysql_connect("localhost");
  mysql_select_db("test");

 // считаем, что хост - уникальный
 $Unique = 1;
 $R=@mysql_query("SELECT * FROM stat") or die("Invalid SQL
query");

 while($Rw=mysql_fetch_row($R))
 { if($Rw[0]===$REMOTE_ADDR)
   { $Unique = 0; break(1); }
 }
    // если хост уникальный...
if($Unique)
 {
 $vdate=date("d.m.y"); $count=$count+1;
 @mysql_query("INSERT INTO stat
VALUES('$REMOTE_ADDR','$vdate','$HTTP_USER_AGENT')");
 }
 else
 {
 $vdate=date("d.m.y");
 mysql_query("UPDATE stat SET vdate='$vdate',
browser='$HTTP_USER_AGENT' WHERE ip='$REMOTE_ADDR'");
 }
 ftruncate($f,0);
 fwrite($f,$count);
 }

flock($f,3);
fclose($f);
echo $count;
?>
```

И напоследок одно важное замечание: наверняка вам захочется вынести подключение к серверу MySQL в начало сценария. Этого не следует делать! Эти строки могут породить какой-нибудь текст в случае ошибки подключения и тогда мы не сможем установить Cookies. Информация о Cookies должна содержаться в заголовке, а если мы отправим какой-нибудь текст в браузер, то уже не сможем отправить продолжение заголовка, содержащего информацию о Cookies. В случае ошибки мы хотя бы успеем установить Cookies (если они включены).

20.1.4. Разработка системы статистики

Предположим, что вы создаете собственную систему статистики наподобие SpyLog. Другими словами, вы раздаете всем желающим счетчики, которые после установки на их сайтах, будут считать посещения, а потом пользователи смогут просмотреть статистику.

Прежде всего, нам нужно обеспечить идентификацию самих сайтов. Для этого будем использовать идентификационный номер сайта — `id`. Установить наш будущий счетчик на сайте клиента можно с помощью следующего тега:

```
<img src=counter.php?id=171>
```

В данном случае идентификатор сайта равен 171. Сразу оговоримся, что идентификатором по умолчанию будет номер 1 (наш сайт). Для подсчета посещений нашего сайта счетчик можно установить так:

```
<img src=counter.php>
```

А можно явно задать идентификатор, то есть id = 1.

Общее число уникальных посетителей для каждого сайта будет храниться в таблице `ids` базы данных `counter`. Хочу обратить ваше внимание на то, что в отличие от примера из предыдущего пункта, здесь мы будем использовать «свою собственную» базу данных. Структура таблицы ids выглядит так:

```
id int primary key, unq int
```

Где `id` — идентификатор сайта, а `unq` — количество уникальных посещений.

Отдельно статистика для каждого сайта будет записываться в таблицу `idN`, где `N` — это идентификатор сайта, например, `id171`. Структура таблиц `idN` будет такова:

```
ip char(15), vdate char(8), browser char (127)
```

Думаю, теперь все стало ясно. Нам осталось только реализовать все это программно. Полный текст счетчика с протоколированием приведен в листинге 20.4.

Листинг 20.4. Простая система статистики

```
<?
// Идентификатор не указан - наш сайт
if (!isSet($id)) $id=1;
// Таблица для статистики
$table="id$id";
// подключаемся к базе данных
mysql_connect("localhost");
mysql_select_db("counter");
// считаем, что хост - уникальный
$Unique = 1;
$R=@mysql_query("SELECT * FROM $table") or die("Invalid SQL
query 1");
while($Rw=mysql_fetch_row($R))
 { if($Rw[0]===$REMOTE_ADDR)
   { $Unique = 0; break(1); }
 }

// если хост уникальный...
if($Unique)
 {
 $R=mysql_query("SELECT * FROM ids WHERE id=$id") or die("Invalid
SQL query 2");
 $Rw=mysql_fetch_row($R);
 $counter=$Rw[1];
 $vdate=date("d.m.y");
 $counter=$counter+1;
// обновляем счетчик
 @mysql_query("UPDATE ids SET unq=$counter WHERE id=$id");
// обновляем информацию о посетителе
 @mysql_query("INSERT INTO $table
VALUES('$REMOTE_ADDR','$vdate','$HTTP_USER_AGENT')");
 }
 else
 {
 $vdate=date("d.m.y");
  mysql_query("UPDATE $table SET vdate='$vdate',
browser='$HTTP_USER_AGENT'
WHERE ip='$REMOTE_ADDR'");
 }
$img = ImageCreateFromPng("images/template.png");
$color = ImageColorAllocate($img, 220, 210, 60);
$x = (ImageSx($img)-6.5*strlen("$counter"))/2;
ImageString($img,3,$x,9,"Visits: $counter",$color);
Header("Content-type: image/png");
ImagePng($img);
ImageDestroy($img);
?>
```

Теперь несколько замечаний. Во-первых, подключение к серверу, которое используется в сценарии, считается некорректным и возможно лишь при неправильном конфигурировании сервера баз данных (когда допускается вход пользователя по умолчанию). В общем случае вам нужно указать имя пользователя и пароль для доступа к серверу баз данных. На практике подключение к серверу MySQL выглядит так:

```
$SERVER = "localhost";
$USER = "user";
$PASSWD = "top_secret";

if(!mysql_connect($SERVER,$USER,$PASSWD))
{
echo "<h1>Произошла ошибка при подключении к серверу MySQL</h1> ";
echo "<p>Сервер $SERVER</p>";
// Пароль никогда не выводим!!!!
echo "<p>Имя пользователя $USER</p>";
exit;
}
```

Во-вторых, таблица idN уже должна существовать! Поэтому, прежде чем «обеспечить» пользователя кодом счетчика, его нужно зарегистрировать, то есть создать нужную таблицу. Для решения этого вопроса есть два метода: просто добавить в сценарий запрос создания нужной таблицы после подключения к серверу или же написать собственный сценарий регистрации пользователей. Первый метод, хотя и рабочий, но некорректный, потому, что при уже имеющейся таблице, данный запрос будет порождать ошибку 1050: таблица существует. Поэтому, сейчас займемся написанием сценария для регистрации.

Этот сценарий должен работать примерно так: находим номер N последнего зарегистрированного пользователя (он будет максимальным), потом присваиваем новому пользователю номер N+1 и создаем таблицу idN+1.

Для регистрации следующего пользователя можно использовать сценарий из листинга 20.5.

Листинг 20.5. Регистрация новых пользователей

```
<?
// подключаемся к базе данных
mysql_connect("localhost");
mysql_select_db("counter");
$R=mysql_query("SELECT * FROM ids") or die("Invalid SQL query
1");

$max=mysql_num_rows($result);
```

```
// новый номер
$N=$max+1;

// Таблица для статистики
$table="id$N";
mysql_query("insert into ids values ($N,0)");

// создаем таблицу
mysql_query("create table $table (ip char(15) primary key, vdate
char(8),
browser char (127))");

echo "Поздравляем! Регистрация завершена успешно!";
echo "Установите следующий код на вашу страничку: ";
echo "<br>&ltimg src=\"reg.php?id=$N\"> ";
?>
```

Вместо явного указания имени сценария (reg.php) вы можете использовать переменную окружения $SCRIPT_NAME. Как видите, сценарий для регистрации новых пользователей также предельно прост — всего 17 строк кода. В этом сценарии я не рекомендую вам использовать символ @ перед названиями функций для работы с MySQL. В случае возникновения ошибки пользователь увидит соответствующее сообщение, а не будет несколько раз запускать сценарий, думая, что ему почему-то отказали в регистрации. Этот сценарий можно усложнить, но это выходит за рамки этой главы — предоставлю вам пространство для творчества.

Практически все готово для нашей системы статистики. Осталось написать небольшой сценарий, который бы показывал пользователю статистику обращений к его сайту. Этот сценарий представлен в листинге 20.6.

Листинг 20.6. Сценарий просмотра статистики

```
<?
mysql_connect("localhost");
mysql_select_db("test");

if (!isSet($id)) $id=1;
$table="id$id";

$R=mysql_query("select unq from ids where id=$id");
$Rw=mysql_fetch_row($R);
$unq=$Rw[0];

$R=mysql_query("select * from $table");
```

```
echo "<html><body><h1>Здравствуйте, уважаемый пользователь!</h1>";
echo "Уникальных посетителей: $unq<br>";

// выводим таблицу со статистикой
echo "<table border>";
echo "<tr><td>Хост</td><td>Последнее посещение</td><td>Браузер
</td></tr>";

while($Rw=mysql_fetch_row($R))
 {
  echo "<tr>";
  for($i=0; $i<mysql_num_fields($R); $i++)
  echo "<td>$Rw[$i]</a></td>";
  echo "</tr>";
 }
echo "</table>";
echo "</body>";
?>
```

Данный сценарий не использует никакой аутентификации. Процедуру проверки правильности имени пользователя вы можете при необходимости добавить самостоятельно. При вызове сценария ему нужно передать всего один параметр — id — идентификатор сайта.

Сценарий можно еще модифицировать, добавив функцию разрешения IP-адреса в доменное имя, однако это замедлит работу сценария.

У нас получилась простенькая система статистики посещения. Наша система выводит только имя узла, последнее посещение пользователя с этого узла и браузер пользователя. Кроме того, она выводит количество уникальных посетителей. То есть, если один и тот же пользователь зайдет дважды, наша система его не «посчитает».

«Серьезные» системы статистики, кроме рассмотренной выше информации, ведут учет по странам (например, вы можете узнать, сколько пользователей зашло к вам из России, а сколько из Франции и США), браузерам (выводится отчет, какой браузер самый популярный). Не плохо бы «придумать» систему аутентификации, а то любой желающий может просмотреть вашу статистику, узнав ваш номер, а это сделать достаточно просто — нужно зайти к вам на сайт и просмотреть страничку со счетчиком в формате HTML. Правда, даже если кто-то и просмотрит статистику, особой беды в этом нет — изменить-то он ничего не сможет. Для придания нашей системе «товарного вида» вам придется ни один час (а может и день) провести за монитором, однако основа у вас уже есть.

20.2. Анализ серверной статистики (кто откуда приходил и что делал). Программа Webalizer

20.2.1. Методика сбора и анализа серверной статистики. Серверные журналы

Для представления статистических данных нужно проанализировать журналы сервера. Вручную это делать, скажем, не совсем удобно, поэтому лучше написать программу, автоматизирующую этот процесс. Но можно воспользоваться уже готовыми программами: зачем заново изобретать колесо? Одной из таких программ является программа **Webalizer** (`http://www.mrunix.net/webalizer/`), разработанная Бредфортом Барретом (Bradford L. Barrett). Данная программа анализирует журналы сервера и генерирует статистические данные в формате HTML на основе информации, полученной из log-файлов. Webalizer также умеет рисовать красивые гистограммы для лучшего восприятия информации. Отчеты Webalizer ничем не уступают отчетам SpyLog и многим коммерческим счетчикам, а иногда даже превосходят их.

Программа Webalizer поддерживает формат журналов сервера CLF (Common Log Format), поэтому с помощью этой программы вы можете анализировать работу и других серверов, поддерживающих данный формат. Кроме формата CLF Webalizer поддерживает форматы журналов FTP-серверов **wu-ftpd и ProFTPD**, а также формат журнала прокси-сервера **Squid**. Согласно формату CLF, файлы журналов должны иметь такую структуру:

```
IP ДФ Дата Действие КО ОПИ URL Браузер
```

Где:

- IP — это IP-адрес удаленного узла;
- ДП — дополнительные флаги;
- Действие — действие, произведенное сервером;
- КО — код ответа сервера, например, 200 — передача произведена успешно, 404 — ошибка: файл не найден;
- ОПИ — объем переданной информации;
- URL — URL, введенные в поле Location браузера.

Рассмотрим пример файла `access_log`, который используется Web-сервером Apache для учета доступа пользователей:

```
192.168.1.1 - - [28/May/2002:15:27:32 +0300] "GET /mrtg/mrtg-
r.gif HTTP/1.1"
404 299 "http://www.server.ru/mrtg/unix-guide.html" "Mozilla/5.0
(X11;
U; Linux i686; en-US; rv:0.9.2.1) Gecko/20010901"
```

Эта запись означает, что 28 мая 2002 года в 15:27 некто с удаленного узла пытался получить файл (метод GET) /mrtg/mrtg-r.gif, используя протокол передачи HTTP 1.1. Но у этого пользователя ничего не вышло, потому что такой файл не найден на сервере, о чем свидетельствует код ответа 404. Размер страницы, содержащей сообщение об ошибке, и которая была передана пользователю, составляет 209 байт. Пользователь в поле браузера Mozilla ввел URL http://www.server.ru/mrtg/unix-guide.html.

Формат журналов сервером FTP отличается от формата CLF и имеет такую структуру:

Дата Время Год ВП Узел Объем_перед_инф Файл Режим_передачи СО ФП Пользователь НП Служба

Где:

- ВП — время передачи;
- ФП — флаг пользователя;
- НП — направление передачи.

Рассмотрим пример файла Xferlog, который является журналом сервера FTP:

```
Wed Jun 12 11:49:35 2002 1 localhost.localdomain 1490 /home/
den/vmware.html
a _ o r dhsilabs ftp 0 * c
Wed Jun 12 11:50:08 2002 1 localhost.localdomain 281 /home/den/
code.bin
a _ o r dhsilabs ftp 0 * c
Wed Jun 12 11:50:15 2002 1 localhost.localdomain 281 /home/den/
code.bin
a _ i r dhsilabs ftp 0 * c
Wed Jun 12 11:52:08 2002 1 localhost.localdomain 888 /home/den/
ftphosts.html
b _ i r dhsilabs ftp 0 * c
```

Из первой записи вы можете узнать, что пользователь dhsilabs зарегистрирован с удаленного узла localhost.localdomain. Начало передачи файла /home/den/vmware.html произошло в среду, 12 июня 2002 года в 11:49. Общее время передачи — одна секунда. Общий объем переданной информации составляет 1490 байт. Для передачи файла использовался режим ASCII (a), не было произведено никаких специальных операций (_). Файл vmware.html пользователь загружал с сервера, на что указывает направление передачи (o). Пользователь dhsilabs является реальным (зарегистрированным) пользователем системы (r). Название службы, которая производила операцию — ftp.

Теперь рассмотрим четвертую запись. Тот же пользователь dhsilabs передал на сервер файл ftphosts.html. Направление передачи — на сервер (i). Режим передачи — двоичный (b).

Вторая и третья записи сообщают о загрузке с сервера и на сервер файла `code.bin` в текстовом (a) режиме.

Тип пользователя обозначается символом возле имени пользователя. Символ r обозначает зарегистрированного в системе пользователя, у которого есть своя запись в файле `/etc/passwd`. Символ **g** — означает гостевую регистрацию, а **a** — анонимную.

Направление передачи, как вы уже догадались, обозначается символами **o** — прием файла пользователем и **i** — прием файла сервером.

На основе такой информации Webalizer генерирует статистику. Информация, предоставляемая программой Webalizer, более информативна, чем информация обыкновенного счетчика. Webalizer может генерировать отчеты о посещении вашего Web-сервера за год, месяц, день и час. Обычно отчеты помещаются в каталог `<DocumentRoot>/usage`, например, `/var/www/html/usage`, но месторасположение этих файлов будет зависеть от настроек вашего Web-сервера и Webalizer.

20.2.2. Установка и запуск Webalizer

Вы можете загрузить последнюю версию программы по адресу `http://www.mrunix.net/webalizer/` или установить версию, которая входит в состав вашего дистрибутива (в большинстве случаев эта программа уже будет установлена у вас, если вы используете операционную систему Linux Red Hat 7 и выше). Если программа не установлена, установить ее можно командой:

```
rpm -Uhv /mnt/cdrom/RedHat/RPMS/webalizer*
```

Естественно, перед вводом данной команды нужно смонтировать диск и правильно указать путь к пакетам RPM.

Если у вас установлена операционная система Windows, скачать Windows-версию программы Webalizer можно по одному из адресов:

- `http://www.medasys-lille.com/webalizer/Download.htm` (русская версия, 123 Кб)
- `http://www.mrunix.net/webalizer/download.html` (более полная версия, 794 — рекомендую)

Честно говоря, работа русской Windows-версии мне не очень понравилась. Дело в том, что весь вывод программы представлен в кодировке KOI8-R, а в Windows использует CP1251. С текстовой информацией проблем нет — выбрал в браузере нужную кодировку, и все нормально, а вот с картинками действительно проблема — вместо русских букв непонятно что.

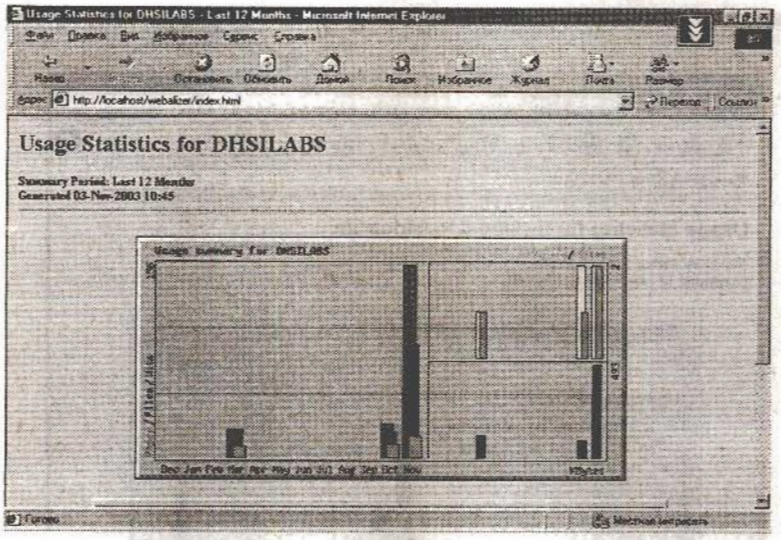


Рис. 20.1. *Программа Webalizer для Windows*

Далее мы будем работать в Linux, но в Windows все то же самое. Запустите программу **webalizer**:

```
webalizer
```

При запуске на экране вы ничего не увидите, но после завершения работы в каталоге `/var/www/html/usage` появятся файлы отчета. Затем убедитесь, что ваш Web-сервер запущен и в окне браузера введите URL:

```
http://localhost/usage
```

В окне браузера вы увидите примерно такую информацию (см. рис. 20.2).

Перейдя по ссылке на страницу, содержащую ссылку на статистические данные за определенный месяц, вы можете просмотреть:

1. Ежедневную статистику (Daily Statistics).
2. Почасовую статистику (Hourly Statistics).
3. Вводимые клиентами URL (URLS), определив таким образом самые посещаемые страницы.
4. Другую информацию (IP-адреса, страны клиентов, браузеры пользователей).

Весьма желательно добавить запуск программы **webalizer** в расписание демона **crontab** для автоматического обновления статистики каждый час. Для этого в ваш файл `/etc/crontab` добавьте строку:

```
01 * * * * root /usr/bin/webalizer
```

Убедитесь, что демон crond запускается автоматически при запуске системы.

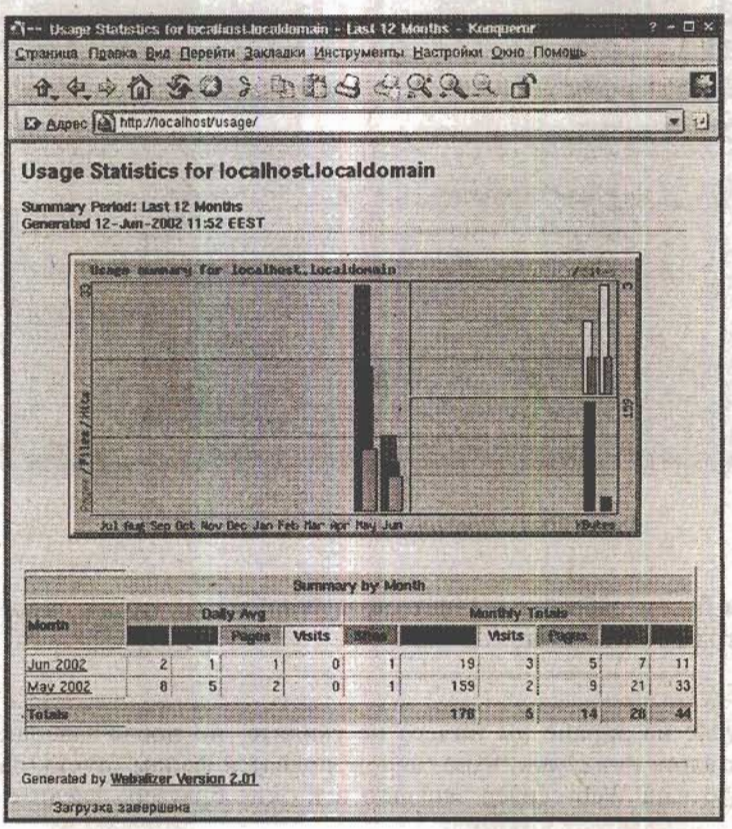


Рис. 20.2. *Программа webalizer для Linux*

Программу **webalizer** можно запускать с параметрами, приведенными в табл. 20.1.

Параметры запуска ***webalizer*** *Таблица 20.1*

Параметр	Описание
-c файл	Указывает альтернативный файл конфигурации. По умолчанию используется файл /etc/webalizer.conf
-n имя	Указывает имя машины, которое будет отображено при выводе статистики
-o каталог	Указывает каталог, в который будут помещены файлы статистики
-t название	Указывает заголовок отчета
-F (clf | ftp | squid)	Указывает тип журнала: **clf** или **ftp** или **squid**

20.2.3. Конфигурационные параметры Webalizer

Если вам нужно считать посещения нескольких виртуальных Web-серверов, я рекомендую для каждого виртуального сервера создать свой файл конфигурации и вызывать **webalizer** только с параметром **-c**, а все остальные параметры можно будет указать в конфигурационных файлах отдельно для каждого виртуального сервера.

О других параметрах программы **webalizer** вы можете узнать, введя команду **man webalizer**, а сейчас мы рассмотрим директивы конфигурационного файла `/etc/webalizer.conf`.

- **LogFile** — определяет журнал сервера. Значение по умолчанию:

  ```
  LogFile /var/log/httpd/access_log
  ```

- **LogType** — указывает тип журнала: web или ftp. Значение по умолчанию:

  ```
  LogType web
  ```

- **OutputDir** — определяет месторасположение отчетов Webalizer. Значение по умолчанию:

  ```
  OutputDir /var/www/html/usage
  ```

- **HistoryName** — задает имя файла, в котором будет содержаться информация о посещении Web-сервера на протяжении одного года.

  ```
  HistoryName webalizer.hist
  ```

  Формат этого файла такой:

  ```
  5 2002 33 21 1 159 28 31 9 2
  6 2002 11 7 1 19 8 12 5 3
  ```

  Первые два числа — это месяц и год, а остальные числа — это статистические данные.

- **Incremental.** Возрастающая обработка позволяет использовать несколько относительно небольших файлов журналов вместо одного огромного. Данная опция полезна для больших сайтов, период ротации журналов которых меньше, чем период ротации обыкновенных Web-серверов. Обычно ротация журналов выполняется 1 раз в месяц, но если сайт очень большой, целесообразно выполнять ротацию 1 раз в неделю, сократив размер основного журнала. Именно для таких случаев полезно использовать возрастающую обработку. Файл `webalizer.current`, который используется при возрастающей обработке и задается с помощью директивы **IncrementalName,** содержит текущие статистические данные. Более подробную информацию о возрастающей обработке вы найдете в файле README. По умолчанию возрастающая обработка не используется.

- **ReportTitle.** Директива **ReportTitle** задает заголовок отчета. Значение по умолчанию:

  ```
  ReportTitle Usage Statistics for
  ```

- **HostName**. С помощью этой директивы можно указать имя Web-сервера. Обычно имя Web-сервера разрешается с помощью DNS, а эту директиву можно не определять.
- **HTMLExtension** — определяет расширение для файлов-отчетов. Значение по умолчанию:

  ```
  HTMLExtension html
  ```

- **PageType**. Директива **PageType** указывает расширения файлов, которые будут считаться Web-страницами. Значения по умолчанию:

  ```
  PageType htm*
  PageType cgi
  ```

- UseHTTPS — позволяет использовать безопасное соединение HTTPS для просмотра статистики. По умолчанию безопасное соединение не используется.
- **Quiet**. Директива **Quiet** подавляет все сообщения при работе программы. Это удобно при запуске программы с помощью демона **crond**. Значение по умолчанию:

  ```
  Quiet yes
  ```

- **ReallyQuiet**. Директива **Quiet** не подавляет сообщения об ошибках и предупреждениях при работе программы. Если вы хотите отключить и эти сообщения, включите директиву **ReallyQuiet**. По умолчанию данная директива выключена:

  ```
  ReallyQuiet no
  ```

Все остальные директивы не столь существенны и в основном относятся к внешнему виду создаваемых отчетов. С помощью этих директив вы можете настроить создаваемые директивы на свой вкус. Я надеюсь, что этой информации вполне достаточно, чтобы настроить Webalizer и легко получать статистическую информацию о работе своего Web-сервера.

Глава 21

Сессии в PHP

21.1. Обзор механизма сессий

21.1.1. Особенности CGI и недостатки Cookies

Одним из недостатков интерфейса CGI является то, что любая CGI-программа не может длительное время "общаться" с пользователем: она должна запуститься, обработать переданные ей параметры и выдать какой-нибудь результат. И на этом все.

Теперь представьте себе, что вам нужно написать некоторую анкету, но полей в ней так много, что они не умещаются в одном окне браузера (а из-за этого, как всегда, страдает дизайн). В результате возникает необходимость разместить поля анкеты на нескольких HTML-страничках, которые бы последовательно появлялись по мере заполнения анкеты. Все бы хорошо, да только возникает проблема хранения данных, внесенных на предыдущих страницах анкеты при заполнении следующих страниц. Дело в том, что по сценарию-то они должны быть переданы все вместе целиком. То есть, вот вы заполнили одну страницу анкеты, перешли к следующей, а где в это время должны храниться ранее внесенные данные?

Данную проблему можно решить путем использования временного файла. Но, предположим, что в один момент с анкетой работают два или более пользователей, тогда вариант с файлами отпадает сразу же — анкетные данные пользователей просто-напросто перемешаются, тут даже не поможет блокировка временного файла. Тогда вам нужно будет использовать базу данных, но стоит ли использовать целую базу данных для хранения промежуточных данных? Более «продвинутые» станут использовать Cookies. Это хорошее решение, но PHP предоставляет более гибкое решение этой проблемы — механизм сессий. К тому же, у пользователя, зашедшего на ваш сайт, Cookies могут быть отключены. Поэтому решение может быть такое: по умолчанию используются Cookies, но если они отключены, ваши сценарии, использующие сессии, все равно будут работать. О том, как это достигается, поговорим немного позже.

Недостаток Cookies также заключается в том, что не предусмотрен механизм согласования имен переменных Cookies. Например, я написал

сценарий, который использует переменную $DB, содержащую имя файла анкеты пользователя. Потом пользователь зашел на сайт к Марку, который использует переменную с таким же именем, но для иных целей, например, для выбора базы данных MySQL. Так как переменная $DB уже установлена, сценарий Марка будет ее использовать, а она содержит совсем не то значение, которое от нее ожидается. В результате сценарий Марка будет работать некорректно. При использовании же механизма сессий коллизии имен отпадают сразу.

Второй недостаток Cookies, о котором мы уже говорили, заключается в том, что сценарии, которые используют Cookies, не будут работать при отключенных Cookies. Механизм сессий не требует от пользователя включения Cookies.

21.1.2. Механизм сессий: принцип работы

Итак, сессии — это механизм, позволяющий однозначно идентифицировать пользователя (браузер), а также создать для этого пользователя файл на сервере, в котором хранятся нужные переменные сеанса в течение всей работы. Для идентификации пользователя используется идентификатор сессии — SID (Session IDentifer). Именно с его помощью сервер может определить, какой пользователь к нему обращается, и выбрать нужные переменные из соответствующего файла.

Для этого пользователю (браузеру) от сервера необходимо передать какое-то уникальное значение SID, а потом запрашивать его у пользователя (браузера) при каждом обращении к серверу. Как же осуществляется передача SID?

Идентификатор сессии SID — это обычная переменная, которая обычно называется $PHPSESSID. Мы уже знаем, что переменную можно передавать либо через Cookies, либо через POST/GET запрос (в качестве параметра). Язык PHP позволяет использовать оба этих метода для обмена SID между сервером и пользователем(браузером).

С использованием Cookies мы уже знакомы. Когда пользователь заходит на сайт, ему присваивается определенное значение SID и записывается в Cookies. На сервере заводится соответствующий файл, в котором будут храниться переменные текущего сеанса. В течение дальнейшей работы сервер будет брать значение SID из Cookies и по нему выбирать переменные из файла с переменными. Этот способ обмена SID является наиболее предпочтительным, так как весь обмен происходит незаметно для пользователя. Однако он не работает, если у пользователя отключены Cookies.

Гарантировать передачу SID может второй способ — через POST/GET запросы, то есть через адресную строку. Суть метода заключается в следующем. Когда пользователь первый раз заходит на сайт, сервер (с помощью PHP), перед тем как передать страницу сайта пользователю, во все ссылки автоматически добавит параметр SID. Например, если изначально была ссылка:

```
<a href=page2.php>Страница 2</a>
```

то после того, как в нее будет добавлен SID, она станет:

```
<a href=page2.php?PHPSESID="7401e923e7aca6d8c66bbf9db9cf6b08">Страница
2</a>
```

Таким образом, видно, что к ссылке был добавлен параметр — это и есть значение SID.

Когда пользователь будет бродить по сайту, он будет нажимать на различные ссылки. В результате вместе с URL-адресом он автоматически будет передавать значение SID. Таким образом, никаких Cookies не надо. Единственное неудобство — это в адресной строке появляются какие-то наборы непонятных символов (то есть SID), которые могут озадачивать пользователя.

Да, еще в рамках второго способа, если на странице имеется форма, то в нее будет автоматически добавлено невидимое поле `<input type="hidden" name= "PHPSESID" value="7401e923e7aca6d8c66bbf 9db9cf6b08">`, автоматически передающее вместе с остальными параметрами формы еще и значение SID.

За то, какой способ передачи идентификатора сессии SID использовать, отвечают две специальные настройки файла `php.ini`:

- **session.use_cookies** — значение 1 означает передачу SID через Cookie, а значение 0 — выключает этот способ передачи;
- **session.use_trans_sid** — значение 1 включает передачу SID через присоединение его к URL-адресам, а значение 0 — отключает его.

По умолчанию включенными являются оба способа передачи SID. Когда пользователь в первый раз заходит на сайт, то ему и Cookies прописывается, и ссылки дополняются. При следующем запросе, сервер пытается сначала считать SID из Cookies.

Если ему это удается, то он и в дальнейшем будет это делать, а дополнение ссылок будет прекращено. Если же у пользователя отключены Cookies и, соответственно, сервер не смог из Cookies прочитать SID, то сервер в этом и во всех последующих случаях будет использовать передачу-приемку SID через автодополнение ссылок.

21.2. Инициализация сессии и регистрация в сессии переменных

Рассмотрим самый простой пример использования сессий (см. листинг 21.1).

Листинг 21.1. Простая сессия

```
<?
session_name("SessionOfIvan");
session_start();
session_register("a");
$a=@$a+1;
echo "<html><body>Нажмите Reload, чтобы увеличить счетчик";
echo "<br>Счетчик: $a";
echo "</body></html>";
?>
```

Функция **session_start()** инициализирует сессию, а функция **session_register("a")** регистрирует в сессии переменную `$a`. Имена переменных задаются без знака доллара. Можно несколько раз регистрировать одну и ту же переменную в сессии (что я и сделал в листинге 21.1) — ее значение не обнулится при повторной регистрации. Если запустить наш сценарий и несколько раз нажать кнопку браузера Reload (Обновить), счетчик будет увеличиваться, а если закрыть окно браузера, счетчик обнулится.

Способ обращения к сессионным переменным зависит от настроек в файле **php.ini**. Если опция **register_globals** включена (**register_globals=on**), то обращение к сессионной переменной практически ничем не отличается от обращения к обычным переменным:

```
session_register("a");
$a = 5;
```

В том случае, если опция **register_globals** в файле `php.ini` выключена (**register_globals=off**), то к сессионным переменным нужно будет обращаться через глобальный массив `$HTTP_SESSION_VARS`:

```
session_register("a");
$HTTP_SESSION_VARS[ "a"] = 5;
```

По умолчанию опция **register_globals** включена.

Примечание.

В файле `php.ini` можно задать полезный параметр — место, в котором будут храниться файлы с сессионными переменными. Его можно задать с помощью опции **session.save_path**: `session.save_path = /tmp`.

Следует также отметить, что в общем случае сценарии, которые одновременно выполняются, ничего не знают друг о друге. Другими словами, если вы запустите сценарий и пару раз нажмете на кнопку «Обновить» (Reload) для изменения счетчика, а потом, не закрывая окна браузера, независимо запустите браузер снова и запустите еще раз этот же сценарий — значение счетчика будет нулевым (в данном случае — 1). Если же вы откроете еще одно окно браузера из уже запущенного браузера (например, в Internet Explorer — это команда меню **Файл ⇒ Создать ' Новое окно**), значение счетчика не обнулится.

Проверить, зарегистрирована ли переменная в сессии, позволяет функция **session_is_registered(string $name)**. Отменить регистрацию какой-нибудь переменной можно с помощью функции **session_unregister(string $name)**. Отменить регистрацию ВСЕХ переменных сессии можно с помощью функции **session_unset()**.

21.3. Имя сессии

Рассмотрим один пример. Предположим, вы написали какой-нибудь сценарий с использованием сессий (ту же анкету), и зарегистрировали в нем переменную `FirstName`. Далее предположим, что пользователь заполняет аналогичную анкету другого разработчика и в ней также используется переменная `FirstName`. В результате значения переменных перемешаются. Чтобы избежать этого, нужно присвоить нашей сессии какое-нибудь имя, например, `SessionOfIvan`. Это можно сделать с помощью функции **session_name()**. Функцию **session_name()** нужно вызвать ДО инициализации сессии, в противном случае толку от нее будет мало (см. листинг 21.2). Если имя сессии не указано, PHP будет использовать имя сессии по умолчанию — PHPSESID.

Листинг 21.2. Сессия с именем

```
<?
session_name("SessionOfIvan");
session_start();
session_register("a");
?>
```

Мы уже знаем, для чего используется SID. Что же такое имя сессии? Попростому можно объяснить так: имя сессии — это название группы сессии, к которой принадлежит какой-нибудь SID.

Функция **session_id([string $sid])** возвращает значение идентификатора сессии. Эта функция может также и изменить текущий SID, если указать параметр `$sid`.

Примечание.

PHP позволяет инициализировать сессию автоматически — сразу при запуске сценария без вызова функции **session_start()**. Для этого нужно установить параметр **session.auto_start=1** в файле конфигурации `php.ini`. Однако я не рекомендую этого делать, потому что в этом случае будет создана "безымянная сессия" и вы не сможете установить имя сессии с помощью функции **session_name()**.

Еще одна полезная функция — **session_save_path([string $path])**. Она возвращает имя каталога, в котором будут находиться временные файлы сессии. Обычно это каталог `/tmp`. С помощью параметра `$path` можно указать другой каталог.

21.4. Еще один способ (устаревший) обойти использование Cookies в сессиях

Мы уже говорили, что PHP, в случае отключения Cookies, автоматически добавляет SID во все ссылки выдаваемой пользователю Web-страницы. Однако в старых версиях PHP этой возможности не существовало. При этом SID добавлялся вручную. Это можно проделать и сейчас, но только зачем это делать при наличии автоматического механизма, не совсем понятно. Однако, может вам и пригодится. На всякий случай рассмотрим и этот способ.

Перед выполнением следующего примера (листинг 21.3) отключите Cookies в вашем браузере, если это не сделать, то PHP будет использовать Cookies и пример будет несколько не так работать.

Листинг 21.3. Пример сессии без Cookies. Сценарий sesq.php

```
<?
session_name("SessionofIvanov");
session_start();
session_register("a");
$a=@$a+1;
echo "<html><body>Нажмите Reload, чтобы увеличить счетчик";
echo "<br>Счетчик: $a";
?>
<a href=sesq.php?<?=SID?>>Click here</a>"
```

Обратите внимание на последнюю ссылку: при *отключенных Cookies* PHP генерирует константу SID. Если Cookies включены, данная константа ничего не содержит (см. рис. 21.1). Для работы нашего сценария первый раз нужно перейти по этой ссылке вместо нажатия на «Reload» (Обновить) в браузере. Потом уже можно будет нажимать и на «Reload».

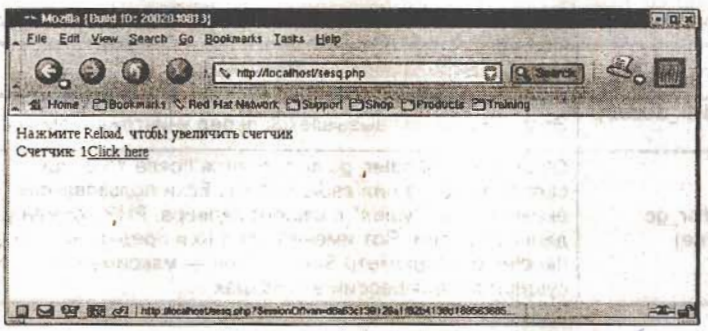


Рис. 21.1. *Использование константы SID*

При использовании константы SID (см. предыдущий листинг) интерпретатору PHP кажется, что данные об идентификаторе и имени сессии пришли из Cookies браузера.

21.5. Обработчики сессии

Обработчики сессии (это обыкновенные PHP-функции) запускаются, как только произошло какое-нибудь событие, связанное с сессией, например, открытие или закрытие сессии. Обычно запускается стандартный обработчик, который выполняет действия по умолчанию. Если же вам стандартные действия не нравятся, или вы хотите написать свой механизм сессий, стандартные обработчики можно переопределить.

В табл. 21.1 описаны все стандартные обработчики сессии PHP 4/5.

Стандартные обработчики сессий *Таблица 21.1*

Обработчик	Назначение
session_handler_open (string $save_path, string $session_name)	Данный обработчик вызывается при инициализации сессии, то есть, когда вызывается функция session_start(). Обычно этот обработчик открывает базу данных (или какой-нибудь файл) для группы сессий $session_name
session_handler_close()	Данный обработчик вызывается, когда нужно закрыть временное хранилище (вся информация уже записана)
session_handler_read (string $sid)	Функция вызывается, когда нужно прочитать данные сессии. В качестве параметра передается идентификатор сессии — $sid. Функция возвращает данные сессии в формате: name1=value1; name2=value2;...; nameN=valueN; Под значением подразумевается сериализированные с помощью функции Serialize данные. Например, var1|i:10;var2|i:20, в сессии были сохранены две переменные — var1 и var2. Значение первой — *10*, второй — *20*
session_handler_write (string $sid, string $data)	Функция предназначена для записи данных сессии с указанным идентификатором. Параметр $data задается в вышеописанном формате

Стандартные обработчики сессий *Таблица 21.1 (продолжение)*

Обработчик	Назначение
session_handler_destroy (string $sid)	Этот обработчик вызывается перед уничтожением сессии
session_handler_gc (int $maxlifitime)	Обработчик handler_gc вызывается после того, как сценарий завершил свою работу. Если пользователь окончательно "ушел" с вашего сервера, PHP должен удалить данные сессии. Вот именно для это и предназначена функция handler_gc. Параметр $maxlifetime — максимальное время существования сессии в секундах

Примечание.

Обработчик **session_handler_gc()** вызывается не каждый раз при завершении работы сценария, а только иногда. Когда именно — это зависит от параметра **session.gc_probability** файла конфигурации. Этот параметр определяет (в процентах) вероятность запуска обработчика **session_handler_gc()** для очередной "генеральной уборки":

```
session.gc_probability = 1
```

Максимальное время жизни "мусора" определяется параметром **session.gc_maxlifetime** (в секундах):

```
session.gc_maxlifetime = 1440
```

Вы, наверное, заметили, что имена обработчиков начинаются префиксом `session_handler`. Это не является обязательным: такой префикс я использовал для большего понимания и, главным образом, акцентирования вашего внимания, что данные функции являются обработчиками сессии. Вы же можете использовать любое удобное для вас имя.

Для переопределения обработчиков используется функция:

```
session_set_save_handler($open,$close,$read,$write,$destroy,$gc).
```

В качестве параметров ей нужно передать имена функций, которые будут использоваться как обработчики. Данную функцию нужно вызывать ДО инициализации сессии (вызова функции **session_start()**), иначе она просто будет проигнорирована.

Теперь рассмотрим пример переопределения обработчиков, который я позаимствовал из официальной документации по PHP (см. листинг 21.4).

Листинг 21.4. Пример переопределения обработчиков handlers.php

```
<?
// Обработчик открытия сессии
function open ($save_path, $session_name)
{
   echo "open ($save_path, $session_name)\n";
   return true;
}
```

```
// Обработчик закрытия сессии
function close()
{
   echo "close\n";
   return true;
}

// Чтение из сессии
function read ($key)
{
   echo "write ($key, $val)\n";
   return "foo|i:1;";
}

// Запись во временное хранилище
function write ($key, $val)
{
   echo "write ($key, $val)\n";
   return true;
}

// Ничего не делает
function destroy ($key)
{
   return true;
}

// Ничего не делает
function gc ($maxlifetime)
{
   return true;
}

session_set_save_handler ("open", "close", "read", "write",
"destroy", "gc");

session_start();
$foo++;
?>
```

Теперь выполните этот сценарий и в результате вы должны увидеть примерно следующее:

```
X-Powered-By: PHP/4.3.3
Set-Cookie: PHPSESSID=7401e923e7aca6d8c66bbf9db9cf6b08
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-cache, post-check=0, pre-check=0
Pragma: no-cache
Content-type: text/html
```

```
open (/tmp, PHPSESSID)
write (7401e923e7aca6d8c66bbf9db9cf6b08, )
write (7401e923e7aca6d8c66bbf9db9cf6b08, foo|i:2;)
close
```

21.6. Сохранение в сессии ассоциативного массива

Рассмотрим полезный пример — сохранение в сессии ассоциативного массива. Предположим, что нам нужно написать анкету, но вопросов достаточно много и они не помещаются в одном экране браузера. Для удобной работы с анкетой нам нужно разбить вопросы на части. Ответы на вопросы будем записывать в массив `ank`, например:

- `ank[count]` — сколько раз пользователь посещал страницу, по этому полю можно ориентироваться, какую страницу с вопросами нужно вывести в браузер;
- `ank[name]` — имя пользователя;
- `ank[surname]` — фамилия пользователя;
- `ank[year]` — год рождения;
- ...

Мы сейчас не будем писать всю анкету, а рассмотрим только, как сохранить ассоциативный массив в сессии (см. дистинг 21.5).

Листинг 21.5. Сохранение ассоциативного массива в сессии

```
<?

session_start();
// регистрируем в сессии переменную $ank - это и есть массив
session_register("ank");

if (!isset($ank['count'])) {
// Элемент массива count не установлен, следовательно,
// пользователь пришел впервые
       $ank['count']=1;
}
else $ank['count']++;

echo "Вы загружали эту страницу $ank[count] раз(а) <p>";

// Пользователь ввел имя, нужно записать это имя в
// массив $ank, ключ name
if (strlen($name)>1) $ank['name']=$name;
```

```
// Выводим форму с просьбой ввести имя
if (!isset($ank['name']))
        echo "<form action=$SCRIPT_NAME><input type=text name=name>
        <input type=submit></form>";
    else
        echo "Вы зарегистрированы под именем: $ank[name]";
?>
```

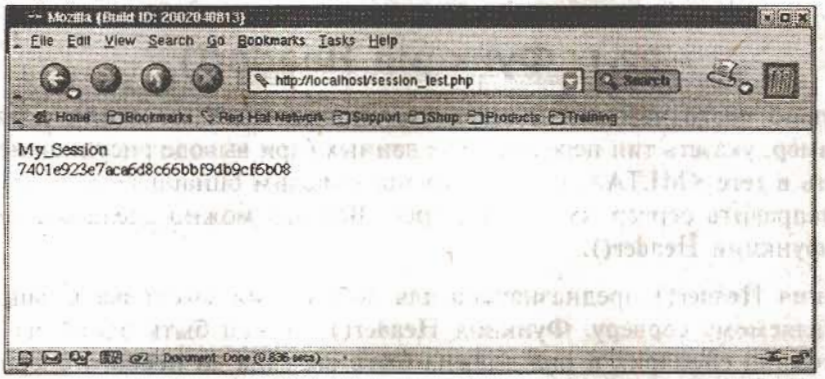


Рис. 21.2. *Сценарий session.php в работе*

Работа с WWW

22.1. Функция Header()

Довольно часто нам нужно передать специальный заголовок серверу, например, указать тип передаваемых данных (при выводе рисунков нельзя сделать в теге <META>, потому что мы выводим бинарные данные) или перенаправить сервер на новый адрес. Все это можно сделать с помощью функции **Header()**.

Функция **Header()** предназначена для добавления заголовка к запросу, отправляемому серверу. Функция **Header()** должна быть одной из первых команд сценария и она должна быть вызвана до первого оператора вывода (echo или функции, которая что-нибудь выводит в браузер), иначе вы получите предупреждение, и ваш заголовок не будет добавлен к запросу. Помните, что скобки **<?** и **?>** также считаются операторами вывода, поэтому убедитесь, что перед первой скобкой не было пробелов или пробельных символов.

Простенький сценарий, представленный в листинге 22.1, перенаправит браузер на сайт www.mail.ru:

Листинг 22.1. Перенаправление браузера на сайт www.mail.ru

```
<?
Header("Location: http://www.mail.ru");
?>
```

22.2. Запрет кэширования

Если вы разрабатываете сценарий, выводящий часто изменяющиеся данные (например, прогноз погоды или цены акций), нужно запретить кэширование браузером и proxy-серверами, иначе пользователь может получить устаревшие данные, которые находятся в кэше браузера или proxy-сервера.

Следующая функция отключает кэширование документа, выводимого сценарием:

```
function DisableCache()
{
// Указываем, что документ "годен" до 4 апреля 1995 года,
// следовательно,
// он не будет прокэширован и браузер запросит его заново
Header("Expire: Mon, 4 Apr 1995 01:01:01 GMT");
// Запрещаем кэширование, HTTP/1.1
Header("Cache-Control: no-cache, must-revalidate");
// Запрещаем кэширование, HTTP/1.0
Header("Pragma: no-cache");
// А это для браузера
Header("Last-Modified: ".gmdate("D, d M Y H:i:s")." GMT");
}
```

Функцию **DisableCache()** нужно вызывать всякий раз, когда вам нужно запретить кэширование, но обязательно до первого оператора вывода — ведь мы устанавливаем заголовки. Не пытайтесь использовать только один из четырех заголовков — нужны все четыре, в противном случае, документ будет прокэширован браузером или proxy-сервером.

22.3. SSI и PHP. Включение одних сценариев и html-страниц в другие

22.3.1. Общие положения

Что такое SSI? Аббревиатура SSI (Server Side Includes) в переводе с английского — это Включения со Стороны Сервера. Предположим, что у нас есть два HTML-документа, и мы хотим вставить оба документа в определенное место в третьем документе. Тогда в третьем (`main.shtml`) HTML-файле нужно написать две инструкции:

```
<!--#include virtual="/first.html" -->
<!--#include virtual="/second.html" -->
```

Документы `first.html` и `second.html` будут «вставлены» сервером в документ `main.shtml` и отправлены пользователю. Предположим, что первый документ содержит следующий код:

```
<h1>First Document</h1>
```

А второй — такой код:

```
<h2>Second Document</h2>
```

Тогда пользователь получит такой файл main.html:

```
<h1>First Document</h1>
<h2>Second Document</h2>
```

Это и есть SSI. В отличие от фреймов, где вспомогательные файлы загружаются в браузере и отображаются отдельно (в отдельных областях — фреймах), SSI обеспечивает добавление документов еще на самом сервере, то есть документ собирается из отдельных частей не в браузере, а уже приходит в него в собранном виде. Пользователь даже не заметит, что на самом деле полученный им документ был составлен из нескольких частей.

Примечание.

Это похоже на инструкцию **include** в языках программирования — текст **include**-файла вставляется препроцессором вместо директивы **include**.

22.3.2. SSI и PHP. Функция virtual

Для одного документа нельзя применить более одного обработчика. Это означает, что нам нужно выбирать — или PHP, или SSI, вместе — нельзя. Разработчики PHP предусмотрели этот неприятный момент и написали функцию **virtual()**, которая может заменить директиву <!--#include virtual="..." -->:

```
int virtual(string $url)
```

Функция **virtual()** будет работать только, если PHP установлен как модуль, а не запускается отдельно.

Обычно функцию **virtual()** нужно использовать для работы с SSI-файлами сложной структуры или для запуска CGI-программ, написанных на другом языке программирования. Если же вам просто нужно включить в вывод сценария какой-нибудь файл, проще это сделать с помощью следующих операторов:

```
$htm = join('',file("http://localhost/main.html"));
echo $htm;
```

Функция **virtual()** непригодна для подключения обычных PHP-файлов — для этого используйте директивы **include** и **require**.

22.4. Загрузка файла на сервер

Multipart-формы и их описание

Загрузка файлов на сервер по протоколу HTTP осуществляется намного чаще, чем вы можете подумать:

- Web-интерфейсы почтовых сервисов позволяют добавить к письму вложение, а для этого нужно сперва загрузить файл на сервер, а только после этого добавлять к письму.
- Интерактивные фотогалереи и фотоальбомы просто не могут существовать без механизма загрузки файлов.
- Web-интерфейсы средств управления контентом, которые используются для управления сайтом, позволяют загружать файлы на сервер в указанный каталог.

Загрузка файла на сервер осуществляется с помощью так называемой multipart-формы, в которой есть поле загрузки файла — <input type=file>. При этом качестве параметра **enctype** указывается значение **multipart/form-data**:

```
<form action=upload.php method=POST enctype=multipart/form-data>
<input type=file name=UFile>
Имя файла на сервере <input type=text name=Filename>
<input type=submit name=upload value=Загрузить>
</form>
```

Как выглядит Multipart-форма можно посмотреть на рис. 22.1.

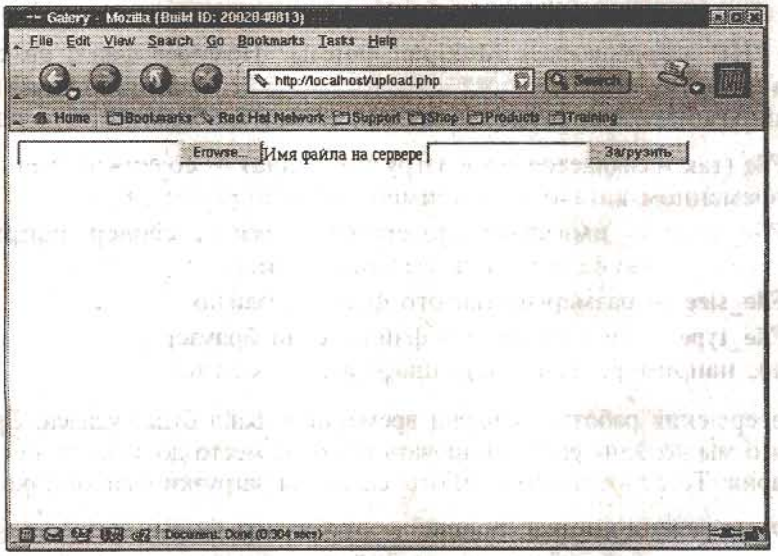


Рис. 22.1. Multipart-форма

Multipart-формы обычно используют метод передачи POST. Как видно из вышеприведенного примера данная форма обычно имеет два поля:

- Поле файла выбора файла для закачки <INPUT type=File>.
- Поле указания имени файла, которое он должен будет иметь на сервере <INPUT type=text>.

Сценарий загрузки файлов на сервер. Обработка multipart-форм

Прежде, чем приступить к написанию сценария загрузки файла, нужно отредактировать файл конфигурации `/etc/php.ini`, чтобы разрешить загрузку файлов. Откройте файл конфигурации и найдите секцию **File uploads**. В ней будут три параметра:

- **file_uploads = On** — разрешает загрузку файлов на сервер по протоколу HTTP.
- **upload_tmp_dir** = /tmp — устанавливает каталог для временного хранения загруженных файлов.
- **upload_max_filesize = 2M** — устанавливает максимальный объем загружаемых файлов.

Если ваш Web-сервер работает под управлением операционной системы Linux, нужно перезапустить сервис:

```
service httpd restart
```

Под Windows 9x/NT ничего делать не нужно, поскольку PHP не встраивается, как модуль в Apache.

Как же PHP обрабатывает Multipart-формы? Получив файл, он сохраняет его во временном каталоге `upload_tmp_dir`, имя файла выбирается случайным образом. Затем он создаст четыре глобальных переменных:

- **$UFile** (так называется поле загрузки файла) — содержит имя файла во временном каталоге, например, `/tmp/phpJ6r5eR`.
- **$UFile_name** — имя файла до его отправки на сервер, например, `/home/den/water.png` или `c:\water.png`.
- **$UFile_size** — размер принятого файла в байтах.
- **$UFile_type** — тип принятого файла (если браузер смог его определить), например, image/png, image/jpeg, text/html.

После завершения работы сценария временный файл будет удален. Это означает, что мы должны его скопировать в другое место до завершения работы сценария. То есть алгоритм работы сценария загрузки файлов такой:

- Если не нажата кнопка "Submit" — отобразить форму загрузки файла.
- Если кнопка "Submit" нажата, то файл уже будет загружен на сервер и его имя будет в переменной `$UFile`. В этом случае сценарий должен

сразу скопировать файл с именем $UFile в какой-нибудь каталог (вы должны иметь права записи в этот каталог). Вы НЕ можете сохранить имя файла в Cookies и скопировать его при следующем запуске сценария — файла с именем $UFile уже не будет — это обыкновенный временный файл и он будет удален при завершении работы сценария.

Копирование производится функцией **copy()**:

```
copy($UFile, "/var/www/html/uploads/".basename($UFile_name));
```

Нужно использовать только функцию копирования. Использовать функцию перемещения нет никакого смысла, поскольку:

- Временный файл будет удален автоматически.
- Если временный каталог находится на другом носителе, мы получим сообщение об ошибке.

Предположим, что нам нужно загрузить файл в каталог uploads, который находится в корневом каталоге Web-сервера (каталоге DocumentRoot).

```
// На всякий случай создаем каталог. Если он уже создан,
// сообщение об ошибке будет
// подавлено оператором @
@mkdir("uploads",0777);

// Копируем файл из /tmp в uploads
// Имя файла будет таким же, как до отправки на сервер
copy($UFile,"uploads/".basename($UFile_name));
```

В Linux все намного сложнее — нам ведь нужно учитывать права доступа к каталогу uploads. Скорее всего, функция **mkdir()** не сработает, потому что у нас нет прав на запись в каталог DocumentRoot (обычно это /var/www/html или /home/httpd/html). Зарегистрируйтесь в системе, как пользователь root, создайте каталог uploads и измените его владельца и права следующим образом:

```
// Создаем каталог uploads
mkdir uploads
// Устанавливаем имя владельца apache и его группу - тоже apache
chown apache:apache uploads
// разрешение записи всем (777) + установка закрепляющего бита
(1)
chmod 1777 uploads
```

Размер файла можно ограничить, при желании можно отредактировать файл .htaccess и ограничить доступ к каталогу uploads — указать или конкретных пользователей, которым можно обращаться к каталогу, или IP-адреса.

Вот теперь мы можем загружать файлы на сервер.

Организация виртуальной фотогалереи с возможностью самостоятельного размещения в ней фотографий пользователей

Рассмотрим небольшой пример: галерею картинок. Для работы галереи необходимы два сценария:

- `galery.php` — просмотр галереи;
- `upload.php` — загрузка картинок.

С первым файлом все ясно — он должен выводить картинки, которые содержит каталог galery (`/var/www/html/galery`), а также некоторую информацию о картинках — размер, дату загрузки, имя файла.

Второй файл более сложен — он должен обеспечивать аутентификацию пользователей, чтобы загружать файлы могли только определенные пользователи, а не кто угодно. Но мы упростим его задачу: ограничимся только загрузкой файлов без аутентификации пользователей — ее можно обеспечить средствами самого Apache. О том, как это сделать, написано в соответствующей главе в начале книги.

Возможно, еще понадобится сценарий для удаления ненужных картинок, но его написание — несложная задача, думаю, вы справитесь сами.

Перед написанием сценария создайте каталог `/var/www/html/galery` и установите к нему права доступа так, как мы это делали для каталога `uploads`.

Начнем со второго сценария. Алгоритм его работы очень прост:

- Если нам передан файл (нажата кнопка **upload**), загружаем файл на сервер.
- Если нам нечего принимать, выводим форму загрузки файла, которая будет обеспечивать также переименование файла — мы укажем новое имя файла, под которым он будет сохранен на сервере.

Форма, выводимая сценарием, изображена на рис. 22.1.

Для выбора файла нажмите кнопку "Browse" и выберите файл. Затем в поле "Имя файла" введите имя файла для сохранения на сервере, и нажмите кнопку "Загрузить".

После загрузки файла сценарий сообщит имя временного файла, а также новое имя файла (см. рис. 22.2).

Если уже существует файл с таким именем, которое указал пользователь, сценарий «припишет» к имени файла случайное число в диапазоне от 0 до 1000.

Листинг сценария `upload.php` приведен в листинге 22.2.

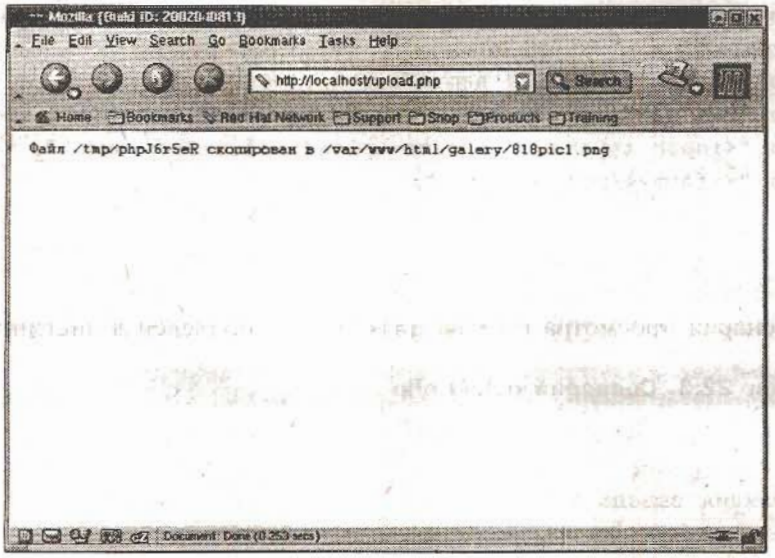


Рис. 22.2. *Копирование файла*

Листинг 22.2. Сценарий upload.php

```
<?

if (isset($upload))
 {
 // Мы получили файл
 $Filename="/var/www/html/galery/".$Filename;

 // Файл существует: "приписываем" к его имени случайное число
  if (file_exists($Filename))
  {
   mt_srand(time());
   $r = mt_rand(0,1000); // случайное число
   $Filename = "/var/www/html/galery/$r".basename($Filename);
   }

 // Копируем файл
  copy($File,$Filename);
  echo "<PRE> Файл $File скопирован в $Filename </PRE>";
 } else
 {
// Кнопка upload не нажата, выводим форму
 echo "<html><head><title>Galery</title></head><body>";
```

```
 echo "<form action=upload.php method=POST enctype=multipart/
form-data>";
 echo "<input type=file name=File>";
 echo "Имя файла на сервере <input type=text name=Filename>";
 echo "<input type=submit name=upload value=Загрузить>";
 echo "</form></body></html>";
  }

?>
```

Код сценария просмотра галереи galery.php приведен в листинге 22.3.

Листинг 22.3. Сценарий galery.php

```
<?

// Функция вывода галереи
/* ------------------------------------------------------- */
function view_gal()
{
// Пустой массив картинок
$Galery = array();
// Каталог галереи
$GalDir = "/var/www/html/galery";

// Ссылка на сценарий upload.php
echo "<html><head><title>Galery</title></head><body>";
echo "<center><a href=upload.php>Загрузка картинки</a><hr
width=100%>";

// Открываем каталог галереи
$dir = opendir($GalDir);

// Читаем все элементы галереи - файлы каталога $GalDir
while(($item = readdir($dir)) !==false)
 {
// Если расширение файла не GIF, JPG, PNG, переходим к следую-
щем файлу
   if(!ereg("^(.*)\\.(gif|jpg|png)$",$item,$p)) continue;

   $path_to_file="$item";
   // Получаем размер картинки (не файла!) и
   // время модификации файла
   $size = @GetImageSize($path_to_file);
```

```
  $time = @filemtime($path_to_file);

// Заполняем массив $Galery
  $Galery[$item] = array(
  'name' => $item,    // имя файла
  't' => $time,       // время
  'url' => $path_to_file,// URL
  'w'   => $size[0],   // ширина
  'h'  => $size[1],   // высота
  'wh' => $size[2],   // ширина x высота
  );
 }

// Сортируем галерею по именам файлов
 ksort($Galery);

 echo "<br><table><tr>";
 $count = 0;

// Выводим галерею в виде таблицы
 foreach($Galery as $k=>$v)
 {
// Выводим по три картинки
 if ($count % 3 == 0) echo "<tr>";
 echo "<td>";
 echo "<a href=http://localhost/galery/".@$v[url]."><img
src=http://localhost/galery/".@$v[url]." width=250 heigth=250>
<br>$v[url] $v[w] x $v[h]</a>";
 $time = @$v[t];
 echo date("d.m.Y H:i:s", $time);
 echo "</td>";
 $count++;
 }
 echo "</table></body></html>";
}

/* ------------------------------------------------------------ */

// Вызываем функцию просмотра галереи
view_gal();
 ?>
```

Вывод сценария `galery.php` изображен на рис. 22.3.

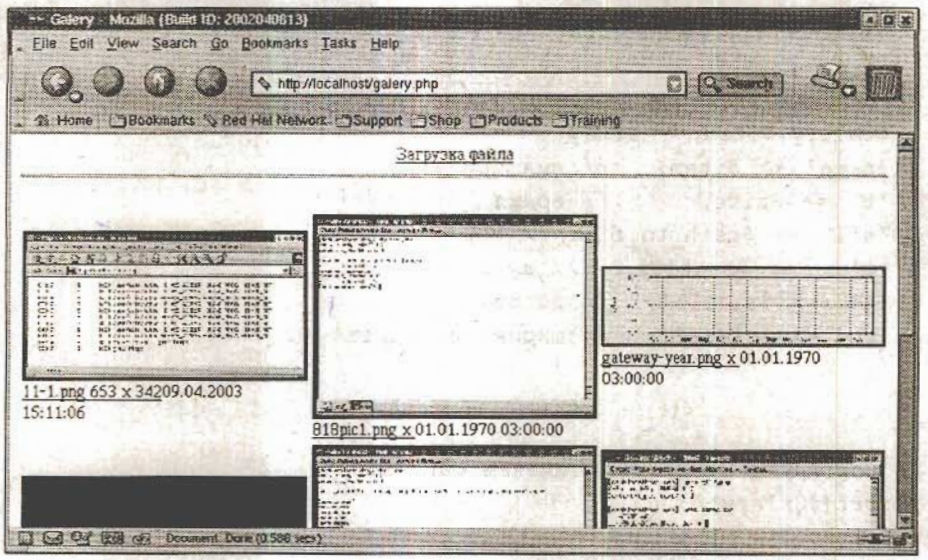


Рис. 22.3. *Просмотр галереи*

Глава 23

Создание и использование шаблонов. Модули

23.1. Разделение проекта на дизайнерскую часть и программирование

Для создания серьезного проекта необходимо, как минимум, два человека: один будет заниматься непосредственно программированием, а второй — разрабатывать дизайн будущего проекта. Такое разделение целесообразно практически всегда. Во-первых, универсальные специалисты встречаются довольно редко: кому-то нравится программировать, а кому-то — рисовать. Лично я предпочитаю первое, поэтому в этой книге вы не найдете всевозможных тонкостей Web-дизайна.

Во-вторых, даже если вы обладаете дизайнерскими способностями и далекие ваши родственники были художниками, все равно после многих часов, проведенных за монитором компьютера при написании кода, у вас вряд ли появится желание разрабатывать дизайн. А ведь проект еще нужно задокументировать — хотя бы написать простенький Readme с инструкциями по установке проекта на сервер хостера. Вот тут вам понадобится еще один человек — технический писатель, который задокументирует ваш проект и объяснит, как с ним работать, простым языком. Ведь очень часто случается, что, написав хорошую программу, у вас уже не хватает ни терпения, ни желания ее описывать.

Получается, что при разработке серьезного проекта происходит разделение на техническую и дизайнерскую части. К технической группе относятся программисты и технические писатели. Возможно, вам еще понадобится и системный администратор, который «разгрузит» вашу команду — вы будете заниматься программированием, не тратя время на администрирование вашей сети/сайта/домена.

К группе дизайнеров относится старший дизайнер и его помощники. Вообще, вы, как ведущий программист, будете контролировать работу всей группы: старший дизайнер будет отчитываться только вам, а работа его помощников и их количество вас вообще не должно интересовать.

Возможно, если так будет выгоднее, будет проще обратиться в какую-нибудь студию Web-дизайна с просьбой разработать дизайн для вашего проекта. Тут уж решать вам — или содержать свою собственную студию дизайна, или обратиться к профессионалам.

В этой главе мы уделим внимание технической части написания серьезного проекта, а именно, написанию библиотекаря, который поможет при разработке больших проектов.

Кстати говоря, все сказанное в значительной степени применимо и к случаю, когда проект создается и поддерживается одни человеком. При этом первостепенное значение имеет разделение двух процессов: написание кода и создание дизайна. Важно, чтобы эти два процесса были как можно обособленны. Поясню на примере. Можно сначала создать несколько сценариев — «кирпичики» сайта. А потом, разрабатывая дизайн сайта, уже эти кирпичики складывать и перекладывать как вам будет угодно. При этом вам не придется при каждом изменении дизайна переписывать весь свой проект.

23.2. Использование модулей. Создание библиотекаря

23.2.1. Общие положения

Практически во всех современных языках программирования высокого уровня предусмотрена возможность создания модулей. Основная программа подключает модуль, потом, в процессе работы, она вызывает функции, которые описаны в модуле. Разработчик выигрывает вдвойне: во-первых, уменьшается размер кода основной программы и программа выглядит читабельнее, а во-вторых, вы можете написать несколько программ, используя один и тот же модуль.

Сейчас мы попробуем написать свой механизм загрузки модулей. Вы можете спросить: зачем писать свой механизм загрузки модулей, если есть инструкции **include** и **require**? Представим такую ситуацию: что делать, если вы дважды включили один и тот же файл директивой **include**? Такое может случиться, если ваш проект действительно очень большой. Выход есть — использовать инструкции **include_once** и **require_once**. Но есть еще один аргумент в пользу библиотекаря: очень часто хостеры настраивают систему так, чтобы инструкции **include** и **require** искали файлы только в определенных каталогах, доступа к которым у вас нет. С их стороны все правильно — они заботятся о безопасности, а вот нас это не очень устраивает. Иногда вообще невозможно подключить файл с помощью **include** и **require**. Придется искать решение своими силами.

23.2.2. Создание библиотекаря

Определим требования к нашему библиотекарю:

- Библиотекарь должен загружать модуль (библиотеку функций) только один раз.
- Библиотекарь должен обеспечивать удобное конфигурирование путей поиска — каталогов, в которых находятся наши модули.
- Загрузка модуля должна осуществляться с помощью какой-нибудь функции. Мы ее назовем **Load()**.
- Библиотекарь должен загружаться автоматически.

Исходный код библиотекаря приведен в листинге 23.1.

Листинг 23.1. Файл library.php

```
<?
if(!defined("LIBRARY_IS_LOAD")) {

    // Библиотека загружена
    define("LIBRARY_IS_LOAD",1);

    // Расширение библиотечных файлов по умолчанию
    define("DefaultExt","lib");

    // Пути поиска по умолчанию
    $INC_DIRS = array(".","./lib","./include");

    // Преобразует относительный путь в абсолютный
    function absolute_path($path, $cur_path)
    {
    // заменим обратные слэши прямыми
    $path = strtr(trim($path),"\\","/");
    // теперь разобьем путь по прямым слэшам
    $APaths = explode("/",$path);

    // Если указан текущий каталог, то путь поиска
    // (Search) равен этому каталогу
    // В противном случае используем текущий
    // каталог (getcwd()).
    if ($cur_path==="") $Search=getcwd(); else
$Search=$cur_path;

    foreach($APaths as $k=>$v)
```

```
      if ($v!=".")
      {
      if(!$k && (strlen($v)>1&&$v[1]==":"||$v==="")) $Search=$v;
      elseif($v==="..")
      {
      if(strlen($Search)>1 && $Search[1]==":") continue;
      $p=dirname($Search);
      if($p==="/"||$p==="\\"||$p==".") $Search=""; else
$Search=$p;
      }
      elseif($v!==="") $Search=$Search."/$v";
      }
      return ($Search!==""?$Search:"/");
    }

    // Преобразует URL в абсолютный путь
    function UrlToPath($path)
    {
    $URL=dirname("$SCRIPT_NAME");
    $cURL=absolute_path(trim($path),$URL);
    return getenv("DOCUMENT_ROOT").$cURL;

    }

    // Преобразуем все пути в массиве $INC_DIRS в абсолютные
    function AbsoluteInclude()
    {
    static $DIR="";
    global $INC_DIRS;

    if ($DIR!==$INC_DIRS) {
    for($i=0; $i<count($INC_DIRS);$i++)
    {
    $value=&$INC_DIRS[$i];
    if ($value[0]==="." && (strlen($value)==1 ||
$value[1]=='\\' || $value[1]=='/')) continue;
    $value=absolute_path($value);
    }
    $DIR=$INC_DIRS;
    }
```

```
    }

    function Load($library)
    {
    global $INC_DIRS;
    static $DIR, $Last=0;

    AbsoluteInclude();
    $l=$Last;
    do {
    $dir=$INC_DIRS[$Last];
    if(@is_file($f="$dir/$library.".DefaultExt)) {
    $cwd=getcwd();
    chdir(dirname($f));
    foreach($GLOBALS as $k=>$v) global $$k;
    $res=include_once($f);
    AbsoluteInclude();
    chdir($cwd);
    return $res;
    }
    $Last=($Last+1)%count($INC_DIRS);
    } while ($Last!=$l);
    die("Library $library not found at ".join("; ",$INC_DIRS));
    }

    error_reporting(E_ALL);

    // После последнего тега >? не должно быть пробелов
    // или иных символов!
}
?>
```

Обратите внимание на следующие ключевые моменты. Массив $INC_DIRS содержит список каталогов, в которых будет производиться поиск файлов модулей. Сначала мы ищем файлы модулей в текущем каталоге, затем в каталоге `./lib`, а затем — в `./include`

```
$INC_DIRS = array(".","./lib","./include");
```

По умолчанию файлы модулей должны иметь расширение `.lib`. Если вас это не устраивает, измените переменную `DefaultExt`:

```
define("DefaultExt","lib");
```

23.2.3. Использование библиотекаря. Подключение модулей

Предположим, что у нас есть модуль `unit.lib` (см. листинг 23.2):

Листинг 23.2. Модуль unit.lib

```
<?

function func()
{
echo "Sample function from $SCRIPT_NAME library";
}

?>
```

Подключить этот модуль в основной программе мы можем так, как показано в листинге 23.3.

Листинг 23.3. Подключение модуля. Использование библиотекаря

```
<?
// Подключаем сам библиотекарь
include "library.php";

// Загружаем модуль
Load("unit");

?>
```

Добавить еще один каталог в пути поиска модулей можно так, как показано в листинге 23.4.

Листинг 23.4. Добавление каталога в качестве дополнительного пути поиска модулей

```
<?
// Подключаем сам библиотекарь
include "library.php";

$INC_DIRS[]="./my_libs";

// Загружаем модуль
Load("unit");

?>
```

Все хорошо, кроме загрузки самого библиотекаря. Ведь он по-прежнему загружается с помощью **include**. Тогда зачем все это мы затеяли?

Автоматически подключить модуль поможет сам PHP. Дело в том, что у большинства хостинг-провайдеров PHP установлен как модуль Apache,

а это позволяет указывать директивы файла `php.ini` в файле `.htaccess`. Создайте файл `.htaccess` (в корневом каталоге вашего виртуального Web-сервера) и добавьте в него строку:

```
php_auto_prepend_file /home/ваше_имя/www/library.php
```

Можно обеспечить автозагрузку библиотекаря и по-другому, но тогда вам придется редактировать либо файл конфигурации Apache, или файл конфигурации PHP, что вам не позволит сделать ни один хостинг-провайдер.

23.3. Использование шаблонов

В предыдущей главе мы, я искренне на это надеюсь, пришли к общему мнению, что для написания серьезного проекта необходимо разделение нашей небольшой Web-студии на два лагеря: лагерь программистов и лагерь дизайнеров. Как уже было сказано ранее, невозможно быть одновременно хорошим программистом и дизайнером: можно быть профессионалом только в одной области или любителем в обеих.

Дабы облегчить труд программиста мы уже написали библиотекарь, позволяющий создавать модули для нашего проекта и загружать их по мере необходимости. Наш модуль обеспечивал все возможности (ну, почти все) механизма загрузки модулей, реализованные в других языках программирования, например, Pascal.

В этой главе мы пойдем далее и облегчим труд дизайнера, а именно, отделим код PHP-сценария от его HTML-шаблона. Под HTML-шаблоном понимается чистая Web-страница до внедрения в нее PHP-кода. Именно разделение PHP-кода и HTML-шаблона позволит спокойно заниматься созданием дизайна Web-страницы в любом Web-редакторе, а потом подключить к нему PHP-скрипты.

Вы, наверное, заметили, что практически в каждой главе, где приводился исходный код PHP-сценариев, HTML-код был «завязан» вместе с PHP-кодом, то есть выводился непосредственно из сценария оператором **echo**.

Чтобы попросту не тратить время, убеждая вас в необходимости такого разделения, приведу небольшой пример. Попробуем взглянуть на следующий код глазами дизайнера (см. листинг 23.5).

Листинг 23.5. Пример сценария со «внедренным» HTML-кодом

```
echo "<table width=100% border=2 cellspacing=0 cellpadding=3
bordercolor=$main_color><tr><td>";
echo "<form method=post action=$SCRIPT_NAME>";
echo "<p align=center>Заказчик <input type=text name=client
size=30> ";
```

```
echo "Адрес <input type=text name=addr size=30> ";
echo "E-mail <input type=text name=email size=15> ";
echo "</td></tr></table><p>";

echo "<table width=100% border=2 cellspacing=0 cellpadding=3
bordercolor=$main_color><tr><td bgcolor=$main_color><font
color=white>Аппаратное обеспечение</td>";
echo "<td bgcolor=$main_color><font color=white>Программное
обеспечение</td></tr>";
echo "<tr><td valign=top>";

echo "<p><b>Процессор</b>";
echo "<br><input type=radio name=cpu value=C1 checked>Intel
Celeron 1.7 Ghz";
echo "<br><input type=radio name=cpu value=C2>Intel Celeron 2 Ghz";
echo "<br><input type=radio name=cpu value=P4>Intel Pentium 4 2 Ghz";
echo "<br><input type=radio name=cpu value=P42>Intel Pentium 4 2.4 Ghz";

echo "<p><b>Оперативная память (DDR)</b>";
echo "<br><input type=radio name=ram value=256 checked>256 MB";
echo "<br><input type=radio name=ram value=512>512 MB";
echo "<br><input type=radio name=ram value=1024>1024 MB";

...
```

У дизайнера, не знакомого с PHP, глядя на все это, глаза просто разбегаются. Думаете, он решится на правку этого кода? Конечно, нет, потому что он боится что-то нарушить, и это вполне естественно. А что, если нужно будет переделать дизайн проекта? Ведь рано или поздно наступит такой момент. И что тогда?

Проанализируем сложившуюся ситуацию. Включив HTML-код в код сценария, вы привязываете себя к этому сценарию: никто кроме вас не будет его редактировать, а тем более переделывать. Возможно, вам это даже выгодно, но это только пока — пока у вас один или два проекта и почему бы не получить дополнительных доход за доработку сценария? А что делать, если вы стали «серьезным» Web-программистом и у вас несколько новых *крупных* проектов в месяц, не говоря уже обо всех старых, написанных ранее. Вы получите дополнительную боль и дополнительные расходы, потому что вам нужно будет заплатить человеку — вашему помощнику за переработку ваших старых, написанных непрофессионально, сценариев, поскольку вам нужно будет заниматься новыми заказами. Ведь заказчика совершенно не волнует тот факт, что вам нужно переделать дизайн какого-то старого проекта — для него главное, чтобы вы вовремя сдали проект.

Все это была теория, теперь самое время перейти к практике. Предположим, что нам нужно написать небольшую ленту новостей. Чтобы вам стало понятнее, посмотрите на рис. 23.1: слева вы увидите небольшую табличку — это и будет наша лента новостей:

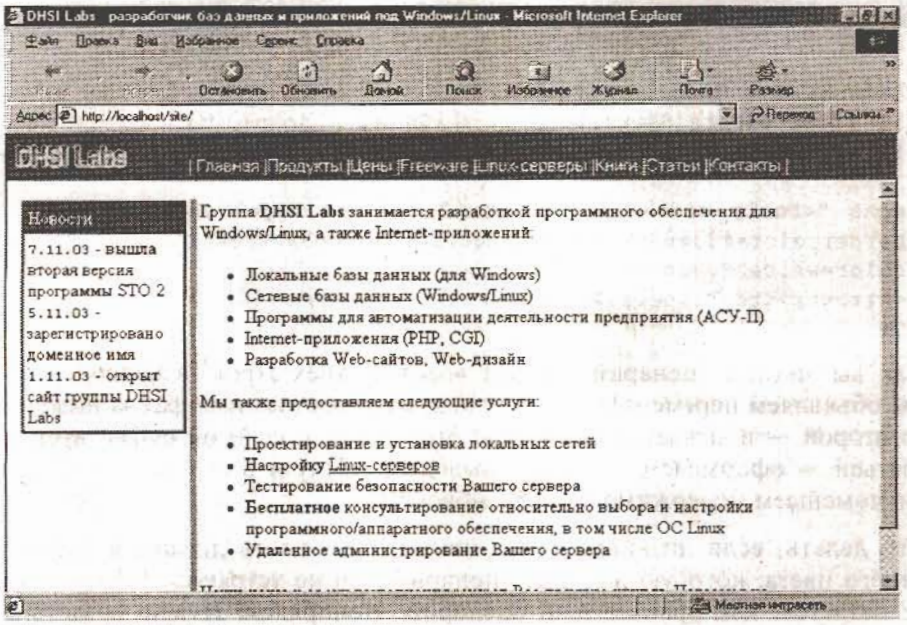


Рис. 23.1. *Лента новостей*

Наш сценарий для вывода таблички с новостями будет состоять из двух файлов:

- Сценария news.php.
- Файла с новостями — **add**/news.html.

Сразу нужно заметить, что нужно не только отделить PHP-код от HTML-кода, но и хранить их отдельно, чтобы дизайнер даже и не заходил в каталог с PHP-скриптами. В этой главе все вспомогательные HTML-файлы мы будем хранить в каталоге **add**.

Что же будет в файле `news.html`? В нем будут сами новости в HTML-формате, но без тегов <html>,<head> и <body>, то есть в этот файл нужно писать то, что должно быть в теге <body> (см. листинг 23.6).

Листинг 23.6. Файл новостей news.html

```
<tt>7.11.03</tt> - вышла вторая версия программы STO 2
<tt>5.11.03</tt> - зарегистрировано доменное имя
<tt>1.11.03</tt> - открыт сайт группы DHSI Labs
```

При такой организации мы значительно упростили работу дизайнера — ему достаточно отредактировать файл news.html, причем в этом файле не будет ничего, кроме HTML-кода.

Теперь перейдем к сценарию news.php (см. листинг. 23.7).

Листинг 23.7. Сценарий news.php

```
<?
$NewsFile = "add/news.html";
if (file_exists($NewsFile))    { $news = join( '',
file($NewsFile) ); } else $news="No news today";

echo "<table width=100% border=2 cellspacing=0 cellpadding=3
bordercolor=#336699><tr><td bgcolor=#336699><font
color=white>Новости</font></td>
</tr><tr><td>".$news."</td></tr></table>";
?>
```

Как вы видите, сценарий состоит всего из трех строк: в первой строке мы объявляем переменную $NewsFile, содержащую имя файла новостей, во второй — получаем содержимое этого файла, если он существует, а в третьей — оформляем табличку: выводим рамку и в середине этой рамки помещаем содержимое файла новостей.

Что делать, если дизайнер хочет немного изменить дизайн и табличка синего цвета, которую выводит сценарий, его не устраивает? Ему ничего не остается, как править наш сценарий. Непорядок! Нужно создать два шаблона — заголовок таблицы и footer-таблицы. Первый файл (add/t_header.html) приведен в листинге 23.8.

Листинг 23.8. Файл t_header.html

```
<table width=100% border=2 cellspacing=0 cellpadding=3
bordercolor=#336699>
<tr>
<td bgcolor=#336699><font color=white>Новости</font></td>
</tr>
<tr><td>
```

Второй файл (t_footer.html) будет содержать следующий HTML-код (см. листинг. 23.9).

Листинг 23.9. Файл t_footer.html

```
</td></tr>
</table>
```

Перепишем наш сценарий news.php так, чтобы он использовал оба эти файла. Окончательный код этого сценария приведен в листинге 23.10.

Листинг 23.10. Окончательный вариант сценария news.php

```
<?
$NewsFile = "add/news.html";
if (file_exists($NewsFile))    { $news = join( '',
file($NewsFile) ); } else $news="No news today";

include "add/t_header.html";
echo $news;
include "add/t_footer.html";

?>
```

Наш сценарий стал проще даже для программиста — ему не нужно знать, какой дизайн используется дизайнером — главное, чтобы сценарий выводил текст новостей, а все остальное для нас не важно. Большие «куски» HTML-кода усложняют восприятие исходного кода сценария, избавившись от них, мы убиваем сразу двух зайцем: помогаем и себе, и дизайнеру.

Как же «встроить» наш сценарий в проект. Если файл, в который вы хотите внедрить сценарий `news.php`, является PHP-сценарием, тогда используйте директиву **include** или **include_once**:

```
include "news.php";
include_once "news.php";
```

Можно также использовать библиотекарь:

```
Load("news");
```

Если же вы хотите внедрить наш сценарий в HTML-файл, тогда вам нужно будет произвести следующие действия:

1. Вставить в нужное место в вашем HTML-файле строку `<? include "news.php"; ?>`:

```
<table border=0 cols=3><tr>
<td width=20% valign=top>
<? include "news.php"; ?>
</td>
    ...
```

2. Создать в каталоге, в котором находится HTML-файл, файл `.htaccess`, в который нужно добавить строку:

   `AddHandler application/x-httpd-php .html`

   Этим мы свяжем интерпретатор PHP с расширением .html.
3. Скопировать файл `.htaccess` в каталог, в котором находится сценарий `news.php` — на всякий случай (если они находятся в одном каталоге) ничего делать не нужно.

В принципе, вот и все. Но есть одно но: вряд ли хорошую новость можно описать одной строкой. Даже если умудриться это сделать, ваш почтовый ящик будет забит сообщениями с просьбами рассказать о той или иной новости подробнее. Эту проблему можно решить так: при формировании файла новостей каждую новость записывать в виде:

```
<tt> дата </td> - <a href=DDMMYY.html target=_blank>Описание</a>
```

Но тогда вам придется оформлять каждую страницу с новостями отдельно. Заставим PHP сделать всю грязную работу за нас, то есть он будет выводить дизайн — шаблоны страниц, а мы — только создавать файлы в формате DDMMYY.html.

После написания данного сценария нужно будет предупредить дизайнера о новом формате ссылок: нужно будет ссылаться не на файлы `DDMMYY.html`, а на файл `i.php$d=DDMMYY` — дизайнеру ведь все равно, какую ссылку писать.

Сценарий `i.php` будет выводить подробное описание новости, параметр `$d` — это дата. Код сценария `i.php` приведен в листинге 23.11.

Листинг 23.11. Сценарий i.php

```
<?
if (isset($d)) {
// Переменная $d установлена
$fname="./news/".$d.".html";
if (file_exists($fname))  { $text = join( '', file($fname) );
} else $text="Новость от $d не найдена";

$text = join(file($fname),'');

include "add/news_header.html";

echo $text;

include "add/news_footer.html";

}
?>
```

Как обычно в файлах `DDMMYY.html` мы будем писать то, что должно быть в теге <body>, а файлы `news_header.html` и `news_footer.html` будут задавать дизайн странички.

Итак, наша небольшая новостная лента состоит из таких файлов:

- `news.php` — сценарий, обеспечивающий вывод новостей.
- `i.php` — сценарий, обеспечивающий вывод подробного описания новости.

- news.html — «база данных» новостей.
- t_header.html, t_footer.html — шаблоны таблицы списка новостей.
- news_header.html, news_footer.html — шаблоны вывода дополнительной информации.

На рис. 23.2 изображена схема нашей новостной ленты. При этом на нем изображены направления передачи данных между пользователем, сценариями и шаблонами.

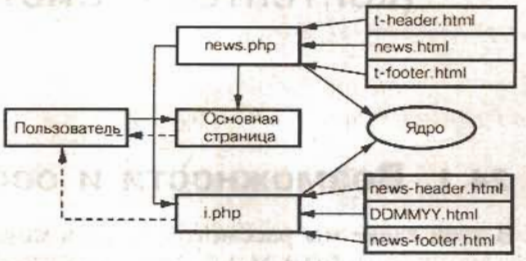


Рис. 23.2. Схема передачи данных

Вполне возможно, что вскоре наш проект разрастется до такой степени, что ему будут нужны вспомогательные функции, которые будут использоваться, как сценарием news.php, так и сценарием i.php. В этом случае имеет смысл вынести все общие функции в отдельный сценарий, который мы назовем ядром проекта — сценарий kernel.php. Каждый сценарий проекта должен будет подключить этот файл для своей работы инструкциями **include** или **include_once** (или функцией **Load()** — не забывайте также о нашем библиотекаре — не зря же мы его писали!). Ядро должно содержать «общие» функции и параметры проекта. Если же параметров слишком много, можно вынести их в отдельный файл — options.php — и подключать его в ядре инструкцией **include_once**:

```
include_once "options.php";
```

Использование ядра изменит нашу диаграмму. Теперь она будет выглядеть, как показано на рис. 23.3.

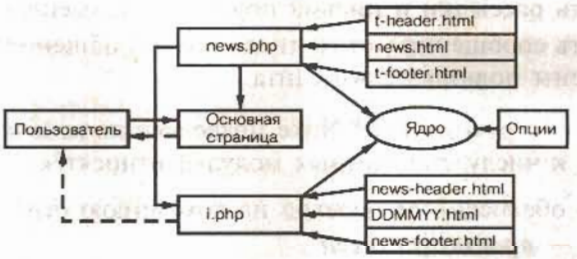


Рис. 23.3. Схема передачи данных с использованием ядра

Глава 24

Система PHP-Nuke — система управления содержимым сайта (контентом). Система PHP-Nuke

24.1. Возможности и особенности PHP-Nuke

В этой главе мы рассмотрим очень мощное средство управления контентом сайта — PHP-Nuke. Загрузить последнюю версию PHP-Nuke можно из официального сайта разработчиков `phpnuke.org`.

PHP-Nuke — это мощное средство управления контентом (содержимым) web-сайта. Ему под силу:

- Создавать и размещать новости (статьи) и голосования.
- Создавать резервную копию базы данных.
- Управлять баннерами.
- Управлять контентом, а также внешним видом сайта, включая управление блоками (что это такое, рассмотрим чуть позже).
- Создавать мемберские (закрытые) зоны.
- Управлять учетными записями пользователей.
- Создавать форумы (кстати, в качестве форума по умолчанию используется уже рассмотренный ранее форум PHPBB, поэтому не будем останавливаться на нем подробно).
- Создавать рассылки и письма новостей (newsletter).
- Создавать сообщения (в том числе SMS-сообщения), которые будут отправлены пользователям сайта.

Модульная конструкция PHP-Nuke позволяет подключать дополнительные модули, к числу стандартных модулей относятся:

- **Home** — обеспечивает переход на домашнюю страницу.
- **AvantGo** — просмотр статей.
- **Downloads** — обеспечивает загрузку на сайт полезных программ, скриптов или другой информации.

- **Feedback** — модуль обратной связи с администрацией сайта, отправляет сообщение администратору сайта, при этом адрес администратора не сообщается.
- **PrivateMessages** — управляет личными сообщениями. Вы можете отправить сообщение пользователю сайта (пользователь должен быть зарегистрирован на этом сайте).
- **Recommend Us** — модуль рекомендации, позволяет отправить ссылку на сайт вашему другу.
- **Search** — поиск статей по ключевому слову, теме, автору, по дате и другим критериям.
- **Statistics** — разнообразная статистика, например, какую операционную систему и какой браузер чаще всего используют пользователи сайта.
- **Stories Archive** — поиск статьи в архиве.
- **Submit News** — модуль отправки новостей. Любой пользователь может опубликовать свою новость на сайте (но только с разрешения администратора).
- **Top10** — лучшая десятка (выводятся списки самых читаемых и самых комментируемых статей).
- **WebLinks** — каталог ссылок.
- **Members List** — список пользователей.
- **WebMail** — Web-интерфейс управления почтой домена.
- **Your account** — управление вашей учетной записью.

Как вы видите, список функций и модулей впечатляет. Позже мы остановимся на некоторых модулях подробнее, а сейчас займемся установкой PHP-Nuke.

24.2. Установка PHP-Nuke

PHP-Nuke полностью бесплатный и доступен на сайте `http://phpnuke.org`. Однако разработчик системы требует, чтобы вы не удаляли баннеры-ссылки на сайт `http://phpnuke.org` — это единственное условие.

Распакуйте содержимое архива `PHP-Nuke-X.X.tar.gz` (X.X — это версия PHP-Nuke, я использую версию 6.5) в корневой каталог вашего сервера, то есть в `$DocumentRoot`. Хотя сейчас я рекомендую вам распаковать его в каталог `$DocumentRoot/phpnuke` — чтобы ничего не нарушить, а потом вы всегда сможете перенести его в корневой каталог.

Создайте базу данных `nuke`. Это можно сделать с помощью какого-нибудь средства управления сервером MySQL или, если у вас есть доступ к консоли сервера (хотя бы по ssh), с помощью команды:

```
mysqladmin create nuke
```

Затем нужно создать таблицы. Запросы находятся в файле `phpnuke/sql/nuke.sql`:

```
mysql nuke < nuke.sql
```

После этого измените права доступа к файлу `phpnuke/html/ultramode.txt`:

```
chmod 666 ultramode.txt
```

Все, теперь можно запустить программу администрирования портала:

```
http://domain.ru/phpnuke/html/admin.php
```

При первом запуске сценарий запросит у вас логин и пароль администратора (рис. 24.1).

Внимание!

Не изменяйте сразу язык интерфейса PHP-Nuke, это нужно сделать только после создания учетной записи администратора!

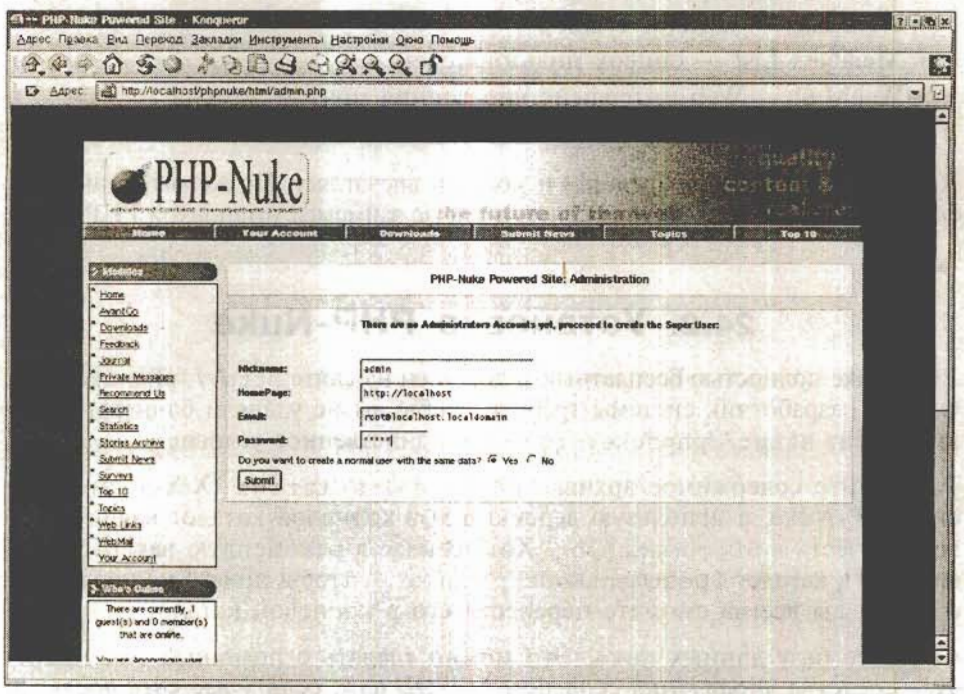


Рис. 24.1. *Первый запуск*

После того, как учетная запись администратора будет добавлена, PHP-Nuke предложит вам зарегистрироваться (см. рис. 24.2). Во избежание подбора пароля по словарю, кроме логина и пароля, вам нужно ввести случайным образом сгенерированное число, которое вы увидите в окне браузера. Данное число запрашивается не только при регистрации администратора, но и при регистрации обычного пользователя.

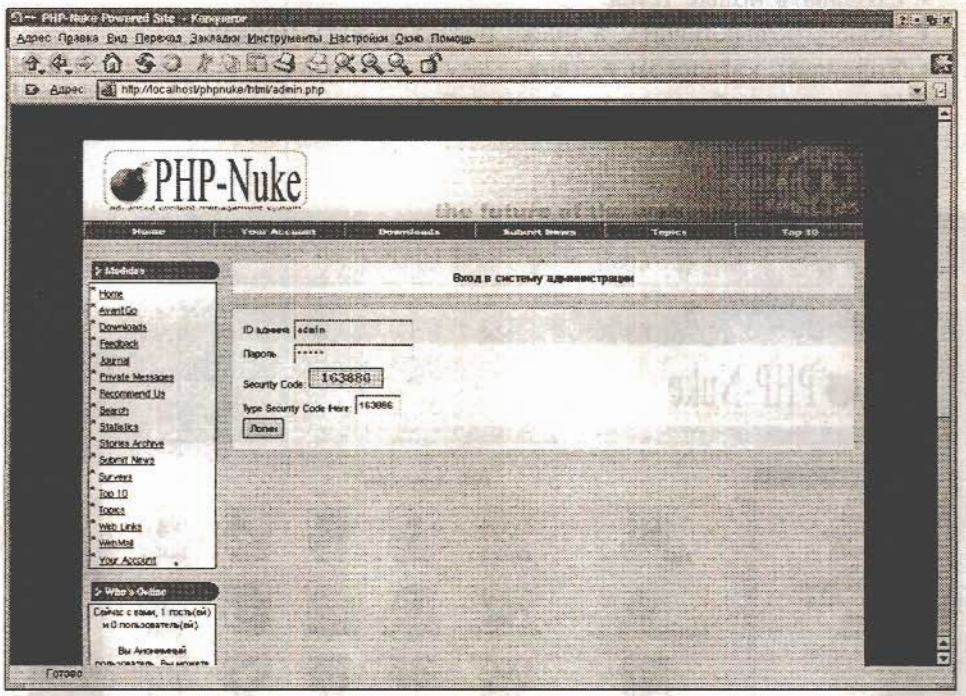


Рис. 24.2. Регистрация в системе

24.3. Возможности администрирования. Настройка сайта

После регистрации вы увидите меню администрации (рис. 24.3), которое позволяет вам:

- Добавить новую статью (новость, блок).
- Сделать резервную копию базы данных.
- Добавить или удалить баннеры и блоки.
- Редактировать контент.
- Управлять загрузками (модулем Downloads).

- Управлять учетными записями администратором и обыкновенных пользователей.
- Администрировать форум PHPBB.
- Управлять каталогами, модулями и сообщениями.
- Создавать рассылки (newsleaaters), редактировать списки подписчиков.
- Управлять опросами (голосованиями).
- Создавать новые темы.
- Включать/выключать, а также редактировать модули.
- Управлять каталогом ссылок.
- Изменять глобальные свойства сайта.
- И многое другое.

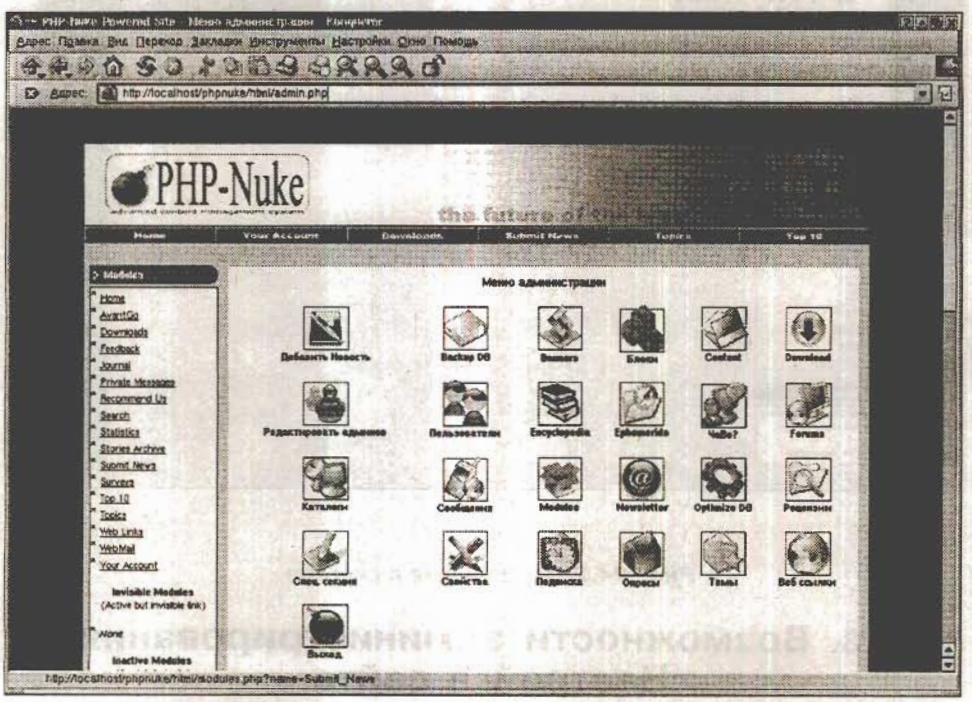


Рис. 24.3. *Меню администрации*

Пример сайта, который поддерживается PHP-Nuke представлен, на рис. 24.4. Обратите внимание на области сайта — Modules, Who's online, Informaton, Login, и др. Данные области называются блоками и именно эти блоки позволяют редактировать элемент меню администрации "Блоки".

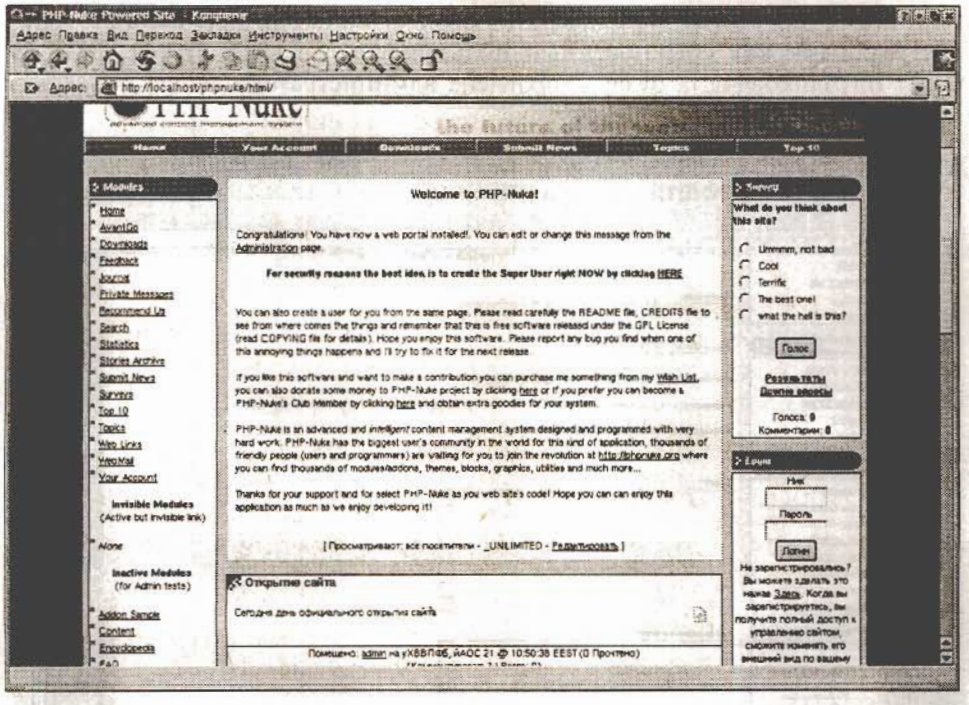


Рис. 24.4. *Пример сайта на PHP-Nuke*

Подробно на каждом элементе меню администрации мы останавливаться не будем — вы и сами разберетесь, что и к чему. Рассмотрим только основные, на мой взгляд, пункты:

- Добавить новость.
- Блоки.
- Контент.
- Редактировать admin.
- Модули.
- Свойства.
- Web-ссылки.

24.4. Управление контентом

Нажмите кнопку «Добавить новость», а потом, используя колесико мыши или полоски прокрутки, пролистайте страницу вниз — там появится форма, позволяющая добавить новую статью (см. рис. 24.5). Введите заголовок, краткий текст (будет отображен на основной странице) и расширенный текст (полный будет отображен при просмотре) статьи, установите ее параметры и нажмите «Ок». В самом низу формы будет еще одна форма — форма голосования (Attach a Poll to this article). Вы можете создать неболь-

шой опрос для статьи, например, понравилась она или нет. Если вы зарегистрированы как обыкновенный пользователь, то ваша статья сразу не будет опубликована, а будет отправлена администратору — для проверки.

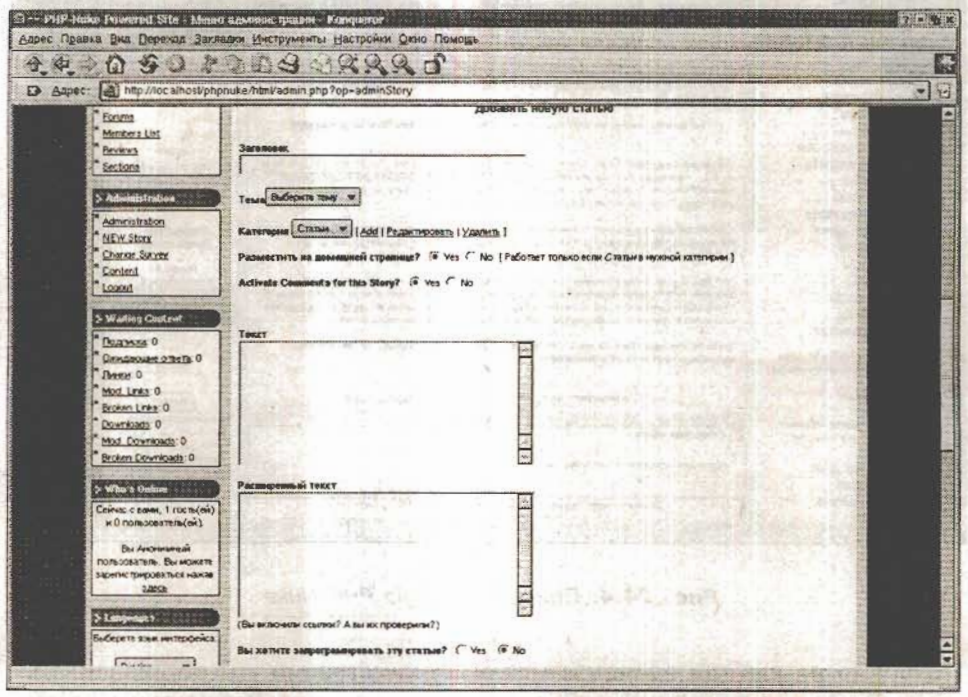


Рис. 24.5. *Добавление статьи*

Пункт **Блоки** позволяет создавать и редактировать блоки (см. рис. 24.6), а также указывать их расположение на странице. Кроме того, можно указать, кто будет видеть тот или иной блок, например, блок **Administration** целесообразно показывать только администраторам, а не всем пользователям.

Основными блоками являются:

- **Modules** — переключение модулей.
- **Administation** — модуль администрирования (не рекомендуется включать из соображений безопасности или отображать только для администраторов).
- **Who's Online** — выводит информацию о количестве зарегистрированных пользователей и гостей, которые в данный момент находятся на сайте.
- **Search** — модуль поиска.
- **Languages** — выбор языка интерфейса.
- **User's Custom Box** — пользовательский блок.

- **Categories Menu** — меню сайта.
- **Survey** — опрос.
- **Login** — модуль регистрации пользователей.
- **Big Story of Today** — лучшая история.
- **Old articles** — старые статьи.
- **Information** — информационный блок.

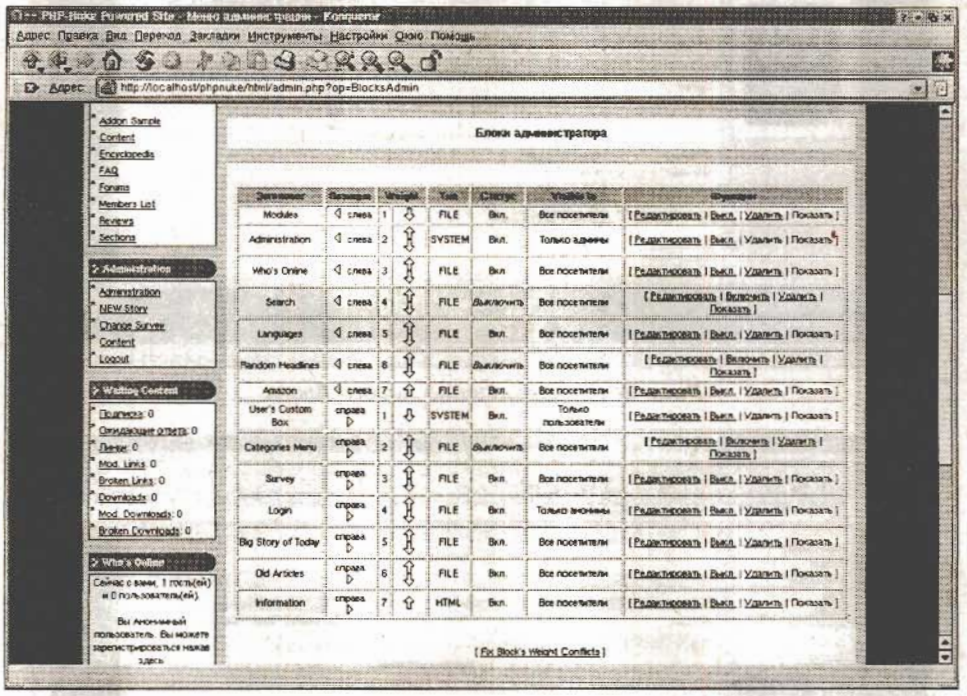


Рис. 24.6. *Блоки*

После списка блоков находится форма добавления нового блока (см. рис. 24.7). При этом нужно указать заголовок, RSS-файл (файл ресурса), текст модуля (если заданный файл недоступен), время обновления блока, а также категорию пользователей, для которой предназначен блок.

Меню **Контент** (рис. 24.8) позволяет:

- Создавать новые категории для страниц.
- Создавать новые страницы.
- Редактировать уже созданные страницы.

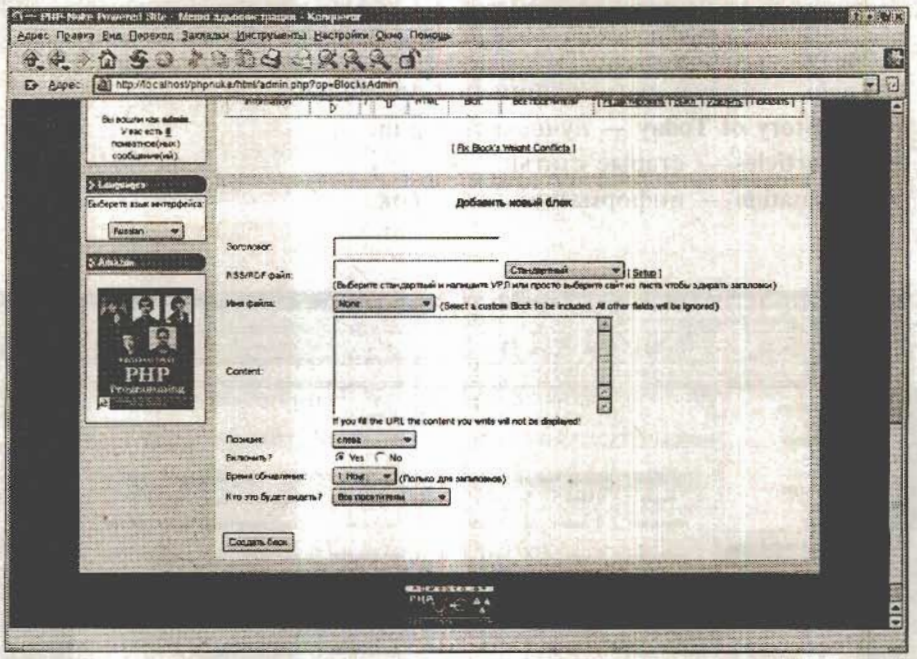


Рис. 24.7. Добавление нового блока

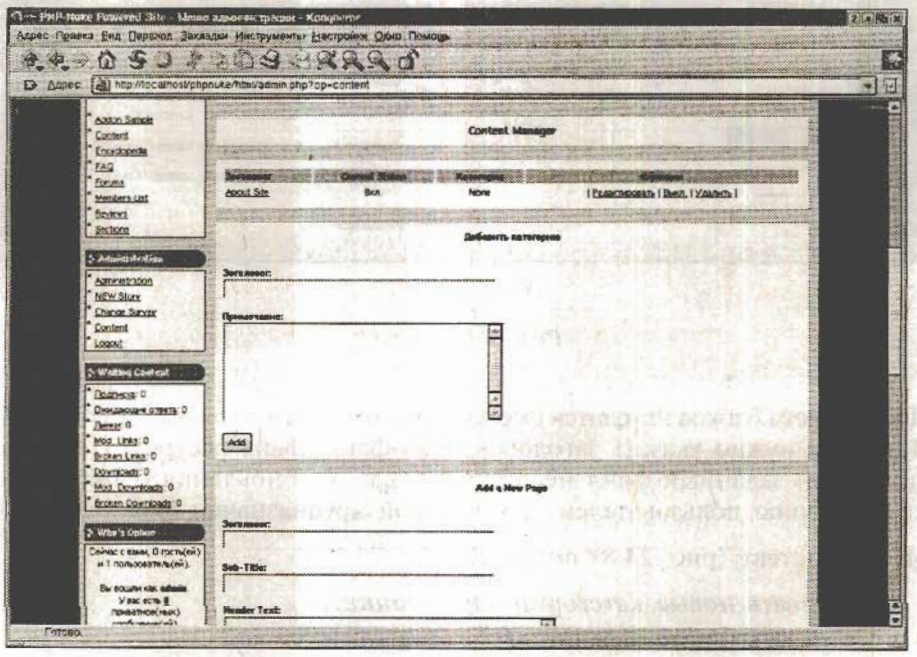


Рис. 24.8. Управление контентом

24.5. Различные настройки

Меню **Добавить нового админа** позволят добавить администратора сайта и установить его полномочия (см. рис. 24.9). А полномочия могут быть следующими:

- **Статьи** — редактирование, создание и удаление статей.
- **Опросы** — редактирование, создание и удаление опросов.
- **Рецензии** — обработка рецензий.
- **Контент** — управления контентом.
- **Темы** — управление темами.
- **Секции** — добавление, изменение и удаление секций.
- **Чаво?** — управление часто задаваемыми вопросами.
- **Newsletter** — создание новостей.
- **Пользователи** — управление учетными записями пользователей.
- **Веб ссылки** — администрирование каталога Web-ссылок.
- **Download** — управление областью загрузки.
- **Форум** — администрирование форума phpBB.
- **Суперпользователь** — пользователю будет предоставлен полный доступ.

Примечание.

Минимальная длина пароля — 5 символов.

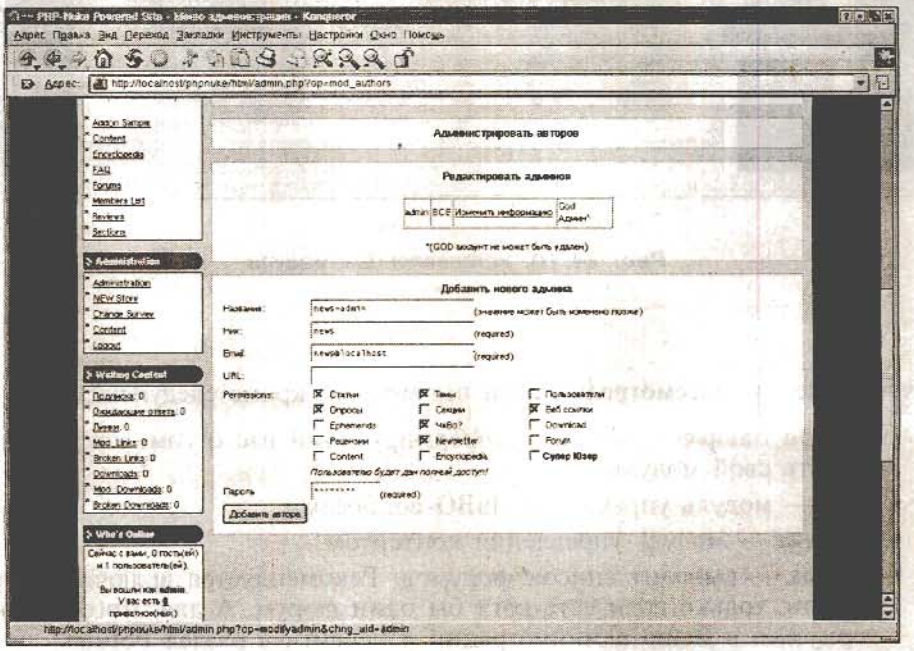


Рис. 24.9. Добавление администратора

Модули играют большую роль в функционировании всей системы PHP-Nuke. Как и блок, модуль можно включить, выключить, определить, кому будет доступен тот или иной модуль, а также поместить модуль на основную страничку (Put In Home). Как выглядит окно работы с модулями, можно увидеть на рис. 24.10.

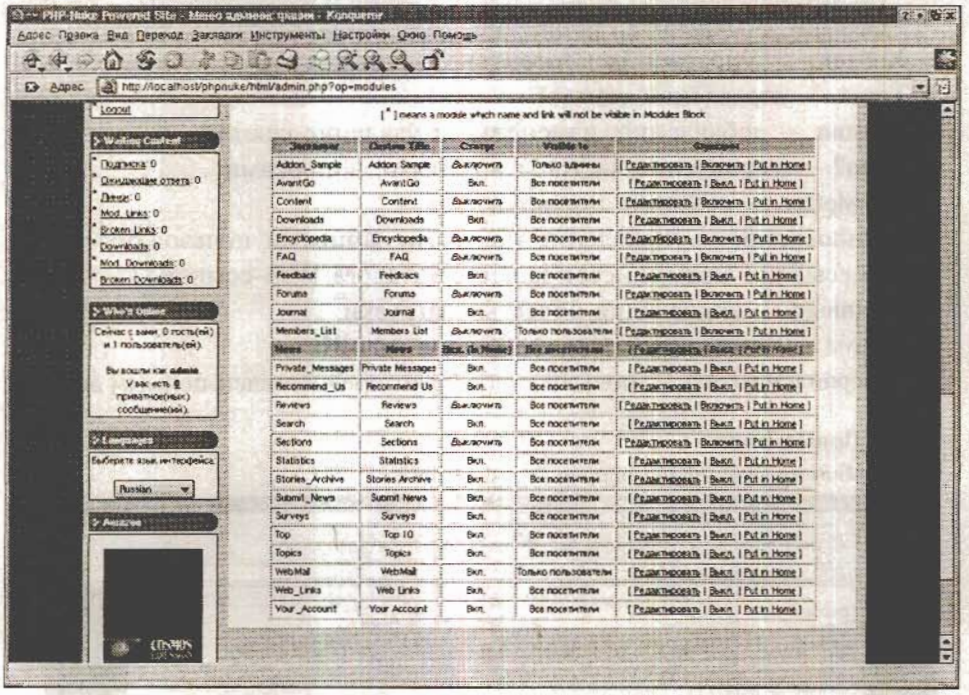


Рис. 24.10. *Установленные модули*

Чуть выше мы рассмотрели почти все модули, кроме следующих:

- **Add On Sample** — модуль, информирующий вас о том, как можно создать свой модуль.
- **FAQ** — модуль управления ЧаВО-вопросами.
- **Контент** — модуль управления контентом.
- **Forums** — выводит список форумов. Рекомендуется включить этот модуль, только если есть хотя бы один форум. А для этого нужно вернуться в меню администрации и перейти в раздел Forums.
- **Reviews** — рецензии.

Меню **Свойства** (рис. 24.11) позволяет изменить глобальные параметры сайта:

- Название.
- URL сайта.
- Логотип.
- Адрес администратора.
- Язык.
- Цветовую тему.
- Параметры почтового интерфейса WebMail (POP/SMTP-серверы и другие параметры).
- Другие параметры.

Опций довольно много, поэтому вам будет с чем разбираться.

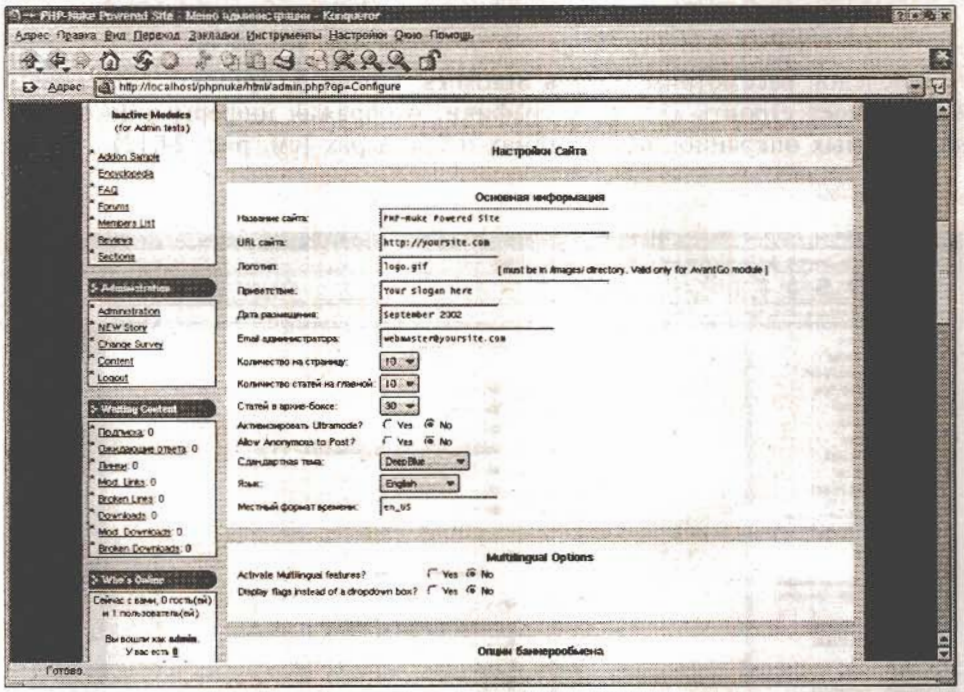


Рис. 24.11. *Глобальные параметры сайта*

Как вы уже успели заметить, PHP-Nuke заменяет сразу несколько отдельных систем:

- Форум (встроенный форум PHPBB).
- Систему обмена почтой (WebMail).
- Систему управления контентом.
- Систему публикации статей.
- Систему рассылки новостей и создания опросов.

Кроме всех этих функций, PHP-Nuke выполняет еще и функции каталога ссылок. Перейдите в меню **Веб ссылки**. Здесь вы можете:

- Очистить базу данных ссылок.
- Добавить категорию ссылок.
- Просмотреть доступные ссылки.
- Просмотреть «битые» ссылки.

Добавить ссылки вы здесь не можете — для этого предназначен модуль Web Links.

Примечание.

Вы заметили, что даже при изменении языка, названия модулей не изменяются. Изменить их можно в разделе **Модули** — просто перейдите на страницу редактирования параметров модуля и введите нужное вам название.

Напоследок рассмотрим модуль **Statistics**, который, кроме сухой статистики, умеет строить красивые графики, отображая информацию об используемых операционных системах и браузерах (см. рис. 24.12).

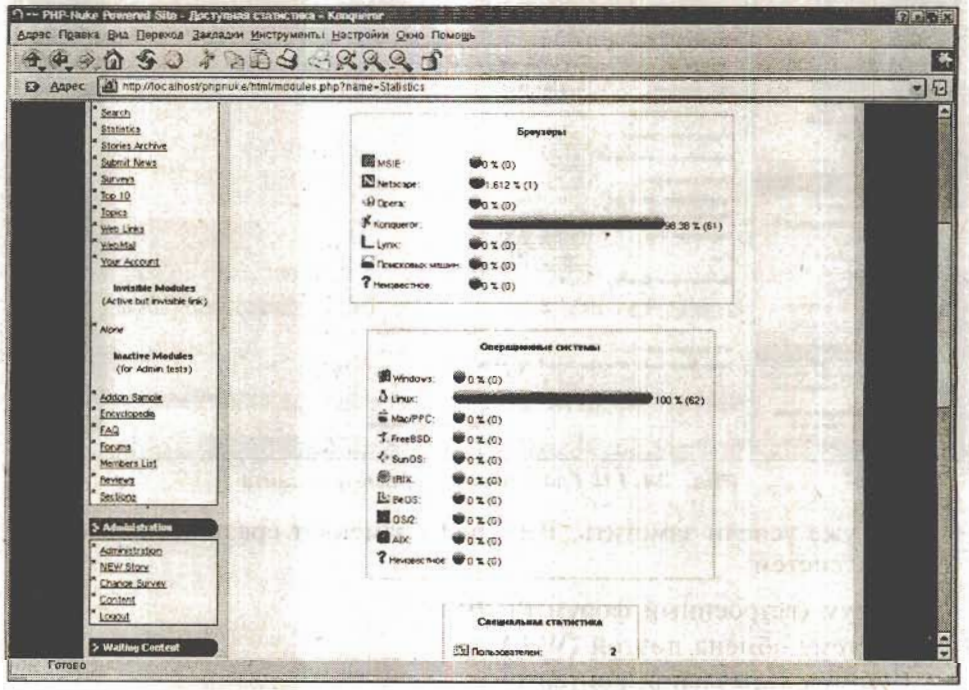
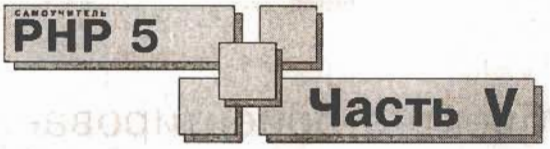


Рис. 24.12. Статистика



PHP для профессионалов

Глава 25. Объектно-ориентированное программирование и PHP

Глава 26. Тонкая настройка интерпретатора PHP

Глава 27. Защищенные соединения

Глава 28. Два проекта в одной главе

Глава 29. Автоматизированное управление параметрами хостинга. Система Monster Controls

Глава 30. Поддержка COM и Java

Глава 25

Объектно-ориентированное программирование и PHP

25.1. Основные принципы и понятия объектно-ориентированного программирования

Введение

Объектно-ориентированное программирование (ООП) – это подход к написанию программ. На каждом этапе развития вычислительной техники появлялся новый способ написания программ. Первые программы создавались с помощью переключателей на передней панели компьютера (сюда же можно отнести перфокарты). Ясное дело, что такой подход не способствовал написанию больших программ. Например, на супер-ЭВМ 40-50-х годов, занимающих целые комнаты, подключение проводов длилось несколько дней, а может и недель, а сам расчет занимал несколько минут или даже секунд.

Точно такая же ситуация сложилась и с перфокартами. В то время пользователи не работали с компьютером непосредственно: они писали программу отдельно, затем оператор-машинистка «записывала» программу на перфокарту, которая передавалась оператору вычислительного отдела. Через определенное время оператор возвращал пользователю результат работы программы — эдакий несколькометровый рулон. Представьте, что будет, если пользователь (программист) допустил ошибку? Вся эта цепочка повторялась заново.

Затем появился язык ассемблера, позволяющий писать более длинные программы. Однако требования к программным продуктам росли, что стало основной причиной появления языков программирования высокого уровня. Первым языком высокого уровня стал Фортран (1950). Такие языки были существенно проще ассемблера и позволяли разрабатывать программы за короткое время.

Однако по мере увеличения исходного кода программы, написанные на языке высокого уровня, становились нечитабельными из-за отсутствия четко выделенной структуры программы. Этот факт послужил предпосылкой для создания структурного программирования (Алгол, Pascal, C). Структурное программирование подразумевает:

- Наличие четкой структуры программы и программных блоков.
- Отсутствие инструкций безусловного перехода GOTO (JMP) или минимальное их использование.

Следующий этап в развитии языков программирования высокого уровня — процедурное программирование, подразумевающее использование подпрограмм с поддержкой локальных переменных и рекурсии. Подпрограммы позволяют сократить исходный текст программы, например, когда одна последовательность операторов должна выполняться в программе несколько раз. Тогда эта последовательность записывалась в виде подпрограммы, которая вызывалась в нужный момент из основной программы.

Все это существенно облегчило жизнь программистов, позволяя создавать не только большие, но и читабельные программы.

Со временем возникала та же проблема — выросла длина исходного кода программы и они опять стали нечитабельными. Чтобы написать сложную программу, был нужен новый подход к программированию. Таким подходом стало объектно-ориентированное программирование.

ООП основано на «трех китах»:

- Инкапсуляции.
- Полиморфизме.
- Наследовании.

Инкапсуляция (encapsulation)

Инкапсуляция — это механизм, объединяющий данные и обрабатывающий их код как единое целое. Также инкапсуляция защищает и то, и другое от вмешательства извне.

Класс — это базовое понятие в (ООП). Попросту говоря, класс — это своеобразный тип переменной. Экземпляр класса (переменная типа класс) называется объектом. Объект — это совокупность данных (свойств) и функций (методов) для их обработки. И данные, и методы называются членами класса. С точки зрения «первого кита», объектом можно назвать все, что поддерживает инкапсуляцию.

Внутри объекта данные и коды (члены класса) могут быть либо открытыми, либо закрытыми. Первые могут быть доступны для других частей программы, которые не являются частью объекта. Закрытые доступны только внутри этого объекта. К сожалению, PHP не поддерживает ин-

капсуляцию полностью: он может только объединять данные и код, но не поддерживает открытые и закрытые члены класса. В отличие от C++, поддержка ООП в PHP довольно ограничена, поэтому многое, что должно работать в C++, в PHP работать не будет.

Полиморфизм (polymorphism)

Полиморфизм (от греч. polymorphos) позволяет одно и то же имя использовать для похожих, но технически разных задач. Язык C не является объектно-ориентированным, поэтому он не поддерживает полиморфизм. Как следствие, для вычисления абсолютного значения чисел разных типов нужно использовать три разных функции: **abs()**, **fabs()**, **labs()**. В C++, который поддерживает полиморфизм, каждая из этих функций может быть названа **abs()**. Данное явление называется перегрузкой функций.

Главным в полиморфизме является то, что он позволяет нам манипулировать с объектами путем создания стандартного интерфейса для схожих действий. Полиморфизм значительно облегчает написание сложных программ.

Наследование (inheritance)

Наследование позволяет одному объекту приобретать свойства другого. Не нужно путать наследование с копированием объекта. При копировании создается точная копия объекта, а при наследовании эта точная копия дополняется уникальными свойствами, которые характерны только для производного объекта.

Прекрасно понимаю, что если вы никогда не сталкивались с ООП, это краткое описание основных механизмов ООП вам ничего не скажет. Тем не менее, в этой книге я попытаюсь описать работу с объектами так, чтобы ваших знаний было достаточно для написания собственных объектов и работы с ними. К тому же, вследствие отсутствия полной поддержки ООП в PHP, вам не нужны глубокие знания объектно-ориентированного программирования. Если вы все же заинтересовались, рекомендую приобрести какую-нибудь книгу по C++.

25.2. Создание класса в PHP

Напомню, что класс — это своеобразный тип переменной, а объект — это экземпляр класса, то есть переменная этого типа. Описание класса в PHP начинается служебным словом **class**:

```
class Имя_класса {

// описание членов класса - данных и методов для их обработки

}
```

Для объявления объекта нужно использовать оператор **new**:

```
Объект = new Имя_класса
```

Данные описываются с помощью служебного слова **var**. Метод описывает так же, как и обыкновенная функция. Методу также можно передавать параметры. Пример класса приведен в листинге 25.1.

Листинг 25.1. Пример класса

```
class Human {
// данные (свойства)
var $Name;
var $age;

// методы
function GetName()
{
echo "Ivan\n";
}

...

// создаем объект класса Human
$obj = new Human;
```

25.3. Доступ к методам и свойствам объекта

Описав класс и создав объект, мы хотим получить доступ к его членам. Для этого в PHP используется оператор ->:

```
$obj = new Human;

$obj->Name = "Denis"; // устанавливаем свойство Name равным Denis
echo $obj->Name;    // выводит "Denis"
$obj->GetName();    // выводит "Ivan"
```

Обратите внимание на вывод метода `GetName()`: `"Ivan"`. Но ведь мы установили свойство `Name` равным `"Denis"`. Очевидно, чтобы метод `GetName()` (см. листинг 25.1) делал то, что нам нужно, необходимо вывести свойство `Name`. Мы не можем написать в середине `GetName()` просто **echo $Name**. Чтобы получить доступ к члену класса внутри класса, нужно использовать указатель `$this`, который всегда знает, к какому объекту он относится. Вот модифицированный метод `GetName()`:

```
function GetName()
{
echo $this->Name.''\n'';
}
```

Аналогично, можно написать метод SetName():

```
function SetName($name)
{
$this->Name = $name;
}
```

Теперь для изменения имени можно использовать метод SetName():

```
$obj->SetName("Denis");    // аналогично $obj->Name = "Denis";
$obj->GetName();       // аналогично echo $obj->Name;
```

Указатель **$this** используется не только для доступа к данным, но и для доступа к методам:

```
function SetName($name)
{
$this->Name = $name;
$this->GetName();
}
```

25.4. Инициализация объектов

Иногда нам нужно выполнить инициализацию объекта — присвоить его свойствам первоначальные значения. Допустим, наш класс называется Human и содержит два свойства: имя человека и его возраст (в днях).

Можно написать функцию, которая будет выполнять инициализацию объекта, например, **Initialize**:

```
function Initialize($name)
{
$this->Name = $name;
$this->age = 1;
}
```

Тогда для инициализации объекта ее нужно будет вызвать сразу после его создания:

```
$obj = new Human;
$obj->Initialize();
```

Но ведь вы (или кто-то другой) можете забыть это сделать. Или вызвать какой-нибудь метод между созданием (оператор **new**) объекта и его инициализацией (вызовом **Initialize**). Последствия могут быть непредсказуемыми, если объект непроинициализирован перед использованием. Конечно, в нашем случае ничего страшного не случится, но, если вы создаете какой-нибудь серьезный проект, с инициализацией объекта лучше не шутить.

Для того, чтобы PHP знал, что определенный метод нужно вызывать автоматически при создании объекта, ему нужно дать имя такое же, как и у класса (Human):

```
function Human ($name)
{
$this->Name = $name;
$this->age = 1;
}
```

Метод, инициализирующий объект, называется конструктором. Деструктор выполняет действия, обратные конструктору — освобождает используемые ресурсы: закрывает файлы, соединения, сокеты, и т.д. Из-за особенностей самого PHP в нем нет деструкторов: используемые сценарием ресурсы освобождаются автоматически при завершении работы. Если же вам все-таки нужно выполнить какие-нибудь специфические действия, создайте метод и вызывайте его явно, когда вам нужно.

25.5. Еще раз о наследовании и полиморфизме

Как уже отмечалось выше, наследование — это не просто создание точной копии класса, а расширение уже существующего класса, чтобы потомок мог выполнять какие-нибудь новые, характерные только ему функции.

Рассмотрим следующий пример (см. листинг 25.2).

Листинг 25.2. Наследование классов

```
class Parent {
function parent_func() { echo "Parent function"; }
function Test() { echo "This is Parent class"; }
}

class Child extends Parent {
function child_func() { echo "Child function"; }
function Test() { echo "This is Child class"; }
}
```

Ключевое слово **extends** говорит о том, что класс `Child` наследует все методы и свойства класса `Parent`. Если объявлен объект `$obj` класса `Child`, то для него справедливы вызовы:

```
$obj->parent_func();  // выведет Parent function
$obj->child_func();     // выведет Child function
$obj->Test();   // выведет This is Child class
```

Родительский класс обычно называют базовым классом или суперклассом, а класс `Child` — производным или подклассом.

Теперь рассмотрим пример, посвященный полиморфизму. **Полиморфизм — это свойства базового класса использовать функции производных классов.** Следующий пример (см. листинг 25.3), я бы называл его «полиморфическим», демонстрирует все преимущества полиморфизма.

Листинг 25.3. Полиморфизм

```
class Figure {
function DrawSelf() { echo "Рисуем абстрактную фигуру"; }
function Draw() { $this->DrawSelf(); }
}

class Rectangle extends Figure {
function DrawSelf() { echo  "Рисуем прямоугольник"; }
}

$a = new Figure();
$b = new Rectangle();

$a->Draw();
$b->DrawSelf();
$b->Draw();
```

Предположим, что класс `Figure()` определяет какую-нибудь абстрактную геометрическую фигуру — мы еще не знаем, что это за фигура: квадрат, эллипс или треугольник. Функция `DrawSelf()` рисует эту фигуру. Конечно, рисовать мы ничего не станем, а просто выведем на экран наши намерения что-нибудь нарисовать. Функция `Draw()` — это просто ссылка на функцию `DrawSelf()`.

Производный класс `Rectangle()` также обладает функцией `DrawSelf()`, но эта функция рисует не произвольную фигуру, а прямоугольник. Эта функция не имеет параметров, но, как мы уже знаем, мы можем указывать произвольное число параметров для функции, не определяя их явно.

Вот теперь проанализируем вывод сценария:

```
Рисуем абстрактную фигуру
Рисуем прямоугольник
Рисуем прямоугольник
```

Как вы видите, при вызове функции `Draw()` класса `Rectangle()` будет нарисован прямоугольник, а не неопределенная фигура. Вот это и есть проявление полиморфизма: функция `Draw()` класс `Figure()` была перезаписана одноименной функцией класса Rectangle(). Функция, переопределенная таким образом, называется виртуальной.

25.6. Пример класса

Попытаемся создать простой класс отправки сообщений, который основывается на стандартной функции **Mail()**.

Класс **EMail** обладает такими свойствами и методами:

- **Свойство From** — в нем содержится информация об отправителе.
- **Свойство Reply** — содержит обратный адрес отправителя.
- **Свойство To** — адрес адресата.
- **Свойство Subject** — тема сообщения.
- **Свойство Message** — само сообщение.
- **Свойства enc_from, enc_to** — содержат, соответственно, кодировку, из которой нужно преобразовать текст сообщения и заголовки, и конечную кодировку. Если эти свойства равны, преобразование не осуществляется. Значение по умолчанию — "k" (KOI8-R).
- **Метод EMail (конструктор)** — инициализирует объект и устанавливает информацию об отправителе (свойства From и Reply).
- **Метод LoadFromFile** — загружает сообщение из указанного файла. Можно напрямую присвоить текст сообщения свойству Message, но намного удобнее загрузить его из файла.
- **Метод MailSend()** — осуществляет отправку сообщения.

Текст класса приведен в листинге 25.4. Вслед за листингом показан пример его применения.

Листинг 25.4. Класс EMail

```
<?
class EMail {

var $From = ''PHP Simple Mail class'';
var $Reply = ''den@localhost'';
// var $Reply = $GLOBALS['USER''].''@''.$GLOBALS['HOSTNAME''];

var $To = '''';
var $Subject = '''';
var $Message = '''';
var $enc_from = ''k'';
var $enc_to = ''k'';

function EMail($from, $reply)
{
// конструктор устанавливает поля From и Reply-to
  $this->From = $from;
  $this->Reply = $reply;
}
```

```
function LoadFromFile($filename)
{
// Загружаем сообщение из файла
   $this->Message = join(''', file($filename) );
}

function MailSend()
{

// Пытаемся преобразовать кодировку, если нужно
if ($this->enc_from !== $this->enc_to)
 {
   // Дополнительные переменные $fc и $tc я использовал для
   // улучшения читаемости кода. Вы можете писать
   // сразу $this->enc_from
   // и $this->enc_to в вызове функции convert_cyr_string
$fc = $this->enc_from;
$tc = $this->enc_to;

$this->Message = convert_cyr_string($this->Message, $fc, $tc);
$this->From = convert_cyr_string($this->From, $fc, $tc);
$this->Subject = convert_cyr_string($this->Subject, $fc, $tc);
 }

// Отправляем сообщение
 return mail($this->To,$this->Subject,$this->Message,''From:
$this->From\nReply-to: $this->Reply\n'');

}        // end of MailSend()

}        // end of class


$m1 = new EMail(''Denis'',''den@localhost'');

// Кому
$m1->To = ''evg'';    // пользователь локальной системы
// Тема
$m1->Subject = ''Hello'';
// Загружаем сообщение
$m1->LoadFromFile(''/home/den/message.txt'');

// Отправляем сообщение
echo $m1->MailSend();

?>
```

Как использовать класс EMail, думаю, ясно. Сначала нужно создать экземпляр этого класса, то есть объект:

```
$m1 = new EMail("Denis'',''den@localhost'');
```

Затем нужно указать пользователя, кому предназначено сообщение, тему сообщения и загрузить текст самого сообщения:

```
$m1->To = ''evg'';
$m1->Subject = ''Hello'';
$m1->LoadFromFile(''/home/den/message.txt'');
```

После всего этого остается только отправить сообщение:

```
echo $m1->MailSend();
```

Если вы хотите отправить еще одно сообщение, создавать объект класса EMail не обязательно, вы можете использовать уже созданный объект `$m1`. Последовательность работы с объектом та же: указываем адрес, тему, текст сообщения и отправляем сообщение, вызвав метод `MailSend()`.

25.7. ООП и PHP 5

Сердцем PHP 4 является Zend Engine 1.0 — это его движок. Движок Zend можно сравнить с двигателем автомобиля — иногда возможности автомобиля ограничены его двигателем. Это сравнение как нельзя лучше подходит для PHP 4 и Zend 1.0. Из-за внутренних ограничений Zend поддержка ООП в PHP 4 довольно ограничена. Например, вы не можете написать выражение $foo->bar()->value=0; которое является корректным для Java и COM (Component Object Model). Также довольно низка производительность механизма перегрузки — опять же таки из-за внутренних ограничений Zend 1.0.

С появлением Zend Engine 2.0, который лежит в основе PHP 5, все ООП-ограничения сняты и теперь PHP 5 с чистой совестью можно назвать языком программирования с полной поддержкой ООП.

В PHP 5 появилось много изменений, касающихся ООП. Только нужно помнить, что при использовании новых возможностей теряется совместимость с PHP 4.

Рассмотрим одну из наиболее важных, на мой взгляд, возможностей — обработка исключительных ситуаций. В четвертой (и более ранних) версии PHP не было никакого механизма обработки исключений. Теперь этот механизм и его синтаксис похож на синтаксис языка C++:

```
try {
   …код
   if (failure) {
      throw new MyException(''Сбой'');
```

```
    }
  ...код
    }
catch ($exception) {
  ... обработка исключения
  throw $exception; // повторная генерация исключения
    }
```

Если вы незнакомы с C++, вышеприведенный код ничего вам не сказал. Попробуем в нем разобраться. Блок try называют блоком обработки исключительной ситуации, иногда его еще называют пробным блоком и блоком возбуждения исключительной ситуации.

Предположим, что у нас есть класс, обладающий свойствами `$a`, `$b` и `$res`. `$res` — это результат от деления `$a/$b`. При делении `$a/$b` может возникнуть исключительная ситуация — деление на ноль, если $b = 0. Напишем обработчик этой ситуации:

```
try {
if ($this->$b==0) throw ''EDivideZero'';
$this->$res = $this->$a/$this->$b;
}
catch ($exception) {
if ($exception===''EDivideZero'') die(''Ошибка: ДЕЛЕНИЕ НА
НОЛЬ'');
else die($exeption);
}
```

В блок try нужно заключить потенциально опасные операции — та же операция деления является таковой, потому что в результате может возникнуть исключительная ситуация. В блоке try должен быть один или несколько операторов throw, которые вызывают исключительную ситуацию. В качестве параметра оператора throw нужно передать имя исключительно ситуации. Далее, в блоке catch, выполняется обработка исключительной ситуации.

Как видно из листинга, если $b==0, то мы вызываем исключительную ситуацию EDivideZero. Обработчик catch обрабатывает эту ситуацию: если это EDivideZero, то выводится строка «Ошибка: ДЕЛЕНИЕ НА НОЛЬ» и работа сценария прекращается. Если же возникла другая исключительная ситуация (в этом примере мы другой ситуации не предусмотрели), просто будет выведено ее имя.

Более подробно о возможностях PHP 5 и Zend 2.0 вы можете прочитать в главе 31, а также по адресам:

- http://vseti.ru/articles/web_dev/php5/
- http://www.zend.com/engine2/zendengine-2.0.pdf

Глава 26

Тонкая настройка интерпретатора PHP

26.1. Директивы файла конфигурации php.ini

Общий формат файла php.ini

Все параметры интерпретатора PHP задаются в файле `php.ini`. Если вы используете операционную систему Linux, то этот файл находится в каталоге `/etc`, а если Windows, то вы найдете `php.ini` в каталоге, в который вы установили PHP.

Если PHP установлен как модуль Apache (в большинстве случаев так оно и есть, при условии, что вы работаете под Linux), настройки интерпретатора можно задавать в файле `httpd.conf` или файлах `.htaccess`. В этом случае перед именем каждого параметра нужно указывать префикс **php_** и указывать значение через пробел, без использования знака равенства.

Параметры в файле `php.ini` указываются в виде:

```
параметр = значение
```

Причем в одной строке может быть только один параметр.

Пример объявления директивы **short_open_tags**:

В `php.ini`:

```
short_open_tags = On
```

В `httpd.conf` или файле `.htaccess`:

```
php_short_open_tags On
```

Сейчас мы рассмотрим только самые важные директивы интерпретатора, описание всех остальных директив вы найдете в приложении.

Настройка обработки ошибок. Директива error_reporting

Одной из самых важных директив является директива **error_reporting**. Мы ее уже рассматривали в этой книге, но я думаю ничего страшного не случится, если мы освежим свои знания.

Параметр **error_reporting** в файле конфигурации PHP — `/etc/php.ini` — позволяет определить степень обработки и протоколирования ошибок. Возможные значения:

- E_ALL — все ошибки и предупреждения.
- E_ERROR — фатальные ошибки во время выполнения программы (например, невозможно открыть файл).
- E_WARNING — предупреждения времени выполнения (нефатальные ошибки).
- E_PARSE — синтаксические ошибки.
- E_NOTICE — замечания времени выполнения (например, сообщение о том, что переменная неинициализирована).

Кроме вышеперечисленных возможны и другие значения, но они не столь важны. Я рекомендую включить оповещения обо всех ошибках и предупреждениях:

```
error_reporting = E_ALL
```

Когда программа отлажена, можно отключить замечания времени выполнения. Для отключения какого-нибудь вида ошибок используется тильда:

```
error_reporting  =  E_ALL & ~E_NOTICE
```

Максимальное время работы сценария. Директива max_execution_time

Директива **max_execution_time** устанавливает (в секундах) максимальное время работы сценария. Ограничение времени рекомендуется использовать, если вы пишите «прожорливые» сценарии, которые «съедают» много процессорного времени, что может отрицательно повлиять на работу всей системы. Обычно сценарию нужно не более трех-четырех минут (это время может быть и больше — в зависимости от поставленных перед сценарием задач), чтобы обработать введенные пользователем данные и выдать результат:

```
max_execution_time = 240
```

«Вредная» директива magic_quotes_gpc

Нельзя не отметить очень «вредную» директиву **magic_quotes_gpc**. Если ее включить, то интерпретатор будет дополнять символы ', ", \,(0) дополнительным слэшем. Почему эта директива вредная? Да потому, что

она «размножит» каждую кавычку, если например, в текстовом поле будет одна кавычка, то с каждым запуском сценария, количество кавычек будет увеличено на одну. Включение этой директивы принесет гораздо больше проблем, чем выгоды. Самое интересное в том, что, когда возникает данная проблема, о директиве **magic_quotes_gpc** вспоминаешь в самую последнюю очередь. Поэтому я вам рекомендую ее отключать.

```
magic_quotes_gpc = off
```

Автоматическое помещение передаваемых сценарию переменных в глобальные массивы

Очень полезной является директива **track_vars**. Если она включена, то все переменные, поставляемые вместе с GET- и POST-данными и Cookies, будут помещены в глобальные массивы $HTTP_GET_VARS, $HTTP_POST_VARS и $HTTP_COOKIE_VARS, соответственно.

```
track_vars = On
```

26.2. Индивидуальная настройка обработки ошибок для конкретного сценария

Если вы не можете отредактировать файл `php.ini` (например, у вас нет на это прав), можно использовать вызов **error_reporting()**, например:

```
error_reporting(E_ALL);
```

Следует учитывать, что действие директивы **error_reporting** в файле конфигурации распространяется на все ваши сценарии, а вызов **error_reporting()** — только на текущий сценарий.

Существует еще один способ избежать вывода сообщения об ошибке — это использование оператора @. Например, если вы не хотите, чтобы пользователь видел сообщение об ошибке какой-нибудь стандартной функции, нужно указать оператор @ перед именем функции, например:

```
@mysql_query(''create table tbl (no int, info text)'');
```

В случае, если ваша программа породила ошибку, нужно принудительно ее завершить, предварительно выполнив какие-нибудь действия по обработке ошибки. Как мы уже знаем, для принудительного завершения работы используется функция **exit()**. Она просто завершает работу сценария:

```
if(!mysql_connect($SERVER,$USER,$PASSWD))
{
echo ''Не могу подключиться к серверу\n'';
exit;
}
```

Обычно намного удобнее использовать функцию **die()**, которая также завершает работу сценария, но перед ее завершением выводит указанное сообщение:

```
$link = mysql_connect($SERVER,$USER,$PASSWD) or
die(''Ошибка\n'');
```

Согласитесь, намного удобнее, чем предложенный выше вариант.

26.3. Синтаксическая проверка выполнения кода

Проверка без запуска кода

Функция **create_function()**, по идее, используется для создания функции с уникальным именем. На практике это позволяет проверить, является ли некоторый текст верным PHP-кодом, не запуская сам код. При этом функция будет пытаться создать функцию, телом которой будет указанный текст. И если ей это удастся (компилятор сможет это сделать), то значит код синтаксически верен.

Функции **create_function()** нужно передать два параметра — аргументы и код функции, который представляет собою программу на PHP. Создадим функцию сложения двух чисел:

```
$Add = create_function('$a, $b', 'return $a+$b;');
echo $Add(2,2);          // выведет 4
```

Аргументы и код должны заключаться в одинарные кавычки, иначе PHP попытается выполнить трансляцию переменных, то есть подставить вместо `$a` и `$b` их истинное значение, а если таких переменных нет, то мы получим сообщение об ошибке.

С помощью функции **create_function()** мы можем проверить синтаксическую корректность кода:

```
$script = "script.php";
$code = join("",File($script));

if (create_function("","?>$code<?"))
 echo "Нет ошибок";
else echo "Есть ошибки";
```

В этом примере сначала получаем содержимое файла `script.php` в одну большую строку `$code`, а затем передаем ее функции **create_function()**. Сейчас мы используем двойные кавычки, поскольку строка `$code` содержит код программы. Если бы мы использовали одинарные кавычки, то функции **create_function()** было бы передано выражение "`$code`", а не код программы.

Обратите внимание! Перед текстом программы мы поставили тег **?>** , а после — тег **<?**. Это сделано для того, чтобы функция настроилась на правильную трансляцию кода: ведь она ожидает *код*, а мы передаем ей *документ PHP*, содержащий теги ввода/вывода. Мы можем не указывать теги **?>** и **<?**, но тогда мы должны быть уверенными в том, что переменная `$code` не содержит этих тегов, то есть является «чистой» PHP-функцией без вставки HTML.

Проверка с запуском кода

Функция **eval (string $code)** выполняет переданный ей код. При этом в качестве кода программы она воспринимает параметр `$code`. Его-то она и запускает. Например:

```
eval('echo 100');   // выведет 100
```

Переменная `$code` может содержать:

- Теги <? и ?>.
- Объявления и вызовы функций.
- Управляющие структуры.
- Вызовы функции **eval()**.

Однако есть некоторые ограничения при использовании этой функции:

- Любая критическая ошибка приведет к завершению *основного* сценария: мы не можем перехватить ошибки, порождаемые переданным кодом функции **eval()**.
- Синтаксические ошибки и предупреждения, порождаемые кодом `$code`, не приведут к завершению работы основного сценария, но функция возвратит false.

При использовании функции **eval()** нужно помнить, что код `$code` не локализируется, то есть этот код будет использовать те же глобальные переменные, что и основная программа. Также нужно помнить о том, что если код содержит оператор **return**, возвращающий какое-то значение, то функция **eval()** возвратит именно это значение.

Напишем с помощью функции **eval()** аналог директивы **include**:

```
$code = join("",file("file.php"));
eval("?>$code<?);
```

Как и в случае с **create_function()**, нужно использовать теги **?>** и **<?**, чтобы обеспечить правильную трансляцию кода.

Вы можете спросить: зачем использовать функцию **eval()**, если вместо вызова функции:

```
eval('$a=10; echo $a;');
```

мы можем просто написать:

```
$a=10; echo $a;
```

а вместо рассмотренного выше аналога **include** использовать саму директиву **include**?

Дело в том, что передаваемый функции **eval()** код не является статическим — он динамический, то есть генерируется или изменяется в процессе работы сценария. Поэтому функция **eval()** просто незаменима.

Предположим, что мы хотим создать семейство функций cosN, где N — это номер от 1 до 100. Тогда нам достаточно написать следующий код:

```
for($i=1; $i<101; $i++)
 eval("function cos$i() { return cos($i); }");
```

После выполнения данного кода будет создано 100 функций: cos1() ... cos100(). Вы можете вызвать любую из них:

```
cos1();
cos100();
```

26.4. Финализаторы. Функция Register_shutdown_function()

Функция **Register_shutdown_function()** предназначена для регистрации так называемой функции завершения, которая будет вызываться, когда сценарий завершит свою работу. Причем эта функция будет вызвана вне зависимости, как завершил сценарий работу: нормально или по ошибке.

Для регистрации такой функции нужно написать саму функцию, а затем передать функции **Register_shutdown_function()** ее имя. Напишем функцию, которая будет сбрасывать данные файлового буфера на диск при завершении работы сценария:

```
function Shutdown()
{
flush($file);
}

Register_shutdown_function("Shutdown");
```

Благодаря такому подходу функция **Shutdown()** будет вызвана:

- при завершении работы сценария (перед возвратом из сценария);
- при вызове функций **exit()** или **die()**;
- при возникновении фатальной ошибки, например, при синтаксической ошибке.

Можно зарегистрировать несколько таких функций — они будут выполнены в порядке их регистрации.

При использовании финализаторов нужно помнить, что функция-финализатор вызывается уже после закрытия соединения с браузером клиента, поэтому, если функция-финализатор выводит информацию с помощью echo, клиент ничего не увидит, так как соединение уже закрыто. Правда, есть одно но: если у вас CGI-версия PHP (то есть PHP запускается как CGI-приложение для обработки файлов с расширением .php, а не как модуль Apache), то функция echo будет работать так, как от нее и ожидают:

```
<?
echo "Сценарий запущен";

function Shutdown()
{
echo "Это сообщение вы не увидите, если у вас PHP работает как
модуль Apache\n";
}

register_shutdown_function("Shutdown");

?>
```

26.5. Информационные функции

К информационным функциям относятся следующие три функции:

phpinfo();
phpversion();
getlastmod().

Функция phpinfo()

Функция **phpinfo()** возвращает детальную информацию об интерпретаторе:

- Версию PHP.
- Версию операционной системы.
- Параметры, заданные при сборке PHP.
- Дату сборки интерпретатора.
- Сведения о дополнительных модулях.
- Переменные окружения — основные и локальные настройки интерпретатора.
- Заголовки HTTP.
- Лицензию PHP.

Как правило, данная функция используется один-единственный раз: при установке интерпретатора для проверки его работоспособности. С выводом этой функции вы уже наверняка встречались.

Функция phpversion()

Функция **phpversion()** просто возвращает версию интерпретатора. Например:

```
PHP Version 4.3.2
```

Функция getlastmod()

Функция **getlastmod()** очень полезна — она возвращает время последнего изменения сценария. Использовать ее можно примерно так:

```
echo date("d.m.Y", getlastmod());
```

26.6. «Засыпание» сценария

Функция **sleep()** прерывает выполнение сценария на указанное время. После этого времени работа сценария будет возобновлена. Время задается в секундах.

Функция **usleep()** аналогична функции **sleep()**, но время задается в микросекундах:

```
sleep(60);
sleep(60000);
```

Для чего нужно «засыпание» сценария? Например, для ожидания какого-нибудь события, например, установления соединения с сервером:

```
$i=1;
while(!mysql_connect("server","user","password"))
{
echo "Пытаемся подключиться к серверу. Попытка $i <br>";
sleep(60);
$i++;
}
```

Защищенные соединения

27.1. Настройка SSL-соединения

SSL (Secure Sockets Layer) является методом шифрования, разработанным компанией Netscape для обеспечения безопасности в Интернет. Этот метод поддерживает несколько методов шифрования и обеспечивает аутентификацию, как на уровне клиента, так и на уровне сервера. SSL работает на транспортном уровне и поэтому обеспечивает надежное шифрование всех типов данных. Более подробно о реализации SSL можно прочитать на сайте компании Netscape — `http://home.netscape.com/info/security-doc.html`

Протокол **S-HTTP** является еще одним «безопасным» Интернет-протоколом. Он был разработан для предоставления конфиденциальности данных, передаваемых через соединение. Конфиденциальность нужна, например, при передаче номеров кредитных карточек и прочей важной информации.

Модуль **mod_ssl** реализует в сервере Apache **слой SSL**, который осуществляет шифрование всего потока данных между клиентом и сервером. Для всех остальных частей Web-сервера модуль **mod_ssl** является прозрачным. Для работы в этом режиме требуется браузер, поддерживающий механизм SSL (этому условию удовлетворяют все современные распространенные браузеры).

Что касается установки, то вам необходимо установить пакет **OpenSSL** (http://www.openssl.org), хотя, возможно, у вас в системе уже установлен этот пакет. Для проверки этого введите **openssl**, и если вы увидите в ответ приглашение `OpenSSL>`, значит OpenSSL уже установлен. В противном случае для установки OpenSSL выполните следующие шаги (перед этим выполните команду **su** для того, чтобы приобрести привилегии суперпользователя):

1. Распакуйте последнюю версию OpenSSL командой:
   `tar zxvf openssl-x.y.z.tar.gz (x.y.z - номер версии).`
2. Перейдите в каталог openssl-x.y.z и выполните команду:
   `./config.`

3. Если все нормально (нет ошибок), введите команду:

```
make
```

Примечание.

При возникновении ошибок, скорее всего, вам придется установить недостающие пакеты.

4. Затем нужно ввести команды **make install** и **ldconfig**. Перед выполнением команды **ldconfig** убедитесь, что в файле `/etc/ld.so.conf` прописан путь к библиотекам OpenSSL (по умолчанию это `/usr/local/ssl/lib`).

Теперь осталось подключить **mod_ssl** к Apache. При этом следует учитывать, что вам нужна версия **mod_ssl**, которая совместима с вашей версией Apache. Иначе модуль **mod_ssl** будет некорректно работать или вообще откажется что-либо делать. Последние цифры в названии модуля указывают на совместимость с определенной версией Apache. Например, для Apache 1.3.14 нужен файл **mod_ssl-2.7.1-1.3.14.tar.gz**. Здесь 2.7.1 — версия **mod_ssl**.

Распакуйте модуль командой:

```
tar zxvf mod_ssl-x.y.z-2.0.0.tar.gz
```

и выполните команду:

```
./configure -with-apache=../apache_1.3.14 -with-ssl=../openssl-0.9.5
```

В данном примере я использую OpenSSL 0.9.5. Теперь перейдите в каталог с Apache, откомпилируйте его и установите сертификат:

```
cd ../apache-1.3.14
make
make certificate
make install
```

Таким образом, вы установите Apache в каталог, указанный в опции **prefix** (по умолчанию `/usr/local/apache`). Теперь попробуйте запустить Apache. Это можно сделать с помощью команды:

```
usr/local/apache/bin/apachectl startssl
```

Параметр **startssl** необходим для включения SSL. Сервер Apache уже функционирует, однако обратиться по протоколу **https** вы еще не можете. Для этого вам нужно сконфигурировать виртуальные хосты, которые будут использовать протокол **https**. Но для начала необходимо настроить Apache для прослушивания порта 443. С этой целью откройте любым

редактором файл /etc/httpd/conf/httpd.conf и добавьте в него следующие строки:

```
Listen 443
NameVirtualHost x.x.x.x:443
```

Примечание.
Порт 443 — это стандартный порт для протокола HTTPS. Именно по этому порту Apache будет прослушивать виртуальные хосты.

Теперь непосредственно приступите к созданию виртуального сервера, работающего по протоколу https, для чего продолжите редактирование файла /etc/httpd/conf/httpd.conf. Пример того, что необходимо при этом ввести, приведен в листинге 27.1.

Листинг 27.1. Виртуальный https-сервер

```
<VirtualHost x.x.x.x:443>
# Эти строки нужны для поддержки SSL
SSLEngine on
SSLLogLevel warn
SSLOptions +StdEnvVars
SSLCertificateFile /usr/local/apache/conf/ssl.crt/server.crt
SSLCertificateKeyFile /usr/local/apache/conf/ssl.key/server.key
SSLLog /usr/local/apache/logs/ssl_engine_log
# ---------------------

ServerName www.dhsilabs.com
ServerAdmin webmaster@den.dhsilabs.com
DocumentRoot /var/httpd/www/html
ErrorLog /var/https/www/logs/error.log

</VirtualHost>
```

После таких строк вы можете конфигурировать свой виртуальный хост как обычно.

Теперь нужно перезапустить сервер **httpd**. При запуске Apache потребует ввести пароль. Если вы не хотите вводить его при каждом запуске системы, перейдите в каталог, где находится файл ssl.key и выполните следующие команды:

```
cp server.key server.key.org
openssl rsa -in server.key.org -out server.key
chmod 400 server.key
```

Почти все готово! Теперь сервер не должен запрашивать пароль и будет работать в нормальном режиме. При обращении `https://host.domain` браузер запросит вас на предмет использования сертификата.

Чтобы Apache по умолчанию стартовал с поддержкой SSL, исправьте в файле `bin/apachectl` условие **start** на **startold**, а **startssl** на просто **start**. Затем, находясь в каталоге `.usr/local/bin`, установте ссылку **openssl**:

```
ln -s /usr/local/ssl/bin/openssl openssl
```

27.2. Генерирование сертификатов

Сертификат гарантирует безопасное подключение к Web-серверам и (или) удостоверяет личность владельца. Идентификация обеспечивается путем применения личного ключа, известного только пользователю данной системы. Когда пользователь посещает защищенный узел для передачи секретной информации (например, номеров кредитных карточек) по протоколу **https**, узел автоматически пошлет ему сертификат.

Итак, давайте приступим к генерированию сертификатов. Для этого сначала выполните команду:

```
openssl genrsa -des3 -out server.key 1024
```

Она создаст файл `server.key`. После этого вы должны подать запрос в службу верификации:

```
openssl req -new -key server.key -out server.csr
```

Здесь вам придется ответить на вопросы. Если вы ошибетесь, ничего страшного, все можно будет повторить заново. В том случае, если запрос сгенерирован правильно, вы должны получить такую надпись:

```
You now have to send this Certificate Signing Request (CSR)
to a Certifying Authority (CA) for signing
```

Отвечая на вопросы, будьте очень внимательны — ваши ответы увидит весь мир.

По всем правилам, вы сейчас должны подписать сертификат у доверенного лица, но за неимением желания платить за это деньги, подпишите сами себя:

```
openssl genrsa -des3 -out ca.key 1024
openssl req -new -x509 -days 365 -key ca.key -out ca.crt
```

В данном случае у вас получится «самоподписанный» сертификат. Если вы хотите получить полноценный сертификат, то вам придется заплатить за подпись деньги. Для этого добро пожаловать на сайт **www.thawte.com**. В России представителем этого сайта является **solutions.rbc.ru**. Компания

ThawTe занимается генерированием полноценных сертификатов. Получить сертификат от ThawTe можно, естественно, за деньги. На сайте же **solutions.rbc.ru** просто перепродают услуги компании ThawTe.

Возвращаясь к генерированию сертификата, скопируйте `sign.sh` из пакета **mod_ssl** в каталог с ключами и подпишите себя:

```
./sign.sh server.csr
```

Если на экране появится надпись:

```
Now you have two files: server.key and server.crt.
These now can be used as following
```

то это означает, что все собрано правильно. Затем скопируйте новые файлы `server.key` и `server.crt` на место старых. Выполните команду **make** в каталоге с **.crt**-файлом.

В заключение вы получите полностью работающий Apache, защищенный SSL. Для сбора полной информации о работе SSL введите:

```
openssl s_client -connect localhost:443 -state -debug
```

27.3. Авторизация и переменные среды окружения

Для авторизации пользователя используются две переменные окружения:

- AUTH_TYPE
- REMOTE_USER

Первая содержит тип авторизации, обычно Basic. А вторая — имя пользователя, прошедшего авторизацию.

В этой книге я не стану описывать механизм авторизации полностью, поскольку этой теме посвящены целые тома. Рассмотрим основную, basic-авторизацию.

Сценарий отправляет браузеру следующий заголовок:

```
WWW-Authenticate: Basic realm="Zone"
HTTP/1.0 Unauthorized
```

Что же произойдет после этого? Пользователь увидит окно, подобное изображенному на рис. 27.1.

Параметр **Zone** — это имя зоны (области), доступ к ресурсам которой получит пользователь, если пройдет авторизацию. Вместо Zone можно указать все, что угодно. Фактически, этот параметр служит только для подсказки в окне браузера.

После вышеуказанного заголовка сценарий должен отправить тело документа, который будет отображен, если пользователь отменил авторизацию (нажал кнопку «Cancel»).

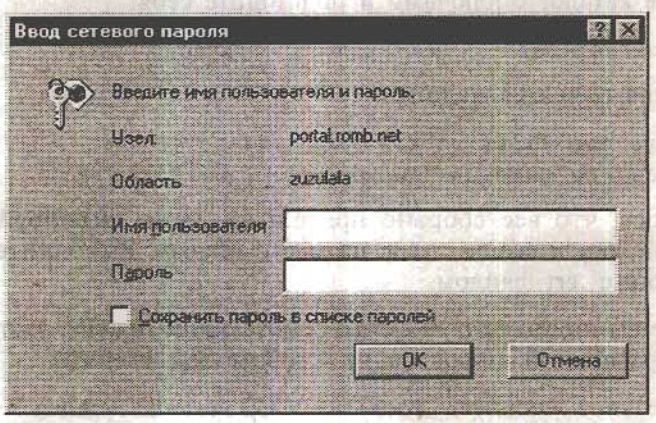


Рис. 27.1. *Ввод сетевого пароля*

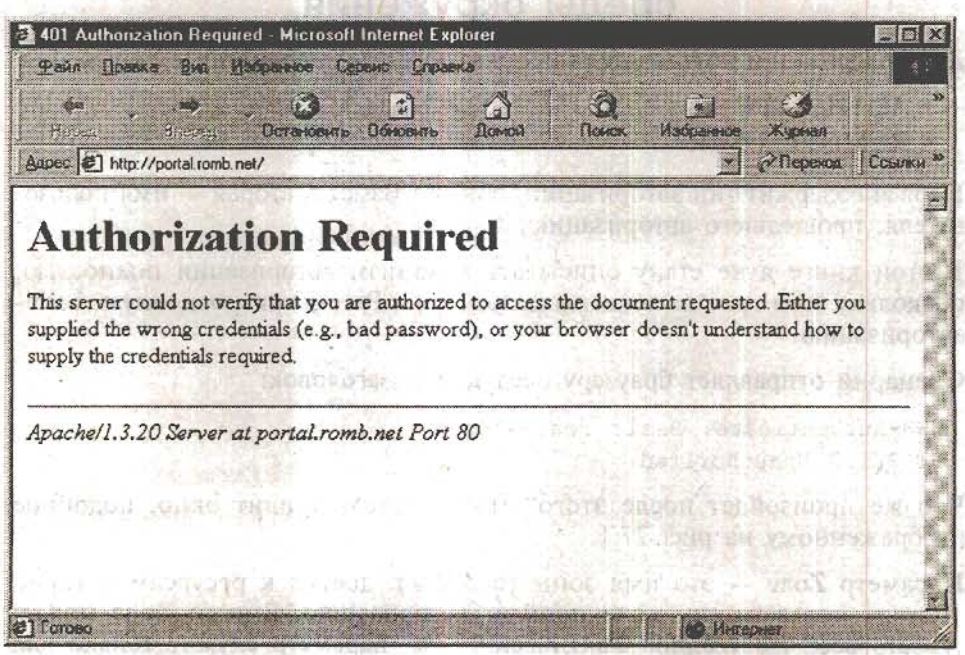


Рис. 27.2. *Ошибка авторизации*

После того, как пользователь ввел имя пользователя и пароль, они будут переданы серверу вместе со всеми остальными заголовками. Нужный заголовок будет выглядеть примерно так:

```
Authorization: Basic GPa95nWkjZ5a==
```

Это ничто другое, как закодированные данные, которые ввел пользователь. Этот заголовок должен быть передан сценарию, который декодировав данные, решит, нужно ли повторить запрос заново или же «пропустить» пользователя.

Если введенные имя и пароль правильны, то они будут записаны в скрытом Cookie, который будет «жить» в течение одной сессии, то есть до закрытия окна браузера. Получается, что пользователю не нужно вводить заново имя пользователя и пароль, и он сможет комфортно работать со сценарием.

После того, как пользователь прошел авторизацию, будет установлена переменная окружения REMOTE_USER, содержащая имя пользователя.

Если Web-сервер поддерживает идентификацию пользователя согласно RFC 931, то эта переменная REMOTE_IDENT будет содержать имя пользователя, полученное от сервера.

Напоследок напомню вам о параметре Cookie, который связан с защищенным соединением — **secure**. Этот параметр устанавливается при передаче Cookies по защищенному соединению.

Глава 28

Два проекта в одной главе

С появлением персональных компьютеров человечество все чаще и чаще прибегает к помощи своих электронных друзей. А разработчики аппаратного и программного обеспечения делают все необходимое, чтобы привлечь все новых и новых пользователей к персоналкам, и чтобы эти пользователи все реже и реже подымались из-за компьютера... Хорошо это или плохо? Судите сами: некоторые без калькулятора 2 + 3 (2 + 2 все и так знают, что 4 будет) сложить не могут, с другой стороны, мы можем, не вставая из-за компьютера, тем более, не выходя из дому в мерзкую погоду, учиться в университете или посещать магазин. Именно об этих двух достижениях — Internet-магазине и дистанционном обучении — мы поговорим в этой главе.

Данная глава имеет большое практическое значение: в ней представлено несколько серьезных сценариев, а после каждого пункта вас ждет домашнее задание. Выполнить его в ваших интересах — практика никогда не помешает.

28.1. Пишем свой Интернет-магазин

28.1.1. Вводная часть

Что такое Интернет-магазин, наверняка знают все или почти все. В Сети существует множество виртуальных магазинов, в которых можно заказать практически все — от компакт-диска до автомобиля или золотых часов, стоимостью 25000 американской валюты.

Принципиально Интернет-магазин не отличается от любого другого магазина. Вы заходите на сайт, регистрируетесь, выбираете товар, затем принимаете решение, хотите ли вы его покупать или нет. Потом с вашего счета снимается определенная сумма, и заказанный вами товар присылается почтой прямо вам домой. Ясное дело, покушать через Интернет-магазин не закажешь, но не потому, что продукты испортятся, а потому, что есть большая вероятность их не дождаться и умереть с голо-

ду, поскольку почтой товары идут от нескольких дней до месяца. Хотя в некоторых городах эта проблема с успехом решена.

Сейчас и мы попытаемся написать свой Интернет-магазин. Если вы заметили, основной особенностью наших виртуальных магазинов от тех, что за «бугром», является совершенно иная система оплаты. Если «там» используются только кредитные карточки, то у нас — практически везде — наложенный платеж. Что это такое? Вы заказываете товар, затем по электронной почте подтверждаете заказ, и администрация магазина отправляет вам товар почтой — сначала вы ничего не платите. Через определенное время вам приходит извещение, вы идете на почту и оплачиваете заказанные вами товары там — без денег вам его никто не даст. Недостатком данного способа является то, что стоимость товара за счет услуг почты возрастает на 10...25%.

Почему в нашем примере мы не будем использовать кредитные карточки? Во-первых, большая часть населения, обладающая кредитками, не будет оставлять номер кредитки, где попало, тем более в нашем Интернет-магазине. Лучше уж заплатить на 15% дороже, чем с твоей кредитки снимут все до последнего рубля. Во-вторых, далеко не у всех есть кредитные карточки. А если и есть, то их владельцы стараются снимать с них деньги сразу после их начисления. В-третьих, для работы с кредитками нужно настроить SSL-соединение, написать сценарий, использующий SSL, а это уже тема для отдельной книги.

28.1.2. Структура Интернет-магазина

Приступим к созданию собственного виртуального магазина. Для начала нужно определиться, что мы будем там продавать. Я предлагаю продавать компакт-диски и книги. Следовательно, у нас будет две категории товара — CD и Book.

Принцип работы нашего магазина будет следующим. Пользователь заходит на наш сайт и регистрируется, то есть вводит свою фамилию, почтовый адрес и E-mail. Не думаю, что один и тот же пользователь будет покупать что-либо более одного раза в месяц, учитывая скорость работы почты. Поэтому заводить специальную таблицу, в которой будут храниться информация о пользователе, я не вижу смысла. Даже если пользователь зайдет два или три раза с целью что-нибудь купить, ничего страшного с ним не случится, если он введет заново свое имя и свой адрес. Тут все выигрывают: и пользователь, и мы. Пользователь не забудет свое имя, а нам не нужно будет хранить на диске многомегабайтную базу данных с данными о пользователях, которые, возможно, не сделали ни одной покупки.

Конечно, такая таблица пригодилась бы нам для статистики и для скидок самым активным покупателям, но при необходимости вы можете сделать все это сами, наша сейчас задача — создать Интернет-магазин.

Вернемся к нашему пользователю. Он зашел, зарегистрировался. Затем ему нужно выбрать категорию товара — книги или компакты. После этого будет выведен список всех книг или компактов. Он выберет нужные ему товары и укажет их количество. После этого, если он нажмет кнопку «Заказ», ему будет предоставлен счет — список заказанных товаров и общая сумма. Если он согласен оплатить эту сумму, нажимает кнопку «Согласен», и на указанный E-mail будет выслано подтверждение заказа. Если он ответит на письмо (ведь он может указать данные соседа), администрация магазина отправит ему наложенным платежом заказанные им товары.

Таблица товаров выглядит следующим образом:

- **Поле ID** — этот идентификатор пригодится нам при оформлении заказа — с номером работать проще (да и нагрузка на трафик меньше), чем с полным названием компакт-диска или книги.
- **Поле категория (cat)** — содержит два символа: BK — книга, CD — компакт-диск.
- **Поле Описание (opis)** — содержит описание товара (64 символа).
- **Поле Price** — это цена товара.

По своему желанию вы можете добавить год выпуска, дату поступления товара в магазин, оставшееся количество товара на складе, но мы этого делать не будем.

У нас будет всего одна таблица, поэтому мы ее можем без зазрений совести поместить в базу данных `test`, но мы этого делать не будем — вдруг вы захотите расширить возможности Интернет-магазина. База данных нашего виртуального магазина будет называться... Нужно придумать название нашему магазину. Ничего, кроме MyShop в голову не лезет. Видно разработчик MySQL тоже не смог придумать более оригинального имени. Но имя, это не главное — «скромный» MySQL теперь используется многими программистами. Вдруг и наш магазин будет когда-то таким же знаменитым, как и MySQL.

Как вы уже догадались, база данных будет называться **myshop**. Создать ее можно тремя способами:

- С помощью средства **phpMyadmin** — универсальное средство.
- Программой **mysqladmin** — если у вас есть доступ к консоли сервера и администратор MySQL разрешил вам создавать базы данных

```
mysqladmin create myshop
```

- С помощью функции **mysql_createdb()**, опять же таки, если администратор разрешил вам создавать базы данных самостоятельно. Обычно *для отдельного пользователя заводится его собственная база данных*, которую используют все сценарии, установленные пользователем:

```
mysql_createdb("myshop");
```

28.1.3. Пишем вспомогательные скрипты

Для автоматизации процесса установки я предлагаю сценарий, приведенный в листинге 28.1.

Листинг 28.1. Файл setup.php

```
<?
require "header.php";
require "options.php";

echo $head;
echo "<H1>Программа установки</h1>";

if(!mysql_connect($SERVER,$USER,$PASSWD))
{
echo "<h1>Ошибка сервера MySQL. MySQL server error</h1>";
echo "</body></html>";
exit;
}

echo "<p>Пытаемся создать базу данных...";
mysql_createdb($DB);

if (mysql_errno()==0)
{
echo "<p>База данных создана успешно! Создаем таблицы...";
mysql_select_db($DB);

mysql_query('
CREATE TABLE tovar (
id int primary key,
cat char(2) not null default \'CD\',
opis char(64) not null default \'\',
price numeric(9,2) not null default \'0\'
)
');
echo mysql_error();
if (mysql_errno()===0)
{
echo "<p>Таблица создана. Добавляем данные";
mysql_query('INSERT INTO tovar values(1,"BK","Колисниченко Д.Н.
Linux-сервер своими руками, 2002","175")');
mysql_query('INSERT INTO tovar values(2,"BK","Колисниченко Д.Н.
Самоучитель Linux, 2003","135")');
```

```
mysql_query('INSERT INTO tovar values(3,"CD","Linux Mandrake
9.1 6CD","235")');
mysql_query('INSERT INTO tovar values(4,"CD","Linux RedHat 7.2
5CD","200")');
echo "<p>Инсталляция завершена!";
}
else { echo "<p>Ошибка при создании таблицы"; }

}
else
{
echo "Ошибка при создании БД: ".mysql_error();
}
?>
```

Немного прокомментирую принцип работы сценария. Сначала мы подключаемся к серверу.

```
if(!mysql_connect($SERVER,$USER,$PASSWD))
{
echo "<h1>Ошибка сервера MySQL. MySQL server error</h1>";
echo "</body></html>";
exit;
}
```

Обратите внимание, что параметры подключения — имя пользователя, пароль, а также имя самого сервера, мы берем из файла `options.php`, который является общим для всего проекта. Код этого сценария приведен в листинге 28.2.

Листинг 28.2. Файл options.php

```
<?
// Имя сервера, можно указать IP-адрес
$SERVER = "localhost";
// Имя пользователя
$USER = "root";
// Пароль
$PASSWD = "";
// Название базы данных
$DB = "myshop";
// Адрес для подтверждения заказа
$REPLY = "buy@myshop.net";
?>
```

После успешного подключения мы пытаемся создать базу данных:

```
mysql_createdb($DB);
```

Если произойдет ошибка, функция **mysql_error()** сообщит нам ее причину. База данных успешно создана (mysql_errno() == 0)? Значит, самое время создать таблицу `tovar`:

```
CREATE TABLE tovar (
id int primary key,
cat char(2) not null default \'CD\',
opis char(64) not null default \'\',
price numeric(9,2) not null default \'0\'
)
```

Если база данных создана успешно, скорее всего, таблица tovar у вас тоже будет создана без проблем. После ее создания мы наполним ее начальными данными — для примера. У нас будет две книги (категория BK) и два компакт-диска (категория CD):

```
INSERT INTO tovar values(1,"BK","Колисниченко Д.Н. Linux-сервер
своими руками, 2002","175")
INSERT INTO tovar values(2,"BK","Колисниченко Д.Н. Самоучитель
Linux, 2003","135")
INSERT INTO tovar values(3,"CD","Linux Mandrake 9.1 6CD","235")
INSERT INTO tovar values(4,"CD","Linux RedHat 7.2 5CD","200")
```

После успешного завершения работы программы-инсталлятора (рис. 28.1) не забудьте удалить файл `setup.php` — из соображений безопасности.

Рис. 28.1. *Установка Интернет-магазина*

Я решил не разбивать проект на много файлов (сценариев), поэтому все действия магазина выполняет один-единственный файл — ***index.php***. Правда, я вынес заголовок в отдельный файл `header.php` (см. листинг 28.3), чтобы вы при необходимости могли изменить логотип, кодировку и другие параметры, не изменяя кода самого сценария. Еще нам понадобится сценарий `options.php`, в котором содержатся параметры подключения.

Листинг 28.3. Вспомогательный файл header.php

```
<?
$head = '<html><head>
<meta http-equiv=Content-Type content=text/html; charset=koi8-r>
<title>My Shop: самые свежие CD и книги!</title></head>
<body><table width=100% border=0 cols=2><tr><td><img
src=myshop_logo.png>
</td><td><h1>Добро пожаловать!</h1><p>Самые новые компакт-диски
и книги издательства "Наука и Техника" - только на нашем сай-
те</td></tr></table>';

?>
```

28.1.4. Пишем скрипт Интернет-магазина

Вот теперь перейдем к написанию сценария `index.php`. Для начала напомню алгоритм работы. Он немного отличается от того, который был в начале — я его несколько упростил «на лету». Итак, пользователь заходит, регистрируется, выбирает категорию (книги или компакты), выбирает товар и нажимает кнопку «Заказать». После этого ему будет отправлено уведомление о сделанном заказе на указанный E-mail. Если он подтвердит заказ, то товар будет выслан ему наложенным платежом.

Регистрационные данные будут храниться в Cookies браузера до окончания сеанса, то есть закрытия окна браузера.

Сценарий использует вспомогательную функцию `ShowTovar()`, показывающей товар определенной категории. Функции нужно передать лишь один параметр — категорию. Код функции приведен в листинге 28.4.

Листинг 28.4. Функция ShowTovar()

```
function ShowTovar($ct="CD")
{
/* Категория по умолчанию - CD
Глобальные переменные (объявлены в файле options.php */

global $SERVER, $USER, $PASSWD, $DB;
if(!mysql_connect($SERVER,$USER,$PASSWD))
```

```
{
echo $head;
echo "<h1>Ошибка сервера MySQL. MySQL server error</h1>";
echo "</body></html>";
exit;
}
mysql_select_db($DB);

echo "<table WIDTH=100% border=2 cellspacing=0 cellpadding=3
bordercolor=#336699>";
echo "<tr><td bgcolor=#336699><font color=white>Наименование</
td>
<td bgcolor=#336699><font color=white>Цена</td>
<td bgcolor=#336699><font color=white>Заказ</td>
</tr>";

if($ct=="BK") echo "<h1>Книги</h1>";
else echo "<h1>Компакт-диски</h1>";
$r=mysql_query("select * from tovar where cat=\"$ct\"");

echo "<form method=post action=index.php>";
echo mysql_error();
for ($i=0; $i<mysql_num_rows($r); $i++)
{
echo "<tr>";
$f=mysql_fetch_array($r);
echo "<td>$f[opis]</td><td>$f[price]</td>
<td><input type=checkbox name=Arr[] value=".$f['id']."
</tr>";
}
echo "</table>";
echo "<p><input type=submit name=buy value=Заказать>
<input type=Reset value=Сброс></form>
<p>&copy Денис Колисниченко (dhsilabs at mail.ru), 2003</
body></html>";
}
```

Вывести список товара определенной категории предельно легко. Для этого достаточно подключиться к серверу, выбрать базу данных, выполнить запрос:

```
select * from tovar where cat="категория"
```

и оформить результат в виде таблицы. В книге мы не раз уже писали подобные сценарии. А вот как организовать выбор товара. Можно сделать так. Каждую строку сделать ссылкой на какой-нибудь сценарий, можно даже на этот, который будет заносить выбранный товар во временное хранилище на нашем диске или в Cookies браузера пользовате-

ля. Затем пользователь нажимает кнопку «Назад» и выбирает следующий товар. Согласитесь, этот способ не очень удобен. Да и не хочется работать с файлами — всегда будут проблемы с доступом.

Я предлагаю сделать следующее. Возле каждого наименования товара поставить независимый переключать — **checkbox**. Пользователь выбирает все необходимые ему товары и нажимает «Заказать». Все, что он выбрал, будет передано нашему сценарию. Вот только как обработать заказ — мы же не знаем имени каждого **checkbox**. Не будем же мы писать `name=checkboxID`, где ID — это номер товара в базе данных.

Тут нам на помощь приходит сам PHP. Кроме скалярных переменных PHP позволяет передавать целые массивы. У нас «именем собирательным» будет массив `Arr[]`. В качестве ключей будем использовать порядковый номер товара (не ID), а в качестве значения — ID. Например, по запросу:

```
select * from tovar where cat='CD'
```

сервер возвратил следующий результат:

```
3 "CD" "Linux Mandrake 9.1 6CD" 235.00
4 "CD" "Linux RedHat 7.2 5CD" 200.00
5 "CD" "ALT Linux Junior 2.2 1CD" 95.00
```

Пользователь выбрал первые два наименования. Что будет передано сценарию. А сценарию будет передан такой массив:

```
Arr[0] = 3
Arr[1] = 4
```

Товар с номером 3 прибыл первым, поэтому номер элемента массива равен 0, а товар с номером 4 «дошел» вторым, следовательно, номер элемента массива равен 1.

Все это было сказано только для общего понимания — мы не будем работать напрямую с номерами элементов массива, поскольку будем использовать цикл **foreach()**.

Следующий фрагмент функции `ShowTovar()` выводит список товара вместе с переключателями возле каждой позиции:

```
for ($i=0; $i<mysql_num_rows($r); $i++)
{
echo "<tr>";
$f=mysql_fetch_array($r);
echo "<td>$f[opis]</td><td>$f[price]</td>
<td><input type=checkbox name=Arr[] value=".$f['id']."
</tr>";
}
```

Указывать номер элемента массива явно (то есть `Arr[$i]`) не нужно — PHP сам позаботится об этом.

Продолжим создание нашего магазина. Для сохранения регистрационных данных пользователя в Cookies браузера будем использовать следующие переменные:

- myshop_name — имя пользователя;
- myshop_family — фамилия;
- myshop_otch — отчество;
- myshop_email — электронный адрес;
- myshop_country — страна;
- myshop_city — город;
- myshop_address — адрес;
- myshop_zip — почтовый индекс.

Префикс myshop_ необходим, чтобы наши имена переменных не пересекались с данными других сценариев, которые, возможно, используют эти же имена переменных.

Первым делом нужно проверить, зарегистрирован ли пользователь. Это можно сделать двумя способами — проверить существование переменной $register — это имя кнопки в форме регистрации. Можно также проверить существование переменной myshop_email — это единственное обязательное поле при регистрации. Что мы и сделаем:

```
if (!isset($myshop_email)) {

echo $head;
echo "<p>Вы не зарегистрированы. Пожалуйста оставьте информацию
о себе. <PRE>";
echo "<form action=index.php method=post>
<p>Ваше имя <input type=text name=myshop_name> Фамилия  <input
type=text name=my
shop_family>
<p>Отчество <input type=text name=myshop_otch> E-mail    <input
type=text name=my
shop_email> (*)
<p>Страна    <input type=text name=myshop_country> Город
<input type=text name
=myshop_city>
<p>Адрес     <input type=text name=myshop_address> Индекс
<input type=text name
=myshop_zip>
<p>           <input type=submit name=register value=Ок>
<p>&copy Денис Колисниченко (dhsilabs at mail.ru), 2003</
body></html>";

}
else
{
```

Если E-mail указан, считаем, что пользователь зарегистрирован. Далее будем проверять существование следующих переменных:

- `$register` — если установлена, значит, пользователя нужно зарегистрировать, то есть добавить с Cookies регистрационные данные;
- `$buy` — нужно вывести счет и отправить уведомление о заказе на указанный E-mail;
- `$myshop_cat` — указана категория, следовательно, нужно вывести список товара указанной категории.

Регистрируем пользователя:

```
if (isset($register))
{
SetCookie("myshop_name",$myshop_name);
SetCookie("myshop_family",$myshop_family);
SetCookie("myshop_otch",$myshop_otch);
SetCookie("myshop_email",$myshop_email);
SetCookie("myshop_country",$myshop_country);
SetCookie("myshop_city",$myshop_city);
SetCookie("myshop_address",$myshop_address);
SetCookie("myshop_zip",$myshop_zip);
}

// Первый вывод нужно сделать только ПОСЛЕ установки Cookies,
иначе они не будут
// установлены
echo $head;
```

Пользователь хочет что-нибудь купить:

```
if(isset($buy))
{
// Формируем сообщение с уведомлением о заказе
$message = "Здравствуйте, $myshop_name!\n".date("d.m.Y")." вы
посетили наш Internet-магазин\n";
$message = $message."Ваш заказ:\n";

// Формируем запрос вида:
// select * from tovar where (id=Arr[0]) or (id=Arr[1]) or ...
or (id=Arr[N])

$q = "select * from tovar where";
foreach($Arr as $k=>$v) $q=$q." (id=$v) or ";
$q = substr($q,0,strlen($q)-4);

// Можно раскомментировать для отладки
```

```
// echo "$q";

// Подключаемся к серверу
if(!mysql_connect($SERVER,$USER,$PASSWD))
{
echo $head;
echo "<h1>Ошибка сервера MySQL. MySQL server error</h1>";
echo "</body></html>";
exit;
}
// Выбираем базу данных
mysql_select_db($DB);

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

// Можно раскомментировать для отладки
// echo mysql_error();

// Выводим счет
echo "<h1>Ваш заказ:</h1>";
echo "<table WIDTH=100% border=2 cellspacing=0 cellpadding=3
bordercolor=#336699
>";
echo "<tr><td bgcolor=#336699><font color=white>Наименование</
td>
<td bgcolor=#336699><font color=white>Цена</td></tr>";

// Это общая сумма
$sum = 0;
for ($i=0; $i<mysql_num_rows($r); $i++)
{
echo "<tr>";
$f=mysql_fetch_array($r);
echo "<td>$f[opis]</td><td>$f[price]</td>";
$message=$message."$f[opis] \t\t $f[price]\n";
$sum+=$f['price'];
echo "</tr>";
}
echo "</table>";
echo "<p>Общая сумма заказа: $sum";

// Продолжаем формировать сообщение пользователю
$message=$message."\nОбщая сумма: $sum\n
```

```
Для подтверждения заказа просто нажмите Reply\n\n
Спасибо за посещение нашего магазина!
-------------------
Регистрационная карточка:\n
Фамилия $myshop_family\n
Имя $myshop_name\n
Отчество $myshop_otch\n
Email $myshop_email\n
Страна $myshop_country\n
Город $myshop_city\n
Адрес $myshop_address\n
Индекс $myshop_zip\n
";

// Отправляем сообщение. Переменная $REPLY указана в файле
options.php
mail($myshop_email,"MySHOP: ваш заказ",$message,"From: $REPLY");
echo "<p><b>Вам отправлено уведомление о заказе\n";

// Нужно, чтобы сценарий опять не попросил вас выбрать
// категорию товара.
// Впрочем, можно закомментировать - как кому больше нравится
exit;

}
```

Если вы внимательно прочитали приведенный фрагмент кода, то заметили, что счет и сообщение пользователю формируется одновременно, поэтому нам не нужно два раза «крутить шарманку», то есть просматривать один и тот же результат `$r`.

Следующий, последний блок, выводит товар указанной категории:

```
if (!isset($myshop_cat))
{
echo "<h2>Вы хотите купить:</h2>
<ul>
<li><a href=index.php?myshop_cat=CD>Компакт-диск</li></a>
<li><a href=index.php?myshop_cat=BK>Книгу</li></a>
</ul>";
}
else
{
ShowTovar($myshop_cat);
}
```

Все, сценарий написан. Можно приступать к тестированию. Полный исходный код сценария Интернет-магазина приведен в листинге 28.5.

Листинг 28.5. Сценарий index.php – полный исходный код

```
<?
// Подключаем вспомогательные файлы
include "header.php";
include "options.php";

// Функция выводит товар указанной категории
// Категория по умолчанию - CD
function ShowTovar($ct="CD")
{
global $SERVER, $USER, $PASSWD, $DB;
if(!mysql_connect($SERVER,$USER,$PASSWD))
{
echo $head;
echo "<h1>Ошибка сервера MySQL. MySQL server error</h1>";
echo "</body></html>";
exit;
}
mysql_select_db($DB);

if($ct=="BK") echo "<h1>Книги</h1>";
else echo "<h1>Компакт-диски</h1>";

echo "<table WIDTH=100% border=2 cellspacing=0 cellpadding=3
bordercolor=#336699>";
echo "<tr><td bgcolor=#336699><font color=white>Наименование</
td>
<td bgcolor=#336699><font color=white>Цена</td>
<td bgcolor=#336699><font color=white>Заказ</td>
</tr>";

$r=mysql_query("select * from tovar where cat=\"$ct\"");

echo "<form method=post action=index.php>";
echo mysql_error();
for ($i=0; $i<mysql_num_rows($r); $i++)
{
echo "<tr>";
$f=mysql_fetch_array($r);
echo "<td>$f[opis]</td><td>$f[price]</td>
```

```
<td><input type=checkbox name=Arr[] value=".$f['id']."
</tr>";
}
echo "</table>";
echo "<p><input type=submit name=buy value=Заказать><input
type=Reset value=Сброс></form>
<p>&copy Денис Колисниченко (dhsilabs at mail.ru), 2003</
body></html>";
}


// Проверяем, зарегистрирован ли пользователь
if (!isset($myshop_email)) {

echo $head;
echo "<p>Вы не зарегистрированы. Пожалуйста, оставьте
информацию о себе. <PRE>";

echo "<form action=index.php method=post>
<p>Ваше имя <input type=text name=myshop_name> Фамилия  <input
type=text name=myshop_family>
<p>Отчество <input type=text name=myshop_otch> E-mail   <input
type=text name=myshop_email>
<p>Страна   <input type=text name=myshop_country> Город
<input type=text name=myshop_city>
<p>Адрес    <input type=text name=myshop_address> Индекс
<input type=text name=myshop_zip>
<p>          <input type=submit name=register value=Ок>
<p>&copy Денис Колисниченко (dhsilabs at mail.ru), 2003</
body></html>";

}
else
{

// Регистрируем пользователя
if (isset($register))
{
SetCookie("myshop_name",$myshop_name);
SetCookie("myshop_family",$myshop_family);
SetCookie("myshop_otch",$myshop_otch);
SetCookie("myshop_email",$myshop_email);
SetCookie("myshop_country",$myshop_country);
SetCookie("myshop_city",$myshop_city);
```

```
SetCookie("myshop_address",$myshop_address);
SetCookie("myshop_zip",$myshop_zip);
}
echo $head;


// Пользователь уже выбрал товар и нажал кнопку Заказ
if(isset($buy))
{
$message = "Здравствуйте, $myshop_name!\n".date("d.m.Y")." вы
посетили наш Internet-магазин\n";
$message = $message."Ваш заказ:\n";


$q = "select * from tovar where";
foreach($Arr as $k=>$v) $q=$q." (id=$v) or ";
$q = substr($q,0,strlen($q)-4);

// echo "$q";

if(!mysql_connect($SERVER,$USER,$PASSWD))
{
echo $head;
echo "<h1>Ошибка сервера MySQL. MySQL server error</h1>";
echo "</body></html>";
exit;
}
mysql_select_db($DB);

$r=mysql_query($q);

// echo mysql_error();

echo "<h1>Ваш заказ:</h1>";
echo "<table WIDTH=100% border=2 cellspacing=0 cellpadding=3
bordercolor=#336699
>";
echo "<tr><td bgcolor=#336699><font color=white>Наименование</
td>
<td bgcolor=#336699><font color=white>Цена</td></tr>";

$sum = 0;
for ($i=0; $i<mysql_num_rows($r); $i++)
```

```
{
echo "<tr>";
$f=mysql_fetch_array($r);
echo "<td>$f[opis]</td><td>$f[price]</td>";
$message=$message."$f[opis] \t\t $f[price]\n";
$sum+=$f['price'];
echo "</tr>";
}
echo "</table>";
echo "<p>Общая сумма заказа: $sum";

$message=$message."\nОбщая сумма: $sum\n
Для подтверждения заказа просто нажмите Reply\n\n
Спасибо за посещение нашего магазина!
-----------------
Регистрационная карточка:\n
Фамилия $myshop_family\n
Имя $myshop_name\n
Отчество $myshop_otch\n
Email $myshop_email\n
Страна $myshop_country\n
Город $myshop_city\n
Адрес $myshop_address\n
Индекс $myshop_zip\n
";

mail($myshop_email,"MySHOP: ваш заказ",$message,"From: $REPLY");
echo "<p><b>Вам отправлено уведомление о заказе\n";

exit;

}

// Выводим категории или товар определенной категории
if (!isset($myshop_cat))
{
echo "<h2>Вы хотите купить:</h2>
<ul>
<li><a href=index.php?myshop_cat=CD>Компакт-диск</li></a>
<li><a href=index.php?myshop_cat=BK>Книгу</li></a>
</ul>";
}
else
{
ShowTovar($myshop_cat);
}

}
?>
```

21.1.5. Тестируем Интернет-магазин

Вот теперь можно проверить работу сценария. Заходим:

```
http://ваш_сайт/myshop/index.php
```

Рис. 28.2. *Регистрация*

Вводим данные о себе и выбираем категорию товара (рис. 28.3):

Рис. 28.3. *Выбор категории товара – книги или компакты*

Выбираем товар (рис. 28.4) и нажимаем кнопку «Заказать»

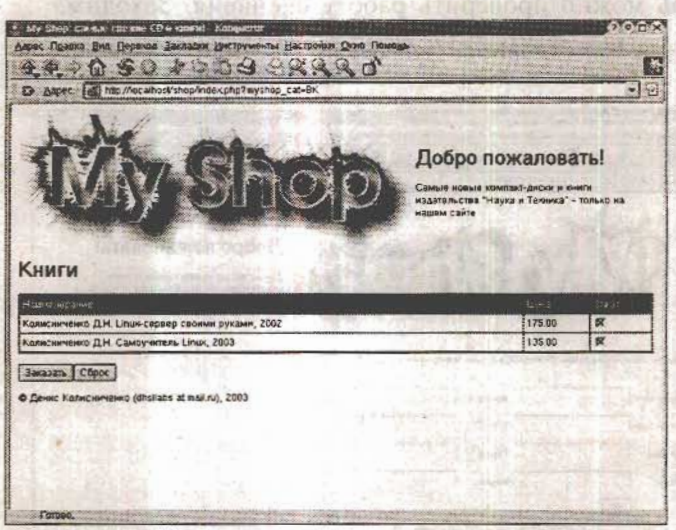


Рис. 28.4. *Выбор товара*

Сценарий покажет вам ваш заказ (рис. 28.5), сообщит общую сумму заказа и отправит уведомление о заказе.

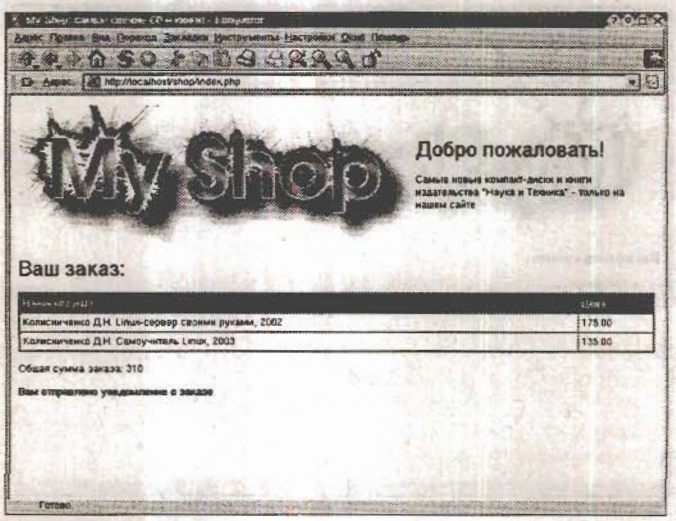


Рис. 28.5. *Ваш заказ*

После этого вам нужно получить сообщение о заказе (рис. 28.6) и просто нажать кнопку «Reply» для его подтверждения.

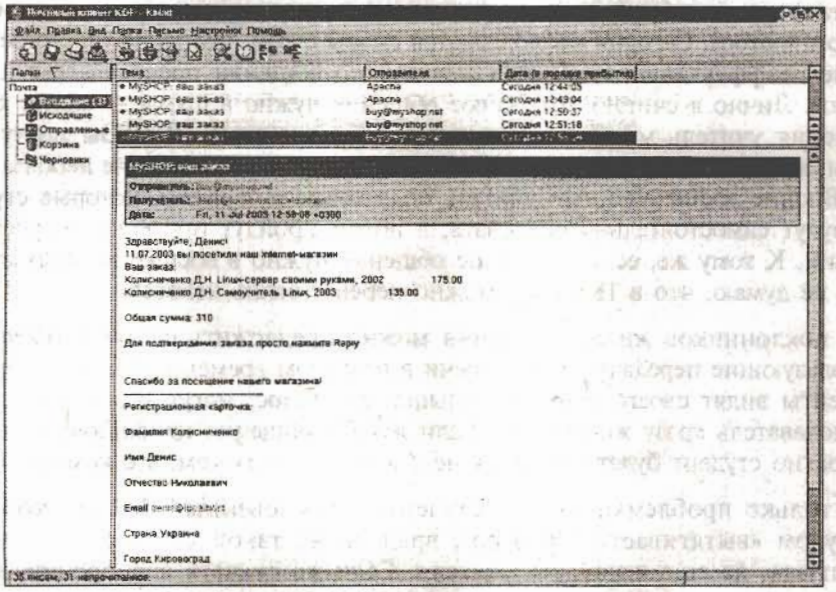


Рис. 28.6. *Уведомление о заказе*

Готовую версию сценария можно взять на моем сайте — `http://dkws.narod.ru/php`. Установить его очень просто — скопируйте в каталог `$DocumentRoot/myshop`, отредактируйте файл `options.php` и запустите сценарий setup.php. Если он работать не будет, создайте базу данных и таблицу вручную.

Вам для самостоятельной работы остается сценарий, позволяющий редактировать таблицу **Товар** — добавлять и удалять записи, а также изменять уже имеющиеся.

28.2. Дистанционное обучение — система проверки знаний

28.2.1. Общие положения

Как вы думаете, что приятнее — покупать вещи в магазине или посещать лекции? Лично мне нравится первое (правда, смотря что именно покупать). Если уже пользователям лень посетить магазин, не думаете ли вы, что они будут ходить на лекции? Для любителей тратить деньги были созданы Internet-магазины, а для тех, кто предпочитает грызть гранит науки, были созданы системы дистанционного обучения.

Дистанционная форма обучения набирает обороты и, скорее всего, станет официальной формой обучения — как стационарная или заочная.

Дистанционное обучение позволяет посещать лекции, не отрываясь от своего компьютера. Сегодня продолжаются споры о том, что дистанционное обучение неэффективно, поскольку нет живого общения преподавателя и студентов. Лично я считаю, что живое общение нужно в школе — когда кроме обучения учитель может воспитывать своих учеников. В ВУЗы поступают люди, желающие чего-нибудь добиться, поэтому живое общение необязательно. Вполне достаточно, что учитель подготовит материалы, которые студенты будут самостоятельно осваивать, а потом пройдут проверку полученных знаний. К тому же, если это самое общение нужно в воспитательных целях, то я не думаю, что в 18-20 лет можно перевоспитать человека.

Для поклонников живого общения можно предложить системы обучения, использующие передачу видео и речи в реальном времени. Да, это не чат — студенты видят своего учителя, слышат его голос, могут задать вопросы и преподаватель сразу же ответит. Если же для общения использовать чат, то все равно студент будет общаться не с живым человеком, а с компьютером.

Вот только проблемка: если у студента обыкновенный dial-up, который с трудом «вытягивает» 33600 bps, вряд ли на такой скорости получится услышать, да еще и увидеть учителя. Если же скачать видеоролик, в котором учитель объясняет материал, то это будет уже не «реальное время». Как видите, пока в нашей стране будут проблемы с линиями связи и дорогами, живого общения в ДО не будет. В последнее время начинают распространяться ADSL-соединения, но пока они не всем доступны.

Еще один момент, который нужно учитывать при ДО. Он связан с проверкой знаний. Где вероятность того, что при сдаче теста студент не откроет книгу и не перепишет правильные ответы?

Конечно, компьютер объективно оценит знания студента. Ведь очень часто случается, что из-за субъективного мнения экзаменатора оценка у одного студента завышена, а у другого — занижена.

В идеале, программа дистанционного обучения должна только передать студенту весь необходимый материал и проверить хотя бы его получение, то есть, скачал ли студент лекцию №5 или он даже не заходил на сайт. А вот для сдачи экзамена студенты должны подняться из-за своего компьютера и придти в университет, где программа-тест проверит уровень полученных знаний. Такие системы будем называть системами проверки знаний (СПЗ). Здесь преподаватель нужен только лишь для того, чтобы присматривать за студентами: а не вытащил ли кто-то шпаргалку?

СПЗ значительно упрощают сам процесс сдачи экзамена. Еще одно преимущество СПЗ — это снижение временных затрат — ведь преподавателю не нужно беседовать отдельно с каждым студентом или проверять его работу — оценка студенту сообщается сразу же.

28.2.2. Параметры системы проверки знаний

В этом пункте мы рассмотрим простую СПЗ, обладающую следующими функциями:

- Восстановление после обрыва соединения. Я писал СПЗ в расчете на модемный канал, где обрывы соединения случаются довольно часто. Реализуется функция через механизм сессий, подробно описанный в этой книге. Если пользователь не закрыл окно браузера, после восстановления соединения, тест будет продолжен с места обрыва. В локальной сети эта функция бесполезна.
- Защита от несанкционированного доступа. Пользователь, перед тем как пройти тест, должен быть зарегистрирован в таблице пользователей SmartTest (кстати, так называется СПЗ). Добавить запись в эту таблицу может администратор СПЗ. Пароли передаются по сети в зашифрованном виде (используется алгоритм MD5) — не беспокойтесь, ваш пароль не увидит ни один сетевой сниффер.
- Несколько вариантов правильных ответов. Максимальное количество вариантов ответа — 6.
- Для каждого теста ведется таблица ответов. В нее заносится имя пользователя и его оценка. Пользователь может пройти тест только один раз. Если же ему нужно пересдать тест, тогда администратору нужно удалить его имя из таблицы ответов определенного теста.
- Номера вопросов генерируются случайным образом.

28.2.3. Организация. Выработка структуры

Подробно описывать работу сценария и комментировать сам сценарий, как в случае с Интернет-магазином, я не стану. У вас уже достаточно квалификации, чтобы самостоятельно во всем разобраться. Подробно рассмотрим лишь структуру таблиц СПЗ SmartTest.

Сценарий `create_tables.php` создает таблицы:

- `ureg` — таблица регистрации пользователей;
- `reg` — таблица регистрации тестов;
- таблицу тестов;
- таблицу ответов.

Запрос, создающий таблицу `ureg`:

```
CREATE TABLE ureg (
login char(16) PRIMARY KEY,
passwd char(128) NOT NULL,
fullname char(32)
)
```

Все просто: в ней содержится логин, пароль и полное имя пользователя. По желанию, можете добавить необходимые вам поля.

С помощью следующего запроса была создана таблица регистрации тестов. Любой тест должен быть зарегистрирован в этой таблице.

```
CREATE TABLE reg (
no int PRIMARY KEY,
desk char(32),
qmax int,
tbl char(64),
autor char(32)
)
```

Поле `no` — это порядковый номер теста, `desk` — описание, `qmax` — максимальное количество вопросов. Данное поле просто необходимо при выборе случайного вопроса, чтобы номер вопроса не вышел за пределы допустимого. Поле `tbl` — это имя таблицы, например, `linux`. Имя таблицы ответов будет `linux_a`. Поле `autor` содержит информацию об авторе, который создал тест.

Запрос для создания таблицы тестов:

```
CREATE TABLE linux (
no int PRIMARY KEY,
quest TEXT,
true char(6),
dif char(1)
)
```

Поле `no` — порядковый номер вопроса. Поле `quest` содержит текст вопроса вместе с вариантами ответов, например:

Какие фирмы участвовали в создании OS/2?

1. Microsoft
2. IBM
3. RedHat
4. Intel
5. Apple

Поле `true` содержит комбинацию правильных ответов. В нашем случае, это 12. Если бы правильным был только вариант 3, то это поле содержало бы 3.

Поле `dif` определяет сложность вопроса. Если он содержит Y, значит, вопрос сложный и за его правильный ответ студенту нужно добавить 3 балла. Если простой, то студент получит 1 балл.

Таблица ответов еще более проста:

```
CREATE TABLE linux_a (
login char(16) NOT NULL PRIMARY KEY,
mark int,
true int
)
```

Первое поле — это логин студента, второе — его оценка, а третье — количество правильных ответов.

Сценарий создания таблиц приведен в листинге 28.6.

Листинг 28.6. Сценарий создания таблиц

```
<?
mysql_connect("localhost");
mysql_select_db("st");

// Создаем таблицу для регистрации пользователей
mysql_query("CREATE TABLE ureg(login char(16) PRIMARY KEY,
passwd char(128) NOT NULL, fullname char(32))");

 $pswd=md5("123456");
 mysql_query("INSERT INTO ureg values('user3','$pswd','Ivanov
Ivan')");
// Создаем таблицу для регистрации тестов
mysql_query("CREATE TABLE reg (no int PRIMARY KEY,desk
char(32),qmax int,tbl char(64),autor char(32))");
mysql_query("INSERT INTO reg VALUES('1','Linux
Test','100','linux','Denis')");

// Создаем таблицу для тестов
mysql_query("CREATE TABLE linux(no int PRIMARY KEY, quest TEXT,
true char(6), dif char(1))");
mysql_query("CREATE TABLE linux_a(login char(16) NOT NULL
PRIMARY KEY, mark int, true int)");

// Заполняем таблицу любыми вопросами (всего вопросов 100)
 for($i=0; $i<100; $i++)
 {
 $msg="Вопрос $i<br>Сколько вам лет?<br>$i?";
 echo $msg;
 if ($i%2==0) $ch='Y'; else $ch='N';
 mysql_query("INSERT INTO linux VALUES($i,'$msg','1','$ch')");
 }
?>
```

Как будем тестировать? Вот алгоритм работы сценария:

1. Пользователь регистрируется. Если логин и пароль правильны, переходим на шаг 2. В противном случае сообщаем, что логин или пароль неправильны, записываем попытку несанкционированного доступа в журнал (нужно обеспечить запись в этот файл).
2. Выводим список тестов. Студент выбирает один из них. Мы проверяем, если он проходил этот тест, тогда сообщаем ему об этом и просим выбрать другой тест. Иначе — переходим на п. 3.
3. Генерируем 10 случайных номеров вопросов и записываем их с Cookies браузера.
4. Задаем по очереди эти 10 вопросов. Если пользователь ответил правильно, то анализируем сложность вопроса, если он сложный, то добавляем 3 балла, а если простой — 1 балл.

```
if ($Answer===$Prev)
{
 $True=@$True+1;
 $Trusted='Y';
 If ($Diff=='Y') $Total=$Total+3;
 else $Total=$Total+1;
} else $Trusted='N';

If ($Diff=='Y') $TT=$TT+3;
else $TT=$TT+1;
```

1. Если счетчик достиг 9 (задано 10 вопросов — от 0 до 9), выводим результат. Также не забываем записать результат в таблицу ответов:

```
if($Counter===9) {
// пора бы и закончить тест
$ATable=$Table."_a";
$Exp=($Total*100)/$TT;
if ($Exp>=90) $Mark=5;
else if($Exp>=75) $Mark=4;
else if($Exp>=50) $Mark=3;
else if($Exp<50) $Mark=2;

mysql_query("INSERT INTO $ATable VALUES('$UserName','$Mark','$True')");
$msg="<br>$FullName, вы ответили правильно на $True вопросов(а).";
$msg=$msg+"<br>Вы набрали $Total баллов из возможных $TT
($Exp)%";
$msg=$msg+"<br>Ваша оценка: $Mark";
echo $msg;
// удаляем сессию.
session_unset();
exit;
}
```

Как вы видите, если пользователь ответил правильно на 90% (или более) вопросов, его оценка — 5, если на 75 и более (до 90), его оценка 4, 50..74 — оценка 3, менее 50% — оценка 2.

28.2.4. Пишем код системы проверки знаний

Для уменьшения размера кода сценария я разбил его на две части:

- первая (листинг 28.7) выводит форму регистрации (входа в СПЗ);
- вторая (листинг 28.8) — тестирует пользователя.

Листинг 28.7. Первая часть скрипта системы проверки знаний (авторизация пользователя)

```
<?
$LoginScript="$HTTP_HOST/smarttest/login.php";
$NewsFile="news.html";

// Устанавливаем cookie для индивидуального счетчика посещений
if(!isSet($STCount)) $STCount=0;
$STCount++;
SetCookie("STCount",$STCount,0x7FFFFFFF);

// Устнавливаем общий счетчик посещений
$f=fopen("counter.dat","a+");
flock($f,2);
$count=fread($f,100);
// Не засчитываем посещения со своего узла
$REMOTE_HOST=gethostbyaddr($REMOTE_ADDR);
if(!($HTTP_HOST===$REMOTE_HOST)) @$count=$count+1;
ftruncate($f,0);
fwrite($f,$count);
flock($f,3);
fclose($f);

// Читаем файл с новостями
if (file_exists($NewsFile))
  { $news = join( '', file($NewsFile) ); }
else $news="No news today";

// Это заголовок
echo "<html><head><meta http-equiv=\"Content-Type\"
content=\"text/html; charset=windows-1251\">";
echo "<meta name=\"Author\" content=\"(c) 2001-2002 Denis
Kolisnichenko\">";
```

```
echo "<title>SmartTest</title><link rel=stylesheet type=\"text/
css\" href=\"st.css\">";
echo "</head><body text=\"#000000\" bgcolor=\"#FFFFFF\"
link=\"#0000EE\" vlink=\"#551A8B\" link=\"#FF0000\"
background=\"smarttest.jpg\">";

// Тело
echo "<h1>Система проверки знаний<br>SmartTest</h1>";
echo "<table WIDTH=\"100%\" BGCOLOR=\"#336699\"><tr><td>";
echo "<font color=\"FFFFFF\"><font size=+1>Сегодня 27.01.02</
font></font></td></tr></table>";
echo "<table WIDTH=\"100%\" border=\"1\" cellspacing=\"0\"
cellpadding=\"3\" bordercolor=\"#336699\">";
echo "<tr><td width=\"50%\"><font color=\"#336699\"><font
size=+1>Новости</font></font></td>";
echo "<td width=\"50%\"><font color=\"#336699\"><font
size=+1>Регистрация</font></font></td></tr>";
echo "<tr><td>"; echo $news;
echo "<td><form action=http://$LoginScript>";
echo "<br> Логин  <input type=text name=\"user\"><br>Пароль
<input type=password name=\"pswd\">";
echo "<br><br><input type=submit value=\"Login\"><p>Вы должны
быть зарегистрированным";
echo "<br>пользователем! По вопросам регистрации обращайтесь к
<a";
echo "href=mailto:st-admin@host.com>администратору системы
SmartTest</a>";
echo "</td></tr></table>";
echo "<p>Вы посещали систему SmartTest $STCount раз(а). Всего
посещений: $count.";
echo "<p>&copy 2001-2003 <a href=http://dkws.narod.ru>Денис
Колисниченко</a>";
echo "<br>Все права защищены!</body></html>";

?>
```

Листинг 28.8. Вторая часть. Непосредственно система проверки знаний SmartTest

```
<?
// Открываем сессию smarttest
session_name("smarttest");
session_start();
session_register("Login");
session_register("FullName");
session_register("UserName");
```

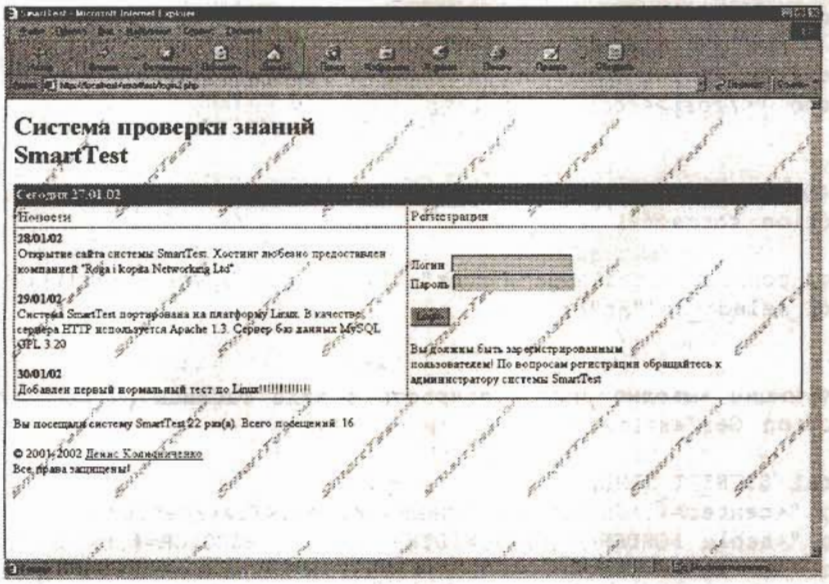


Рис. 28.7. *Вход в систему проверки знаний*

```
$Dir="/smarttest/logs";
$text=""; // объявим переменную и будем использовать ее как
глобальную

// Функция получает текст вопроса, выводит его и создает
// средства (checkbox'ы) для
// выбора правильных вариантов ответа

function GetQuestText()
{
global $Counter, $text;
$tmp=$Counter+1;
//echo "<title>SmartTest</title></head><body bgcolor=white>";
//echo "<h1>Система дистанционного контроля
знаний<br>SmartTest</h1>";
echo "<br><b>Вопрос $tmp</b><br>";
echo "$text";
echo "<br>";
echo "<form action=$SCRIPT_NAME><br>";

// Имена checkbox'ов - aN, где N от 1 до 6
echo "<input type=checkbox name=a1 value=\"1\">1 <input
type=checkbox name=a2 value=\"2\">2 ";
echo "<input type=checkbox name=a3 value=\"3\">3 <input
type=checkbox name=a4 value=\"4\">4 ";
```

```
echo "<input type=checkbox name=a5 value=\"5\">5 <input
type=checkbox name=a6 value=\"6\">6 ";
echo "<br><input type=submit value=\"Ответить!\">";
//echo "</form></body></html>";
}

// Функция подключения к серверу
function connect()
{
mysql_connect("localhost","root","") or die(mysql_error());
mysql_select_db("st");
}

// Функция выводит список вопросов в виде таблицы
function GetTestList()
{
global $SCRIPT_NAME;
echo "<center><i>Список доступных тестов:</i></center><br>";
echo "<table BORDER COLS=3 WIDTH=\"100%\" BGCOLOR=#CEFFCE>";
echo "<tr ALIGN=CENTER BGCOLOR=#3333FF>";
echo "<td><b><i><font color=#FFFFFF>Номер</font></i></b></td>";
echo "<td><b><i><font color=#FFFFFF>Название</font></i></b></
td>";
echo "<td><b><i><font color=#FFFFFF>Автор</font></i></b></td>";

$res=mysql_query("SELECT no,desk,autor FROM reg");
while ($Row=mysql_fetch_row($res))
{
 echo "<tr>";
 for($i=0; $i<mysql_num_fields($res); $i++)
 echo "<td><a href=$SCRIPT_NAME?tno=$Row[0]>$Row[$i]</a></td>";
 echo "</tr>";
 }
echo "</table>";
}

// Функция проверяет имя пользователя и его пароль. Если
// они правильны, она
// присваивает глобальной переменной Login значение 1

function Register()
{
global $user, $pswd, $Accept, $Login;
global $UserName, $FullName;
$Login=0;
$res=mysql_query("SELECT * from ureg");

while ($Row=mysql_fetch_row($res))
```

```
{
 if($Row[0]==$user)
  {
   if($Row[1]===md5($pswd))
        { $FullName=$Row[2];
          $UserName=$user;
          $Login=1;
          break; }
  }
}

/*   Основная программа              */

if ($Login==0) {

// Неправильные имя пользователя и/или пароль

 echo "<html><head><meta http-equiv=\"Content-Type\"
content=\"text/html;   charset=windows-1251\">";
 echo "<title>SmartTest - Access denied.</title></head><body
bgcolor=white>";
 echo "<h1>Система дистанционного контроля
знаний<br>SmartTest</h1>";
 echo "<center><img src=stop.jpg alt=STOP><h2>Access denied.</
h2>";
 echo "<p>Проверьте имя пользователя и пароль!";
 echo "</body></html>";
 unset($Accept);
 session_unset();
 global $Dir;
 global $REMOTE_ADDR;

// Нужно обеспечить запись в этот каталог
 @mkdir($Dir,755);
 $f=fopen("$Dir"."/access.log","a+") or Die("Cannot to create
access.log");
 flock($f,2);
 $dt=date("d.m.y H.i.s");
 if(PHP_OS=="Linux") $NL="\n"; else $NL="\n\r";
 fputs($f,"$dt Access denied for user $user [$pswd]. IP:
$REMOTE_ADDR $NL");
 flock($f,3);
 fclose($f);
 exit;
}
else
{
```

```
// Все Ок, выводим список тестов

echo "<html><head><meta http-equiv=\"Content-Type\"
content=\"text/html; charset=windows-1251\">";
echo "<title>SmartTest</title></head><body bgcolor=white>";
echo "<h1>Система дистанционного контроля знаний<br>SmartTest</
h1>";
echo "<B><I>Здравствуйте, $Row[2]!</B></I><BR>";

GetTestList();
}
return;
}

connect();
echo "login=$Login $UserName";
if(!isSet($Login)) { Register(); }
else
{

// Регистрируем в сессии служебные переменные
session_register("TestNo");
session_register("Arr");
session_register("Counter");
session_register("Table");
session_register("True");
session_register("Max");
session_register("Prev");
session_register("Diff");
session_register("Total");
session_register("TT");

// При первом запуске устанавливаем номер теста и
// инициализируем базу

if(!isSet($TestNo))
     {
      $Res=mysql_query("SELECT no,qmax,tbl FROM reg WHERE
no=$tno");
      $Row=mysql_fetch_row($Res);
      $Max=$Row[1];
      $Table=$Row[2];
      $Diff=$Row[3];
      $QDiff=0; $Total=0;
      $ATable=$Table."_a"; // таблица ответов
      echo $ATable;
```

```
      $R=@mysql_query("SELECT * FROM $ATable") or die("Invalid
SQL query [$ATable]");
      while($Rw=mysql_fetch_row($R))
      { if($Rw[0]===$UserName)
         {
          echo "Вы уже проходили выбранный тест<br>";
          echo "Ваша оценка $Rw[1]. Вы правильно ответили на
$Rw[2] вопросов(а)";
          GetTestList();
         exit;
         } // if($Rw)
      } //While($Rw);

      $TestNo=tno;
      $Counter=0;

      // проверяет все ли разные элементы в массиве
      function checkarr($Arr)
      {
           // разные
           $Res=1;
            for ($i=0; $i<10; $i++)
             for ($j=$i+1; $j<10; $j++)
               if($Arr[$i]===$Arr[$j]) { $Res=0; break(1); }
       return $Res;
      }

      mt_srand(time()+(double)microtime()*1000000);
      for ($i=0;$i<9;$i++) $Quest[]=mt_rand(0,100);
      while(checkarr($Quest)==0)
        {
          for ($i=0;$i<9;$i++) $Quest[$i]=mt_rand(0,100);
        } // while

      // массив сгенерирован, пишем его ...
     $Arr=serialize($Quest);
     // сейчас нужно выдать первый вопрос...
     $Res=mysql_query("SELECT * FROM $Table WHERE
no=$Quest[$Counter]");
     $Row=mysql_fetch_row($Res);
     $Prev=$Row[2];
     $text=$Row[1];
     $Diff=$Row[3];

     GetQuestText();
     } // if (!isSet)
else // уже не первый запуск, $TestNo установлена
```

```
// пришел ответ на предыдущий вопрос
  {
   if(!isSet($a1)) $a1=""; if(!isSet($a2)) $a2="";
if(!isset($a3)) $a3="";
   if(!isSet($a4)) $a4=""; if(!isSet($a5)) $a5="";
if(!isSet($a6)) $a6="";
   $Answer=$a1.$a2.$a3.$a4.$a5.$a6;
   if ($Answer===$Prev)
   {
    $True=@$True+1;
    $Trusted='Y';
    If ($Diff=='Y') $Total=$Total+3;
    else $Total=$Total+1;
   } else $Trusted='N';

   If ($Diff=='Y') $TT=$TT+3;
   else $TT=$TT+1;

   if($Counter===9) {
   // пора бы и закончить тест
   $ATable=$Table."_a";
   $Exp=($Total*100)/$TT;
   if ($Exp>=90) $Mark=5;
   else if($Exp>=75) $Mark=4;
   else if($Exp>=50) $Mark=3;
   else if($Exp<50) $Mark=2;

   mysql_query("INSERT INTO $ATable
VALUES('$UserName','$Mark','$True')");
   $msg="<br>$FullName, вы ответили правильно на $True вопросов(а).";
   $msg=$msg+"<br>Вы набрали $Total баллов из возможных $TT
($Exp)%";
   $msg=$msg+"<br>Ваша оценка: $Mark";
   echo $msg;
   // удаляем сессию.
   session_unset();
   exit;
   }
   $Quest=unserialize($Arr);
   $Res=mysql_query("SELECT * FROM $Table WHERE
no=$Quest[$Counter]");
   $Row=mysql_fetch_row($Res);
   $Prev=$Row[2];
   $text=$Row[1];
```

```
    $Diff=$Row[3];
    $Counter=$Counter+1;
    GetQuestText();
  }
} // от самого первого else
?>
```

Как и обещал, домашнее задание: напишите сценарий, который будет создавать новые таблицы тестов, регистрировать новые тесты, в общем, работать с таблицей регистрации тестов. Можно также написать сценарий управления таблицей пользователей, который, кроме удаления, добавления и изменения записей, позволял бы просматривать результаты пользователя по всем пройденным тестам.

Если же вам лень или нет времени, обойтись можно, используя средство **phpMyAdmin**. Обычно его предоставляют некоторые хотинг-провайдеры.

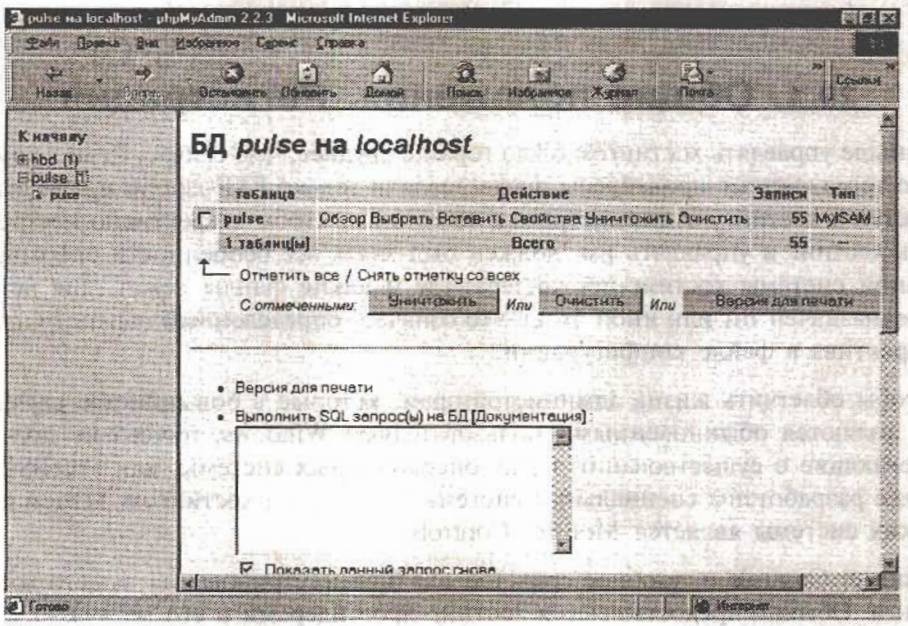


Рис. 28.8. phpMyAdmin

Глава 29

Автоматизированное управление параметрами хостинга. Система Monster Controls

29.1. Общие положения. Регистрация

Раньше управлять хостингом было гораздо труднее, чем сейчас. Дело в том, что тогда хостинг-провайдеры предоставляли только FTP-доступ и/или ssh-доступ к хостингу. Администратор, то есть вы, чтобы эффективно настроить хостинг и управлять им, должен был знать все особенности операционной системы хостинг-провайдера: где и какие файлы лежат, для чего предназначен тот или иной файл, что означает определенный параметр или директива в файле конфигурации.

Чтобы облегчить жизнь администраторам, которые в большинстве случаев являются обыкновенными пользователями Windows, ничего не подозревающие о существовании таких операционных системы, как FreeBSD, были разработаны специальные системы управления хостингом. Одной из таких системы является Monster Controls.

Monster Controls — удобное средство для администрирования вашего хостинга. Обычно предоставляется хостинг-провайдерами и его не нужно устанавливать самостоятельно. Monster Controls используются далеко не всеми хостерами, но встречаются довольно часто, особенно на хостингах США.

Для активизирования Monster Controls, зайдите на свой сайт так:

```
http://www.имя.com/monstercontrols/index.cgi
```

После этого «монстры» предложат вам ввести имя пользователя и пароль (см. рис. 29.1).

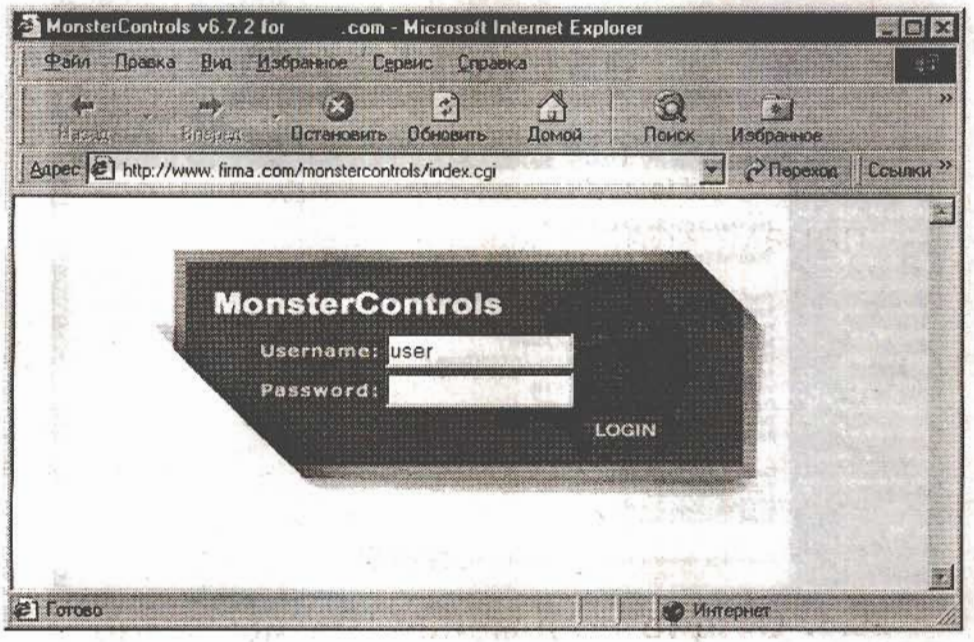


Рис. 29.1. Регистрация

29.2. Использование Monster Controls

Основное меню. Статистика

Сразу после того, как вы вошли в систему Monster Controls, вы увидите различную информацию о своем домене (рис. 29.2):

- Доменное имя (Your domain name).
- Имя сервера (Your server name).
- Путь SSL-соединения для вашего домена (Your SSL path for your domain).
- Имя каталога для HTML-документов (The path to your web directory).
- IP-адрес вашего домена (Your domain's current IP address).
- Использованный Web-трафик за этот месяц (Web traffic so far this month).
- Использованный FTP-трафик за этот месяц (FTP traffic so far this month).
- Общий трафик (Web + FTP).
- Текущий размер вашего сайта (Current disk space usage).
- Количество файлов и каталогов (Total number of files & directories).

В этом же окне вы можете нажать кнопку «Disk Usage Details», чтобы получить детальную информацию об использовании диска. А нажав на кнопку «Install now», вы установите расширения FrontPage, если они, конечно, вам нужны.

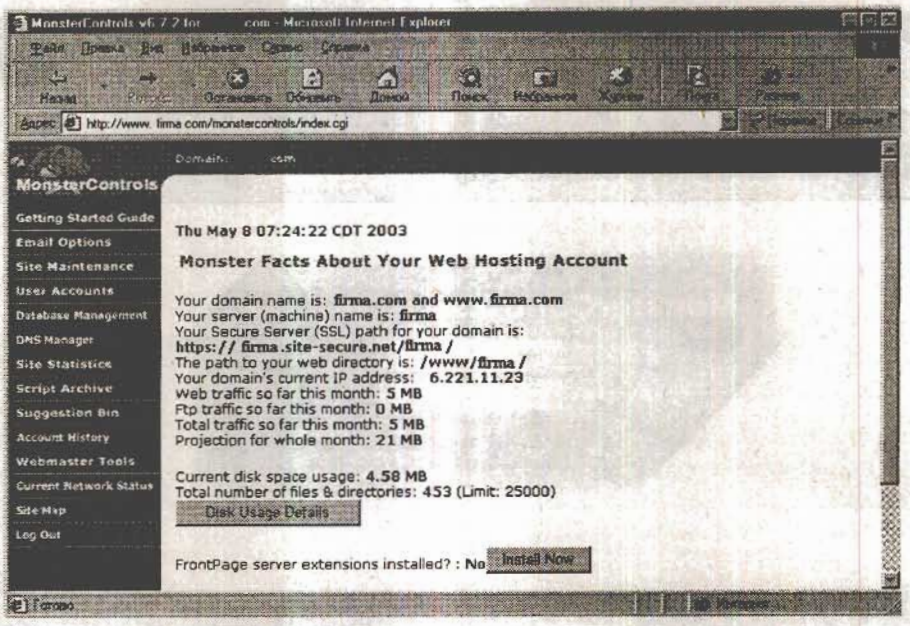


Рис. 29.2. *Статистика*

По просьбе администратора домена я заменил реальное имя домена абстрактным именем `firma`. IP-адрес я также изменил, выбрав при этом случайный адрес.

Слева вы видите меню — разделы системы Monster Controls:

- **Email options** — настройки электронной почты.
- **Site Maintenance** — управление сайтом.
- **User accounts** — учетные записи домена.
- **Database Management** — управление СУБД MySQL.
- **DNS Manager** — управление системой DNS для вашего домена.
- **Site Statistics** — подробная статистика о сайте.
- **Script Archive** — архив всевозможных сценариев, которые могут вам пригодиться.
- **Account History** — журнал учетной записи.
- **Webmaster tools** — вспомогательные утилиты для Web-мастера.
- **Current Network Statistics** — статистика о работе сети.
- **Log out** — выход из системы.

Теперь рассмотрим подробнее некоторые разделы.

Раздел Email options

Здесь находятся настройки электронной почты вашего домена, вы можете:

- Прочитать почту и отправить новое сообщение (webbased email).
- Установить перенаправление почты (email redirects).
- Настроить автоответчики (simple auto responders).
- Создать списки рассылки (mail lists).

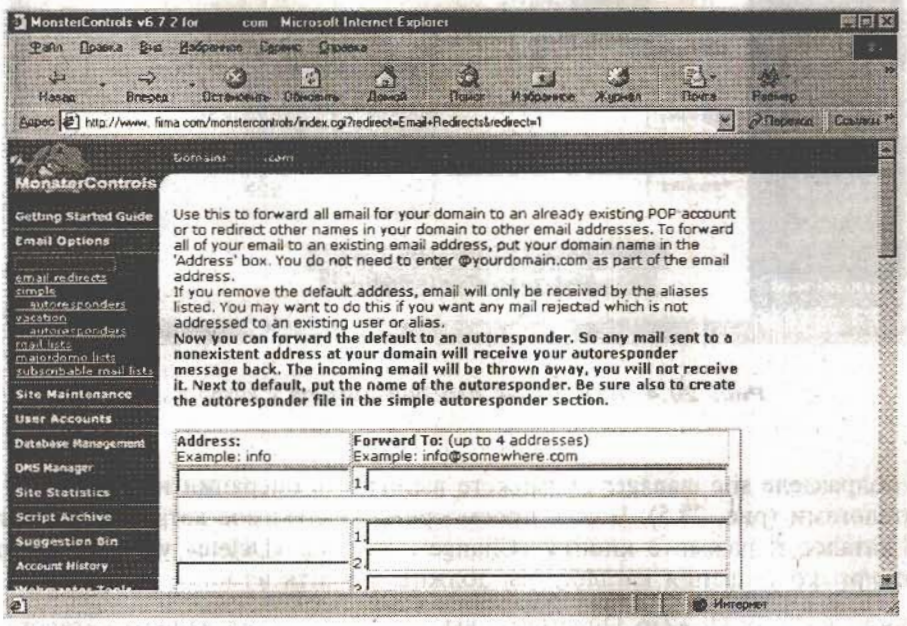


Рис. 29.3. Установка перенаправления для почты

Раздел Site Maintenance

Наиболее часто вы будете посещать именно этот раздел, поскольку он содержит команды для основного (можно сказать, ежедневного) администрирования сайта. Здесь вы можете:

- Назначить пароли для доступа к определенному каталогу вашего сервера, а также указать пользователей, которым разрешен доступ к этому каталогу (см. рис. 29.4).
- Изменить права доступа к любому файлу или каталогу, удалить или создать файл или каталог, заархивировать файл или каталог в формате zip, и многое другое.

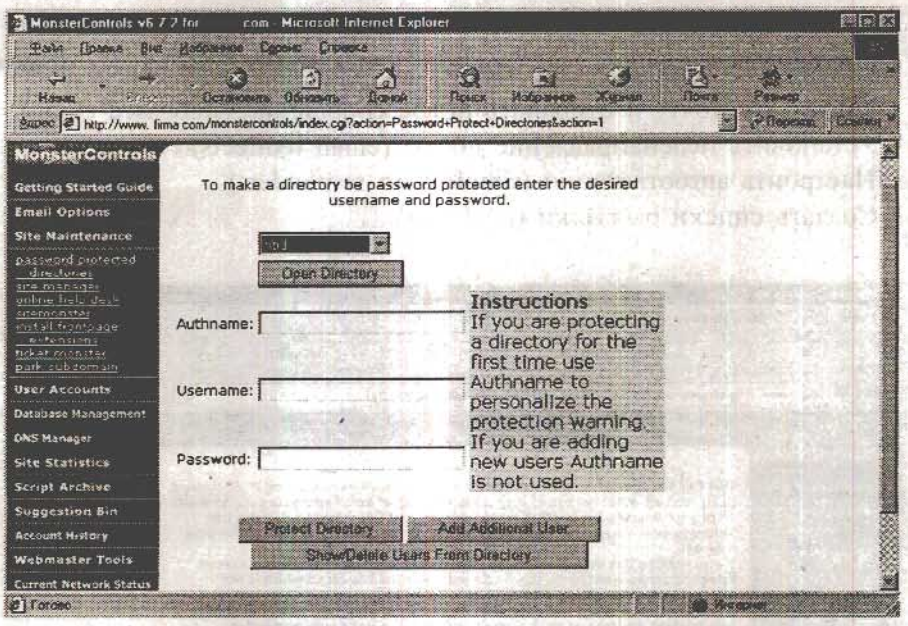


Рис. 29.4. *Назначение паролей доступа к каталогам*

В подразделе **site manager** вы можете выполнять операции над файлами и каталогами (рис. 29.5). Чтобы просмотреть содержимое каталога, выберите каталог и нажмите кнопку «Change». Кнопка «Delete» удаляет пустой каталог: до удаления каталога вы должны удалить из него все файлы.

Нажав на кнопку «Zip Directory», вы создадите архив вашего каталога в формате Zip, а кнопка «Zip Whole Site» предназначена для архивирования всего сайта.

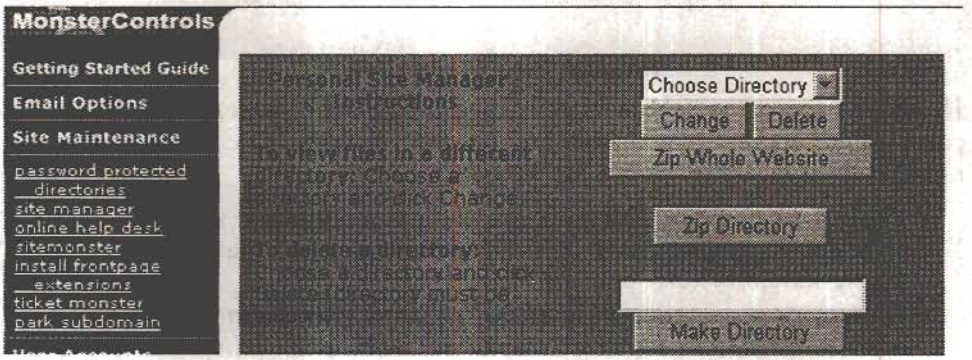


Рис. 29.5. *Операции над каталогами*

Если вы хотите создать каталог в текущем каталоге, введите его имя и нажмите кнопку «Make Directory». Если текущий каталог не выбран, каталог будет создан в вашем домашнем каталоге.

Операций над файлами значительно больше (рис. 29.6):

- Создание файла (кнопка «New File»).
- Просмотр и редактирование файла («View» и «Edit», соответственно).
- Перемещение и переименование («Move» и «Rename»).
- Удаление («Delete»).
- Архивирование/деархивирование («Zip»/«UnZip»).
- Загрузка файла («Browse» или «Обзор», а затем «Upload»).
- Создание копии («Duplicate», «Copy»).
- Установить права доступа к файлу или каталогу («Set on File», «Set On Directory»).
- Добавить счетчик («Add Counter»).

Остановимся подробнее на следующих операциях с файлами: копирование, перемещение, переименование и загрузка (upload).

Для копирования файла в этот каталог выберите файл, введите новое имя в поле **New Name** и нажмите кнопку «Duplicate». Если вам нужно скопировать файл в другой каталог, выберите этот каталог, выберите файл и нажмите кнопку «Copy».

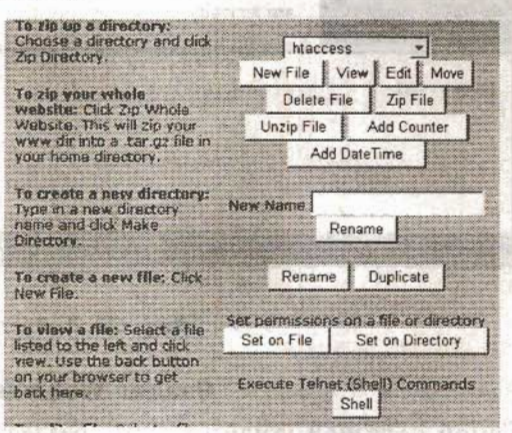


Рис. 29.6. *Операции над файлами*

Для переименования файла введите новое имя в поле **New Name** и нажмите кнопку «Rename». Для перемещения файла следует выбрать каталог, в который вы собираетесь переместить файл и нажать кнопку «Move».

Чтобы загрузить файл, нажмите кнопку «Browse» (Обзор), выберите локальный файл и нажмите кнопку «Upload». Файл будет загружен в выбранный вами каталог.

Еще одна довольно интересная операция — это установка счетчика. Система автоматически добавит счетчик в HTML-файл: вам не нужно его писать вручную, тратя свое время.

Вы устали от всех этих кнопок? Хочется вводить привычные команды (ls, cd, cat, rm, cp, ...)? Нажмите кнопку «Shell» — появится окно,

в которое можно вводить команды, результат выполнения которых вы увидите в нижней части окна (см. рис. 29.7). Следует заметить, что набор команд ограничен.

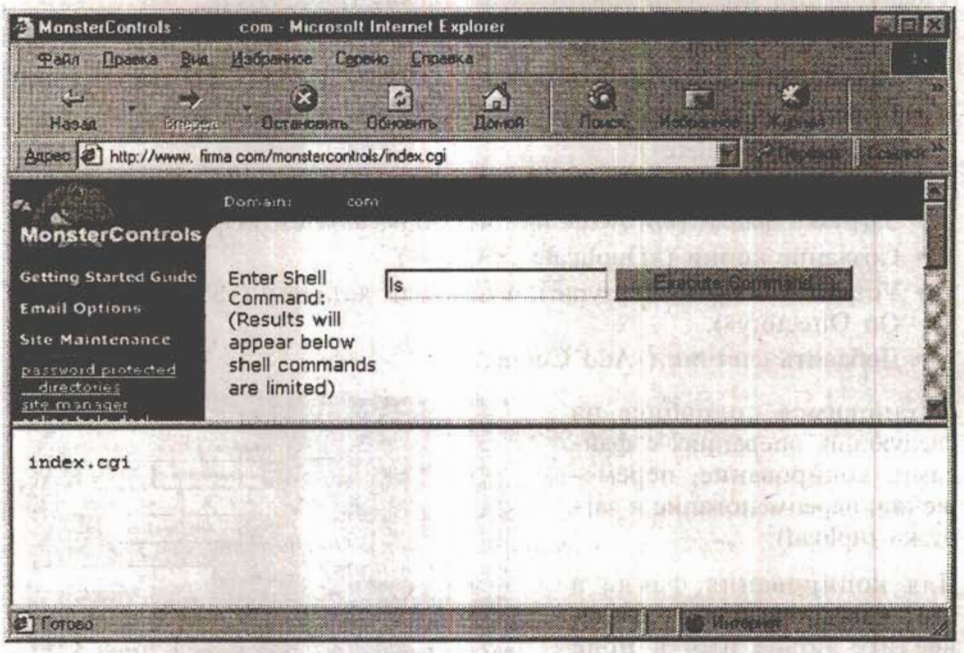


Рис. 29.7. Ввод команд

Раздел Webmaster tools

Самой полезной утилитой этого раздела оказалась утилита редактирования расписаний демона crond (см. рис. 29.8).

Выберите день и время, когда должна быть запущена та или иная программа. Я заметил одну интересную особенность: вы не можете передать программе какой-нибудь параметр, то есть для запуска PHP-программы вы не можете использовать команду: `/usr/bin/php file.php`. Интерпретатор будет запущен, но вот на вход ему файл `file.php` направлен не будет. Поэтому, если вам нужно запустить программу с параметрами, создайте сценарий, который будет выполнять эту команду, например,

```
#!/bin/bash
/usr/bin/php file.php
```

Потом в расписание crond нужно будет добавить этот сценарий. Не забудьте перед этим сделать сценарий исполнимым.

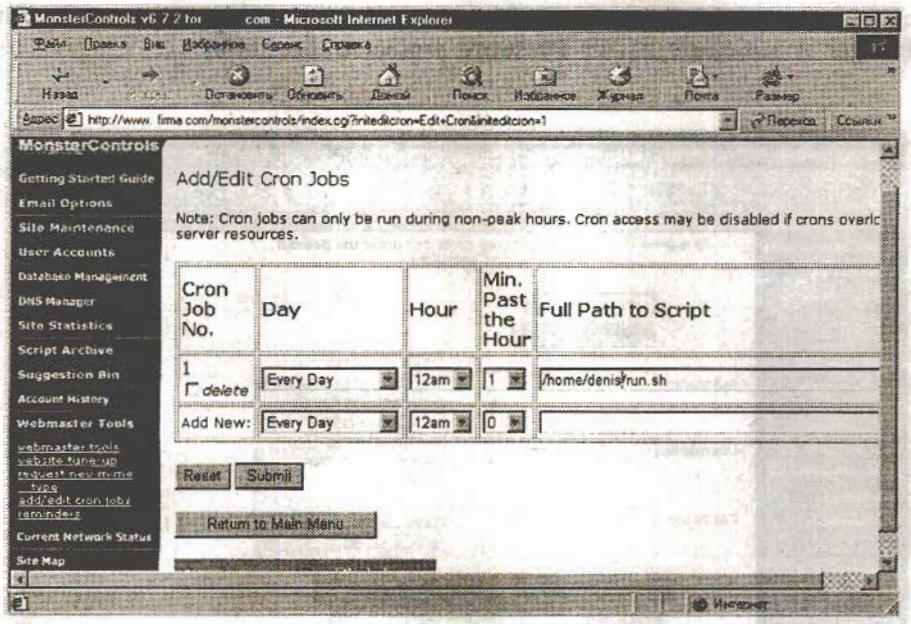


Рис. 29.8. *Редактирование расписания crond*

Система администрирования хостинга Monster Controls существенно облегчает труд администратора, сокращая время, потраченное на администрирование вручную, примерно раза в два. Хотя, вам придется потратить от получаса до двух часов на освоение системы. Но когда вы привыкните к «монстрам», вам уже не захочется редактировать файлы конфигурации в редакторе **vi**, зарегистрировавшись по **ssh**.

29.3. Делаем мемберскую зону

Мемберская зона доступна только определенным пользователям сайта, все остальные пользователи не имеют к ней доступ.

Создать закрытую (мемберскую) зону позволяет практически каждое средство управления контентом. В крайнем случае, можно настроить сервер Apache так, чтобы он аутентифицировал пользователей, но в этом случае вам нужен доступ к файлам конфигурации сервера Apache.

Сейчас мы рассмотрим, как сделать закрытую зону с помощью Monsters Controls. О том, как сделать то же самое с помощью PHP-Nuke, написано в соответствующей главе.

Зарегистрируйтесь в системе Monsters Controls. Откройте меню **Site Manager** и выберите пункт **Password protected directories**.

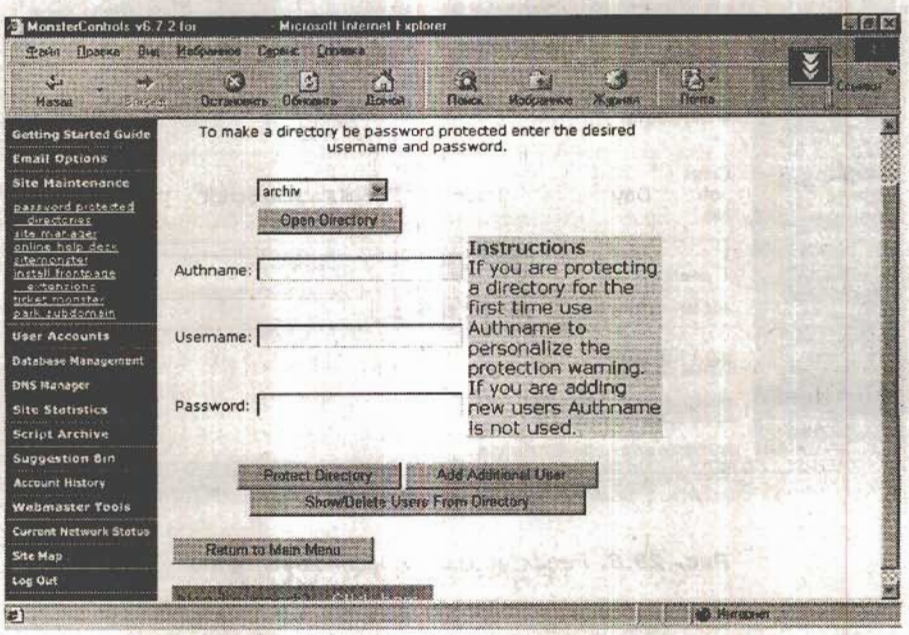


Рис. 29.9. *Site Manager*

Выберите каталог, который вы хотите защитить, введите имя пользователя и его пароль и нажмите кнопку **Protect Directory**. Если вы хотите добавить дополнительного пользователя, нажмите кнопку **Add additional User**. Для просмотра списка пользователей, которые уже получили доступ, нажмите кнопку **Show/Delete Users From Directories**.

Глава 30

Поддержка COM и Java

PHP, начиная с четвертой версии, поддерживает технологии COM и Java. Такой поддержкой может похвастаться не каждый язык программирования. Поскольку обе технологии не только встречаются, но и активно используются в Internet, в книге им уделено немного внимания.

На Windows-платформе PHP обеспечивает поддержку COM — это означает, что вы можете управлять практически любыми Windows-приложениями. Ясное дело, PHP должен быть установлен тоже под Windows. Если у вас Linux, этот материал не представляет для вас никакой практической ценности.

30.1. Технология COM

Технология COM (Component Object Model — модель компонентных объектов) позволяет определить объект и работать с ним другим программам и объектам с помощью методов, которые стандартизированы на двоичном уровне. Данная модель использована в технологиях DCOM и ActiveX (рис. 30.1).

Примечание.
Все вышеупомянутые технологии были разработаны корпорацией Microsoft.

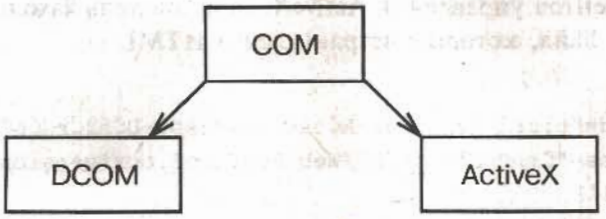


Рис. 30.1. Использование модели COM

Модуль DCOM (Distributed COM — распределенная COM-модель) позволяет программам обмениваться данными с помощью удаленного вызова OLE-объектов, которые расположены на различных системах.

ActiveX — технология Microsoft, аналогична технологии COM, но содержит средства работы в Internet. Позволяет загружать на компьютер пользователя любой документ из библиотеки Web-узла вместе со всеми необходимыми элементами для его обработки.

Используя COM, мы можем внедрить в свою программу любой OLE-объект. Например, вы пишите бухгалтерскую программу, внутри которой нужно отредактировать какой-нибудь документ, например, ведомость о получении зарплаты. Но времени писать собственный редактор у вас нет, к тому же пользователи очень хорошо знают Word и умеют с ним работать, следовательно, не нужно писать справочник по своему текстовому процессору. Вы внедряете в свою программу OLE-объект — приложение Word, которое при необходимости будет активизировано. При этом нужно учитывать два фактора:

- Вызываемое приложение должно быть установлено у пользователя. Если Word у пользователя не установлен, ваш редактор работать не будет.
- Данная технология работает довольно медленно, так что пользователи будут жаловаться на низкое быстродействие вашей программы.

ActiveX — это элементы управления. Вы можете написать свой элемент управления, а другой пользователь или программист будет его использовать в своей программе. Имеется возможность встраивания элементов управления в следующие пакеты:

- MS Office
- MS Visual C++
- Borland Delphi

Учитывая то, что все эти технологии — COM, DCOM и ActiveX — изобретение ума Microsoft, фактически вы привязываете себя к платформе Microsoft. Помню, когда-то я написал систему проверки знаний с использованием элементов управления ActiveX: пользователь заходил на сайт и вызывал OCX-файл, который встраивался в HTML-код так:

```
<OBJECT
     classid="clsid:91F16385-A054-11D4-BBDF-DC5BCBCCEC6C"
     codebase="C:/Projects/RE/Web/RegForm1.ocx#version=1,0,0,0"
     width=471
     height=229
     align=center
     hspace=0
     vspace=0
>
```

Браузер, получив соответствующий код, встраивал элемент управления. Потом пользователь может работать с ним, как с обыкновенной программой в окне Internet Explorer. Получается как бы программа в программе.

При использовании ActiveX я столкнулся со следующими проблемами:

- Использование dll оказалось затруднительно — вместе с OCX файлом нельзя было передать библиотеки, приходилось их отдельно устанавливать на каждый компьютер пользователя.
- Без использования dll размер OCX-файла вырос до неприличных размеров — более 600 Кб. Вы захотите ждать загрузки элемента управления размером 623 Кб при модемном соединении в 28,8 К?

Преодолев эти проблемы, я понял, что проект получился не универсальным:

- Пользователь должен использовать только MS Internet Explorer. Конечно, есть возможность использовать ActiveX в Netscape Navigator, но все «попавшиеся мне под горячую руку» OCX-плагины работали некорректно.
- Кроме того, что пользователя привязывали к ОС Windows, так еще и к браузеру. А что делать, если пользователь предпочитает Mozill'у или Оперу?
- Сами понимаете, сервер также должен быть под управлением Windows. И не просто под управлением Windows, но еще должен быть установлен именно IIS (Internet Information Server)

Исходя из всего этого, было принято решение переписать проект на PHP. При этом проект получился очень универсальным: мы не привязываемся к платформе Microsoft — как клиент, так и сервер могут быть под управлением любой операционной системы — Windows/Linux/FreeBSD.

Надеюсь, я отговорил вас использовать COM в своих проектах.

В листинге 30.1 приведен пример работы COM-модуля. При этом данный сценарий производит следующие действия:

- Создает экземпляр WordApp.
- Отображает окно приложения.
- Создает новый документ.
- Вводит текст.
- Сохраняет введенный текст под новым именем test_com_php.doc.

Листинг 30.1. Поддержка COM в Windows

```
<?php
$word = new COM("word.application") or die("Невозможно вызвать
WordApp");
print "Word is running, version {$word->Version}\n</br>";
// Отображаем окно Word'a
```

```
$word->Visible=1;
// Создаем новый документ
$word->Documents->Add();
// Вводим текст
$word->Selection->TypeText("This is a test...");
// Сохраняем документ
$word->Documents[1]>SaveAs("test_com_php.doc");
// Закрываем окно
$word->Quit();
?>
```

30.2. Поддержка Java

Поддержка Java является безусловно очень полезной функцией, позволяющей нам использовать уже готовые классы и сервлеты Java.

Поддержка Java не является встроенной, поэтому нужно установить соответствующее расширение. Если вы работаете под Linux, вам нужно перекомпилировать PHP, включив поддержку Java. Для Windows уже есть готовое, откомпилированное, расширение в виде динамической библиотеки. Добавьте в ваш конфигурационный файл секцию java:

```
[java]

java.class.path = .\php_java.jar
java.home = c:\jdk
java.library = c:\jdk\jre\bin\hotspot\jvm.dll
java.library.path = .\
```

А также раскомментируйте строку:

```
extension=php_java.dll
```

Проверить, включена ли поддкржка Java, позволяет функция **phpinfo()**. Если поддержка Java включена, проверим ее работу. Создайте следующий класс (см. листинг 30.2), описывающий линию. Наш класс содержит четыре атрибута (координаты линии и ее тип) и четыре метода — два конструктора, метод установки координаты и тестовый метод sayHello, возвращающий знаменитую фразу Hello, world!

Листинг 30.2. Класс Line

```
public class Line {

// атрибуты
public int x1;
public int x2;
public int y1;
public int y2;
```

```
public int type;

// Конструктор по умолчанию
public Line() {
setCoordinates(0,0,0,0);
this.type = 1 ;
}

// Конструктор
public Line(int x1,int y1, int x2, int y2) {
setCoordinates(x1,y1, x2, y2);
this.type = 1 ;
}

// Методы

public void setCoordinates(int x1, int y1, int x2, int y2) {
this.x1 = x1;
this.y1 = y1;
this.x2 = x2;
this.y2 = y2;
}

public int get_x1 { return this.x1; }
public int get_x2 { return this.x2; }
public int get_y1 { return this.y1; }
public int get_y2 { return this.y2; }

public sayHello String() {
return "Hello, world!";
}

}
```

Следующий PHP-сценарий (листинг 30.3) использует поддержку класса Line

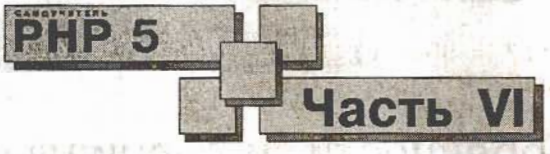
Листинг 30.3. Использование класса Line

```
<?

// Создаем объект класс Line
$myLine = new Java("Line");
echo "<br>Line created at (0,0,0,0) ...";
// Устанавливаем координаты
echo $myLine->setCoordinates(5,5,7,77);
// Выводим Hello World
echo $myLine->sayHello();

?>
```

На экране вы увидите:

```
Line created at (0,0,0,0) ... Hello, World!
```

Зачем нам поддержка Java? Предположим, что вы пишите коммерческий сценарий, стоящий уйму денег. Естественно, что вы не хотите показывать исходный код. В случае с PHP, весь код виден «невооруженным глазом». Java позволяет скрыть код от пользователя. Объединив возможности Java с мощью PHP, вы получите очень мощную смесь. Открытые функции вы можете писать на PHP, а все, что нужно скрыть ото всех — на Java. Так как средства Java для доступа к базам данных, мягко говоря, не совсем удобны, я бы порекомендовал писать на PHP именно работу с базами данных. Функции mysql_connect() и mysql_query() известны всем, и вам нечего скрывать, ведь так? Возможен и второй, непрофессиональный, подход к использованию классов Java. Как правило, для профессионального программиста нет особой разницы на каком языке. Но начинающий программист начинает решать задачу, исходя из возможностей языка. Если вы хорошо знаете Java, но нужно писать на PHP (возможно, браузер пользователя не поддерживает Java, а вывести результат все равно нужно) — пишите основную обработку информации на Java, а ее вывод — на PHP. Если, конечно, вам так будет проще.



Специальные технологии PHP 5

Глава 31. Особенности и специальные возможности PHP 5

Глава 32. SQLite

Вместо заключения: Горячая десятка ошибок

Глава 31

Особенности и специальные возможности PHP 5

31.1. Обзор нововведений

Общая информация

Совсем недавно вышла новая, пятая, версия PHP. На момент написания этих строк доступна только альфа-версия, но, я надеюсь, к моменту выхода из печати будет доступна стабильная версия. Как обычно, загрузить ее можно по адресу `http://www.php.net`.

Рассмотрим самые основные изменения в пятой версии. Прежде всего, данная версия использует вторую версию Zend (Zend Engine 2), что позволяет еще больше расширить возможности самого интерпретатора PHP.

Zend Engine — это основа PHP, поэтому нужно говорить не столько об изменениях в PHP, сколько о новшествах Zend 2.0.

Добавлена поддержка SQLite, причем теперь SQLite поддерживается по умолчанию. SQLite — это небольшая система управления данных, которая обеспечивает выполнение базовых операций над таблицами. Может быть, в скором времени SQLite станет таким же популярным, как и MySQL.

Третье изменение коснулось производительности встроенных функций. Теперь стандартные функции PHP работают на 40% быстрее.

Развитие языка разметки страниц XML также не осталось незамеченным разработчиками PHP: теперь поддержка XML стала полной. Кроме всего прочего, поддерживаются новые расширения DOM и XML.

Новые директивы настроечного файла php.ini

В файле php.ini появились новые директивы:

- **session.hash_function** — функция хеширования сессий;
- **session.hash_bits_per_character** — число бит в символе (данный параметр используется при хэшировании сессии);
- ***mail.force_extra_parameters*** — *дополнительные почтовые параметры;*
- **register_long_arrays** — регистрация длинных массивов.

Первые две управляют процессом хеширования, вторая позволяет указывать расширенные параметры почты, а третья — регистрировать большие (или длинные — как кому больше нравится) массивы.

Новые механизмы поддержки потоков

Поддержка потоков претерпела много изменений:

- Быстрее стали работать функции **readfile()**, **fpassthru()** и некоторые внутрение функции для работы с потоками.
- Появилась функция **stream_socket_client()** — более мощный аналог **fsockopen()**.
- Новая функция **stream_socket_server()** создает сокет сервера.
- Функция **stream_socket_accept()** принимает соединение клиента.
- Функция **stream_socket_get_name()** получает локальное или удаленное имя сокета.
- Очень полезная функция **stream_copy_to_stream()**, которая копирует поток в поток.
- Появилась функция чтения информации из потока до определенного символа или строки — **stream_get_line()**.
- Поддерживается динамическая загрузка OpenSSL — шифрования потоков «на лету».
- Функции **opendir()**, **stat()** и **unlink()** теперь могут работать с FTP-соединениями. Для этого в качестве имени файла укажите префикс **ftp://**.

Новые функции графической библиотеки GD

Какой же Web-программист не рисовал счетчик посещения? Хоть раз в жизни, да было. Библиотека GD предназначена для поддержки графики в PHP. Изменения коснулись и этой библиотеки:

- Появился фильтр изображений, позволяющий применять различные фильтры к картинкам, imagefilter().
- Появились новые функции рисования.

Теперь число параметров функции **fgetss()** может быть любым. Также была изменена функция **get_extension_funcs()**, возвращающая список встроенных в механизм Zend функций, если вы укажите «zend» в качестве параметра.

Разные новые функции

Добавлена поддержка библиотеки **libmcrypt** версии 2.5.6 или выше. Начиная с версии 2.5.6, данная библиотека обеспечивает более надежный уровень защиты.

Появились новые функции для работы с iconv. Правда, не знаю, будете ли вы их использовать, но вот список новых функций:

iconv_strlen()
iconv_substr()
iconv_strpos()
iconv_strrpos()
iconv_mime_decode()
iconv_mime_encode()

Появились новые интересные функции:

- **proc_nice()** — изменяет приоритет текущего процесса.
- **idate()**, date_sunrise() и date_sunset() — новые функции для работы с датами.
- **strpbrk()** — ищет в строке список символов.
- **get_headers()** — возвращает заголовки, возвращаемые сервером при указании какого-нибудь URL.
- **str_split()** — разбивает строку на массив символов.
- **array_walk_recursive()** и **array_combine()** — новые функции для манипуляции с масивами.

Новыми параметрами стали обладать следующие функции (все параметры необязательные, поэтому сценарии, написанные на PHP 4 с использованием этих функций, будут работать):

- Параметр функции **openssl_sign()** позволяет выбрать алгоритм шифрования (хэширования).
- Благодаря новому параметру, функции **sha1()**, **sha1_file()**, **md5()** и **md5_file()** теперь могут возвращать результат в бинарном виде.
- Новый параметр функции **mkdir()** позволяет создавать каталоги рекурсивно.
- Дополнительный параметр функции **file()** не включает в результат символ \n в конце каждой строки, а также пустые строки.

Полностью переделана функция **range()**: теперь она может работать с действительными числами (float) и обладает необязательным параметром «шаг», позволяющий указать шаг диапазона.

Не прошли мимо и IPv6. При работе с IP-адресами поддерживается протокол IP версии 6 (IPv6).

Исправлено много ошибок в различных функциях, в том числе **dirname()**, ***strip_tags()***, ***pathinfo()***,**mb_send_mail()**. Особенно приятно то, что функция **is_executable()** теперь корректно работает под Windows. Данная функция возвращает TRUE, если переданный ей файл (точнее, его имя) является исполнимым, то есть программой.

Как вы видите, разработчики PHP постарались, выпустив пятую версию. Однако радоваться пока еще рано — вы уже можете установить новую версию и работать с ней. Однако, хостинг-провайдеры, как правило, не спешат с обновлением версий. Поэтому вам придется подождать месяц-другой после выхода стабильной пятой версии PHP, пока она будет установлена на сервера провайдеров. В книге я старался писать сценарии, чтобы они работали, как в четвертой, так и в пятой версии PHP.

Все, о чем мы только что говорили, дает только поверхностное понятие обо всех изменениях, коснувшихся новой версии PHP. Чтобы «докопаться до истины», нам нужно внимательно рассмотреть новую версию Zend 2.0, которая лежит в основе PHP 5. Перед чтением следующего материала рекомендую ознакомиться с объектно-ориентированным программированием в PHP.

31.2. Zend Engine 2.0

Zend — это движок PHP, это его мозг, его основа. PHP — это всего лишь надстройка над Zend. PHP 4 использовал Zend версии 1.0, которая имела много ограничений, о которых мы поговорим чуть позже. Версия Zend 1.0 — это не самая первая версия движка PHP. Третья версия PHP использовала «половинную» версию Zend — Zend Engine 0.5, которая была еще больше ограничена в своих возможностях, чем «копейка» (версия 1.0).

Какие же функции выполняет движок? Он обеспечивает интерфейсы для функций, а также выполняет синтаксический разбор конструкций сценария, то есть реализует синтаксис языка PHP.

31.2.1. Новая объектная модель и объектные дескрипторы

Zend 1.0 (и его предыдущая версия — 0.5) нерационально использовал память при работе с объектами, потому что объекты являлись языковыми величинами. Это означает, что, когда вы выполняете операцию присваивания для объекта, объект обрабатывался как другие примитивные типы — переменные, списки, то есть объект копировался целиком. В новой объектной модели, которая используется в Zend 2.0, используется технология объектных дескрипторов, напоминающая аналогичную технологию в языке Java. Теперь вы работаете не с объектом, а его дескриптором. Это позволяет не только сэкономить память, что не так важно для нас, поскольку объемы памяти, установленной на серверах, давно перевалили объем 1 Гб, но получить полный контроль над созданием и дублированием объектов.

Когда вы создаете новый объект, вы получаете его дескриптор, но не в коем случае не сам объект. Потом, при передаче объекта в качестве ар-

гумента функции, копируется опять же таки дескриптор, а не сам объект. Сам объект *никогда* не копируется и не дублируется.

Попробуем создать новый объект:

```
$ob = new Some_Class();
$ob->method();
```

Если раньше, в четвертой версии PHP (первой версии Zend), данный код создавал объект, то теперь создается только дескриптор объекта класса **Some_Class()** и вызывается один из его методов.

Разберем следующий участок кода:

```
class Some_Class()
{
    function set($value) { $this->member=$value; }
    function get() { return $this->member; }
…
}

function f1($obj) { $obj->set("f1_value");

$ob = new Some_Class();
$ob->set("my_value");
f1($ob);
echo $ob->get();
```

Метод **set()** позволяет установить значение поля $member (это строка), а метод **get()** — возвращает это значение. Сначала мы устанавливаем значение этого поля равным «my_value». Потом вызываем функцию **f1()**, которая устанавливает значение этого поля равным «f1_value». В предыдущей версии Zend функция **f1()** работает с копией объекта, с его дублем, поэтому последний оператор напечатает «my_value», поскольку функция **f1()** изменила не объект $ob, а его копию, которая создана Zend 1.0 при передаче аргумента функции **f1()**.

Во второй версии Zend, использующей новую объектную модель, сценарий выведет строку «f1_value», поскольку функция **f1()** теперь работает не с копией объекта $ob, а с его дескриптором, по которому она «добирается» до самого объекта $ob. Данное нововведение не только экономит память, но и увеличивает производительность работы с объектами.

31.2.2. Механизм возврата функциями объектов

Данный механизм был и в первой версии Zend, однако во второй версии он значительно переработан и улучшен.

Механизм возврата объектов в первой версии Zend был, мягко говоря, плохим. Во-первых, невозможно передать объект с помощью инструкции return. Во-вторых, приходилось использовать довольно специфический синтаксис, например, **$res = &$f1()**, чтобы **f1()** вернула объект по ссылке. В-третьих, невозможно вернуть объект по ссылке, присвоив его аргументу функции, переданному по ссылке.

Вторая версия Zend сняла все эти ограничения, и вы смело можете использовать инструкцию return для возврата объекта, а также присваивать объект переданному по ссылке аргументу:

```
function ChooseClass($class_type)
{
        switch ($class_type) {
            case "pop":
                $obj = new POPClass();
                break;
            case "smtp":
                $obj = new SMTPClass();
                break;
        }

 return $obj;
}

$ob = ChooseClass("pop");
```

31.2.3. Разыменование объектов

Вторая версия Zend позволяет разыменовывать объекты, возвращенные методами, например:

```
$object->method()->c_method().
```

Метод **method()** возвращает объект, который, в свою очередь, обладает методом **c_method()**. Новый механизм разыменования объектов позволяет «добраться» до метода **c_method()**. В предыдущей версии, которая использовала старую объектную модель, такой фокус был технически невозможен. В следующей версии Zend планируется добавить возможность разыменования объектов, возвращенных функциями: $func->method()->c_method().

Вот, как неуклюже выглядит код PHP 4, выполняющий те же действия, что и $object->method()->c_method():

```
$tmp =& $object->method();
$tmp =& $tmp->method();
$tmp->c_method();
```

31.2.4. Генная инженерия (клонирование объектов)

Нет, ничего общего с клонированием человека Zend 2.0 не имеет, но зато он умеет клонировать объекты.

До появления второй версии Zend программист не имел возможности определить, какой именно копирующий конструктор использовать при клонировании объекта. Zend 1.0 всегда делал побитовую копию всех полей объекта, что не очень хорошо. Новая версия Zend работает аналогично большинству объектно-ориентированных языков программирования: после удаления последней ссылки на объект, вызывается деструктор объекта перед его удалением из памяти. Напомню, что в PHP 4 (Zend 1.0) деструкторов вообще не было. Конечно, мы могли создать метод **Destructor()** и вручную его вызывать при окончании работы с объектом, но, согласитесь, это не то. Сейчас все изменилось: перед удалением объекта будет запущен его деструктор.

Однако из-за особенностей PHP эта новая возможность обладает подводными камнями:

- Из-за фатальной ошибки деструктор объекта может быть не запущен.
- Механизм подсчета ссылок может ошибиться, если объекты ссылаются друг на друга, в результате деструктор или вообще не будет запущен, или же будет запущен не вовремя, то есть будет еще одна (или даже несколько) ссылка на объект, а деструктор сработает.

Но в любом случае, наличие деструкторов — явление положительное. А использовать его нужно так:

```
class MyClass
{
    function __destruct()
    {
        … // Кодадеструктора
    }
}
```

Деструктор класса должен обязательно называться **__destruct**. Деструктор, как и конструктор, будет неявно вызываться движком перед удалением объекта. Также мы можем вызвать родительский деструктор:

```
parent::__destruct(); // в теле деструктора
```

31.2.5. Оператор delete

Мы знакомы с оператором **new**, создающим новый экземпляр класса. Во второй версии движка появился новый оператор **delete**, удаляющий экземпляр класса (объект). Оператор **delete** удаляет объект даже из памяти,

если на него ссылаются другие объекты. Попытка обращения к этим объектам после удаления главного объекта (на который они ссылались) приведет к фатальной ошибке, поэтому используйте оператор **delete** с осторожностью:

```
$ob = new My_Class();
delete $ob;
```

31.2.6. Конструкторы и иерархия классов

В Zend 1.0 конструктор — это метод, имя которого совпадает с именем класса. Из-за этого затруднено перемещение объектов по классовой иерархии. Сейчас поясню почему. Довольно часто приходится использовать вызов родительских конструкторов из порожденных классов. Если объект «переходит» к другому родителю, вызов родительского конструктора становится невозможным — мы же не можем предусмотреть список всех возможных имен «родителей».

Zend 2.0 предлагает называть конструктор __construct(), а не именем класса. Получается, что мы унифицировали все имя конструкторов и вызвать конструктор родительского класса становится довольно просто:

```
parent::__construct();
```

Предположим, что у нас есть родительский класс Poly (многоугольник) и дочерний Rectangle():

```
class Poly {
     function __construct()
     {
         // конструктор многоугольника
     }
     …
};

class Rectangle extends Poly {
     function __construct()
     {
          parent::__construct();
          // конструктор для Rectangle
          …
     }
     …
};
```

Так как у квадрата четыре угла, то он тоже является многоугольником:

```
class Square extends Poly {
     function __construct()
```

```
    {
        parent::__construct();
        // код инициализации прямоугольника
    }
    ...
};
```

Как мы знаем, квадрат — это тот же прямоугольник, только с равными сторонами.

Поэтому в иерархии объектов его нужно поместить между многоугольником и прямоугольником:

```
class Square extends Rectangle {
    function __construct()
    {
        parent::__construct();
        // код инициализации прямоугольника
    }
    ...
};
```

Как вы видите, мы не изменяли имя конструктора родительского класса, как это делалось в предыдущей версии Zend.

31.2.7. Дополнительные возможности

К дополнительным возможностям Zend 2.x относятся:

- Множественное копирование.
- Приватные и защищенные поля.
- Константы.
- Обработка исключений.
- Исправленная перегрузка ОО-синтаксиса.

В настоящее время продолжаются дискуссии относительно множественного копирования в Zend 2.x. В версии 2.0 множественное копирование на данный момент не поддерживается.

В любом развитом объектно-ориентированном языке программирования можно установить уровень доступа к полю объекта — приватный или публичный. К приватным полям имеют доступ только методы объекта, а к публичным разрешался доступ извне, то есть функциям и методам, не *принадлежащим данному классу. В Zend 1.0 все поля класса были публичными*. Вторая версия исправила это недоразумение. Данное нововведение позволяет решить проблему конфликтов имен полей разных родительских классов.

Приватные поля объявляются так:

```
class my_class {
     private $priv_field;

     function some_method(…)
     {
          $priv_field="private";
     }
…
};

$obj = new my_class;
$obj->priv_field = "public"; // произойдет фатальная ошибка
```

В отличие от приватных полей, защищенные поля (protected) доступны методам данного класса и методам дочерних классов, но не доступны извне:

```
class my_class {
     protected $prot_field;

…
};
```

Во второй версии Zend появилась возможность объявлять поля-константы, значение которых не может быть изменено:

```
class my_class {

const c = "Hello";

}

echo "my_class::\n";
```

Две последние возможности подробно рассматриваются в главе, посвященной объектно-ориентированному программированию на PHP.

SQLite

32.1. Введение в SQLite

32.1.1. Установка

Что же такое SQLite? Основное отличие SQLite от MySQL состоит в том, что SQLite не подключается к серверу баз данных, он сам является сервером. SQLite читает таблицы прямо с жесткого диска. Недостатком SQLite является то, что мы не можем хранить PHP-сценарии, использующие SQLite, на одном сервере, а саму базу данных — на другом. Может для вас это и не является критичным, поскольку обычно сервер MySQL устанавливается на том же компьютере, что и Web-сервер, но я считаю, что это не универсально.

Кроме недостатков SQLite имеет и преимущества — высокую производительность. Ведь он работает с таблицами напрямую, а не через сетевые сокеты, как MySQL — даже если MySQL установлен на одном компьютере, что и Web-сервер, обмен данными будет происходить через сетевую подсистему операционной системы.

Скачать SQLite можно по адресу:

- `http://pecl.php.net/package/SQLite`
- `http://sqlite.org/`

SQLite для Windows реализован в виде одной библиотеки — **php_sqlite.dll**, скачать которую вы можете по адресу `http://snaps.php.net/win32/PECL_STABLE/php_sqlite.dll`.

Установка SQLite проблем не вызывает, поэтому подробно останавливаться на ней мы не будем. Если вы устанавливаете SQLite для Linux, внимательно прочитайте файл INSTALL, а если для Windows, то скачайте файл `php_sqlite.dll` и «пропишите» его в файле конфигурации `php.ini`:

```
extension=sqlite.dll
```

32.1.2. Встроенные константы SQLite

Прежде чем перейти к рассмотрению функций для работы с SQLite, рассмотрим встроенные константы SQLite:

- SQLITE_ASSOC — результат будет возвращаться в виде ассоциативного массива;
- SQL_NUM — результат будет возвращен в виде списка, то есть ключи массива — числа;
- SQL_BOTH — двойное индексирование элементов массива: в зависимости от вашего желания вы можете работать с массивом, как если бы он был ассоциативным и как, если бы он был списком.

Данные константы аналогичны константам MYSQL_ASSOC, MYSQL_BOTH, MYSQL_NUM.

32.1.3. Список функций SQLite

Теперь перейдем к рассмотрению списка функций (я их упорядочил в алфавитном порядке):

- **sqlite_array_query()** — выполняет SQL-запрос и возвращает результат — массив, тип которого задан вышеупомянутыми константами;
- **sqlite_busy_timeout()** — устанавливает тайм-аут;
- **sqlite_changes()** — возвращает число записей, которые были изменены, вследствие последнего SQL-запроса;
- **sqlite_close()** — закрывает базу данных;
- **sqlite_column()** — получает одно поле из записи, то есть возвращает одну колонку результата;
- **sqlite_current()** — получает текущую запись из результата;
- **sqlite_error_string()** — возвращает текстовое описание ошибки;
- **sqlite_fetch_array()** — получает следующую запись из массива-результата;
- **sqlite_fetch_single()** — возвращает первую запись результата;
- **sqlite_fetch_string()** — это всего лишь псевдоним для **sqlite_fetch_single()**;
- **sqlite_field_name()** — возвращает имя поля;
- **sqlite_has_more()** — позволяет узнать, есть ли еще записи;
- **sqlite_last_error()** — возвращает код последней ошибки;
- **sqlite_last_insert_rowid()** — возвращает идентификатор последней добавленной записи;

- **sqlite_libversion()** — возвращает версию SQLite;
- **sqlite_next()** — переходит к следующей записи;
- **sqlite_num_fields()** — возвращает число полей (колонок) в результате;
- **sqlite_num_rows()** — возвращает число записей в результате;
- **sqlite_open()** — открывает базу данных;
- **sqlite_popen()** — открывает постоянное «соединение» с базой данных. Если база данных не существует, она будет создана;
- **sqlite_query()** — выполняет SQL-запрос и возвращает идентификатор результата;
- **sqlite_rewind()** — переходит к первой записи.

Я перечислил не все функции SQLite, однако перечисленных функций вам должно вполне хватить для работы с SQLite. Рассмотрим эти функции подробнее.

32.2. Открытие и закрытие базы данных

Для открытия базы данных используется функция **sqlite_open()**:

```
sqlite_open ( string $filename [, int $mode [, string
&error_message]])
```

Первый параметр — это имя файла таблицы, второй — режим доступа к таблице. По умолчанию используется режим 0666, позволяющий читать и записывать файл. Последний параметр — это строка, в которую будет записано сообщение об ошибке, если базу данных невозможно открыть.

Функции **sqlite_close()** нужно передать всего лишь один параметр — дескриптор базы данных.

Функция **sqlite_popen()** открывает постоянное «соединение» с базой данных:

```
sqlite_popen ( string $filename [, int $mode [, string
&error_message]])
```

Ее аргументы аналогичны функции **sqlite_open()**. Функция **sqlite_popen()** использует механизм постоянных ресурсов PHP. Вы открываете постоянное соединение, работаете, потом работа сценария завершается.

При следующем запуске сценария функция **popen()** ищет постоянное соединение, если оно существует, она будет его использовать, а не открывать базу данных заново. При частых запусках сценария это позволяет повысить его производительность.

32.3. Выполнение запросов

Функция **sqlite_query()** позволяет выполнить запрос к базе данных:

```
sqlite_query ( resource $dbhandle, string $query)
sqlite_query ( string $query, resource $dbhandle)
```

Обратите внимание, что функцию можно вызывать двояко: или сначала указывать дескриптор базы данных, а потом — SQL-запрос, или наоборот. Пример использования приведен в листинге 32.1.

Листинг 32.1. Открытие и закрытие базы данных

```
<?
// открываем базу данных
if ($db = sqlite_open('mysqlitedb', 0666, $sqliteerror)) {

// создаем таблицу tbl
     sqlite_query($db,'CREATE TABLE tbl (bar varchar(20))');
// Добавляем в таблицу новые записи
     sqlite_query($db,"INSERT INTO tbl VALUES ('val')");
// Получаем результат
     $result = sqlite_query($db,'select bar from tbl');
// Выводим результат
     var_dump(sqlite_fetch_array($result));
} else {
     die ($sqliteerror);
}
?>
```

Если запрос невозможно выполнить, функция sqlite_query() возвращает FALSE.

32.4. Функции для работы с результатом

Функция **sqlite_column()** используется для получения одной колонки результата:

```
sqlite_column ( resource $result, mixed $index_or_name [, bool
$decode_binary])
```

Первый параметр — это результат, второй — имя поля или индекс, а третий параметр указывает на необходимость двоичного кодирования.

Функция **sqlite_fetch_single()** возвращает строку — первую колонку результата:

```
sqlite_fetch_single ( resource $result [, int $result_type [,
bool $decode_binary]])
```

Первый параметр — это результат, а второй — тип результата. Данная функция похожа на функцию **sqlite_fetch_single()**, но возвращает только первую колонку, а не все сразу. Использование этой функции продемонстрированно в листинге 32.2.

Листинг 32.2. Функция sqlite_fetch_single

```
<?
if ($dbhandle = sqlite_open('mysqlitedb', 0666, $sqliteerror))
{

     $sql = "SELECT id FROM tbl WHERE id = 77";
     $res = sqlite_query($dbhandle, $sql);

     if (sqlite_num_rows($res) > 0) {
          echo sqlite_fetch_single($res); // 77
     }

     sqlite_close($dbhandle);
}
?>
```

Функция **sqlite_array_query()** выполняет SQL-запрос и возвращает массив-результат:

```
sqlite_array_query ( resource $dbhandle, string $query [, int
$result_type [, bool $decode_binary]])
```

Здесь:

- первый параметр — это дескриптор базы данных;
- второй — запрос;
- третий — тип результата (см. константы SQLite);
- а четвертый — это флаг кодирования данных.

Пример использования:

```
// Выполняем запрос
$q = sqlite_query($dbhandle, "SELECT * from tbl LIMIT 100");
// Результат будем писать в массив $rows
$rows = array();
// Записываем результат
while ($r = sqlite_fetch_array($q)) {
     $rows[] = $r;
```

Функция **sqlite_next()** перемещает указатель результата на следующую запись:

```
sqlite_next ( resource $result)
```

Параметр result — это наш результат. Если функция вернула FALSE, значит достигнута последняя запись.

Функция **sqlite_rewind** перемещает указатель результата на первую запись:

```
sqlite_rewind ( resource $result)
```

Функция **sqlite_seek()** переходит к записи с указанным номером:

```
sqlite_seek ( resource $result, int $rownum)
```

Если функция возвращает FALSE, значит записи с таким номером нет.

32.5. Информационные функции

Функция **sqlite_changes()** возвращает число записей, которые были изменены вследствие последнего запроса:

```
sqlite_changes ( resource $dbhandle)
```

Функция **sqlite_field_name()** возвращает имя поля по его номеру:

```
string sqlite_field_name ( resource $result, int $field_index)
```

Функция **sqlite_has_more()** возвращает TRUE, если результат еще содержит данные, иначе возвращается FALSE:

```
sqlite_has_more ( resource $result)
```

Функция **sqlite_last_error()** возвращает код последней ошибки (целое число):

```
sqlite_last_error ( resource $dbhandle)
```

а функция **sqlite_error_string()** — ее текстовое описание (строка текста):

```
sqlite_error_string( resource $dbhandle )
```

Функции **sqlite_num_rows()** и **sqlite_num_fields()** возвращают, соответственно, число записей и полей в результате:

```
sqlite_num_rows(resource $dbhandle)
sqlite_num_fields(resource $dbhandle)
```

32.6. Другие функции

Функция **sqlite_busy_timeout()** позволяет установить тайм-аут для базы данных:

```
sqlite_busy_timeout ( resource $dbhandle, int $milliseconds)
```

Напомню, что 1 секунда — это 1000 миллисекунд.

Вместо заключения: Горячая десятка ошибок

Ошибка 1

Наверное самой распространенной ошибкой начинающего PHP-программиста является «ошибка доллара» — денежная такая ошибка. Нет, это никак не связано с деньгами, просто вы забыли указать символ доллара перед именем переменной.

Неправильно:

```
$i = s + 10;
k = d*7;
for (i=1;i<10;i++) { ... }
```

Правильно:

```
$i = $s + 10;
$k = $d*7;
for ($i=1;$i<10;$i++) { ... }
```

Ничего страшного, привыкнете — поначалу сам делал такие ошибки, а потом привык до такой степени, что, программируя на С, пишу символ доллара перед именем переменной.

Ошибка 2

Вторая наиболее распространенная ошибка начинающих программистов заключается в следующем:

```
If ($k = 1) { ...}
```

Нужно писать так:

```
if ($k == 1) {...}
```

Выражение $k = 1 всегда будет истинным, поскольку вы присваиваете переменной $k значение 1.

Ошибка 3

Третья ошибка встречается при сравнении строк. Так как язык PHP сам автоматически приводит значение переменной к нужному типу, для сравнения строк (и логических переменных) желательно использовать оператор === вместо ==.

Неправильно:

```
if ($s1 == $s2) { ...}
```

Правильно:

```
if ($s1 === $s2) { ...}
```

Ошибка 4

Следующая ошибка часто допускается программистами, которые до этого программировали на PHP третьей версии. Заключается она в следующем:

```
$a = 0;
if (strval($a)=== "") echo "a - пустая строка"; // ничего не выведет
```

Ноль и "" не совсем тождественные понятия, поэтому в данной ситуации лучше использовать оператор ====:

```
if ($a === "" ) { ... }
```

Ошибка 5

Пятая ошибка уже была описана в этой книге. Остановимся на ней еще раз. Предположим, что нам нужно найти подстроку ab в строке abba:

```
if(strpos("abba","ab")!=false) echo "Строка ab есть в abba";
```

Очевидно, наш сценарий должен вывести строку "Строка ab есть в abba". Однако он ничего не выводит! Дело в том, что вхождение строки ab в строку abba начинается с нулевой позиции, а 0 и false при использовании оператора сравнения != — это синонимы. Правильнее будет написать так:

```
if(strpos("abba","ab")!==false) echo "Строка ab есть в abba";
или
if(strpos("abba","ab")===false) echo "Строки ab нет в abba";
```

Ошибка 6

Шестая ошибка — логическая. Ошибки такого рода часто встречаются в больших сценариях и очень плохо отлаживаются. Рассмотрим следующий код:

```
for ($i=0;$i<mysql_num_rows($r);$i++)
{
// после этого следуют строк 100-200 сценария
...

// а здесь программист забывает, что в качестве счетчика
// используется переменная
// $i в верхнем цикле и использует ее снова:

for ($i=0;$i<$messages;$i++) mail($f[email],"Test",$Msg);
...
}
```

Ошибка 7

«Ошибка неучтенной единицы». Заключается она в том, что нумерация индексов массива начинается с нуля, а вы думаете, что с единицы, или наоборот. И в первом, и во втором случае вы можете получить ошибку выхода за диапазон массива. Кроме того, в первом случае может случиться такое, что вы «недосчитаетесь» одного элемента массива.

Ошибка 8

Эта ошибка опять связана со строками. Предположим, что вы хотите вывести значение переменной:

```
echo 'Результат = $Res';
```

В итоге вы получаете вывод: Результат = $Res. Между строками в кавычках и в апострофах существует большая разница. Если строка заключена в апострофы, то все символы трактуются как есть. Исключения составляют последовательности \' и \\. Первая представляет собой обыкновенный апостроф, а вторая — обратную наклонную черту.

Строки в кавычках позволяют, кроме всего прочего, вывести значение переменной, поэтому никогда не используйте строку в апострофах для этой цели.

Ошибка 9

Предположим, что у нас есть некая функция и некоторая переменная, которая объявлена за пределами функции, то есть глобальная. В функции мы пытаемся получить ее значение, но у нас ничего не выходит. Чтобы глобальная переменная стала доступной для функции, нужно использовать инструкцию global:

```
global $G;
```

Ошибка 10

Наиболее часто встречающимися являются синтаксические ошибки (parse errors) — вы пропустили точку с запятой, забыли закрыть строку, и т.д. Чтобы свести количество таких ошибок до минимума, старайтесь писать программу, четко выделяя блоки, старайтесь не писать несколько операторов в одной строке. В общем, ваша программа должна быть читабельной — вы же ее пишете для себя. Читабельность программы позволяет не только быстро увидеть синтаксическую ошибку, но и «вспомнить» алгоритм программы через месяц, полгода, год.

Неправильный код с точки зрения читабельности:

```
$d = sin($s[1])*111; if ($i<77) { for($j=1;$j<10;$j++) { $s +=
$A[j]; echo "$s\n"; }}
```

Правильный код:

```
$d = sin($s[1])*111;
if ($i<77)
{
   for($j=1;$j<10;$j++)
      { $s += $A[j]; echo "$s\n";
      }
}
```

Предметный указатель

__construct 551
__destruct 550

A

Apache 71
Apache настройки: блок директив Directory 81
Apache настройки: блок директив Limit 82
Apache настройки: блок директив Location 84
Apache настройки: директива Action 79
— AddEncoding 79
— AddHandler 79
— AddIconByType 78
— AddLanguage 79
— AddType 79
— Alias 80
— Allow 82
— AllowOverride 81
— BindAddress 77
— CacheNegotiatedDocs 76
— DefaultType 79
— Deny 82
— DirectoryIndex 77
— DocumentRoot 75
— ErrorDocument 81
— ErrorLog 75
— FancyIndexing 78
— Group 74
— HostNameLookups 75
— KeepAlive 77
— KeepAliveTimeOut 77
— LanguagePriority 79
— Limit 81
— Listen 77
— MaxClients 77
— MaxRequestsPerChild 77
— MaxSpareServers 76
— MinSpareServers 76
— Options 81
— Order 82
— PidFile 75
— Port 74
— Redirect 80
— Require 82
— ScriptAlias 81
— ServerAdmin 74
— ServerName 73
— ServerRoot 74
— ServerType 74
— StartServers 76
— Timeout 76
— TransferLog 75
— User 74
— UserDir 75
— VirtualHost 85

B

break 154

C

chmod 263
continue 154

E

echo 116

G

global 214

I

include 158
include_once 160

M

mail() 329

O

OpenSSL 487
or die() 246

R

require 158
require_once 160
return 207

S

SQL: внешние ключи 310
SQL: ключи 309
SQL: оператор CREATE 303
— DELETE 306
— GROUP BY 308
— INSERT 304
— SELECT 306

— UPDATE 305
SQL: поля и их типы 300
SQL: сортировка записей 308
SQL: условие WHERE 306
SQL: функция AVG 308
— MAX 308
— MIN 308
— SUM 308
SSI 431
SSL 487

Z

Zend 1.0 477
Zend 2.0 477

А

Ассоциативный массив 187

Б

Библиотека GD 281
— libmcrypt 545
Блокирование доступа 267
Блокировка исключительная 268
— разделяемая 268

Г

Генератор случайных чисел 235

З

Заголовки HTTP 62
Запрет кэширования 430
Запуск внешних программ 238

И

Идентификатор SID 420
Инкапсуляция 469
Инструкция global 214
— include 158
— include_once 160
— require 158
— require_once 160
— return 207

К

Каналы 275
Класс 469
Класс HtmlMimeMail 332
Класс HtmlMimeMail: метод addattchment() 335
— addhtmlimage() 335
— send() 335
— setbcc() 336
— setcc() 336
— sethtml() 336
— settext() 336
Класс POP3 336
Класс POP3: метод deleteMessage() 338
— listMassages() 338
— login() 338
— open() 337
— resetDeleteMessages() 338
— retrieveMessage() 338
— statistics() 338
Конкатенация строк 144, 221
Константы SQLite: SQL_BOTH 555
— SQL_NUM 555
— SQLITE_ASSOC 555
Конструкция or die() 246
— switch-case 156

Л

Локаль 226

М

Массив 185
Массив ассоциативный 187
Математические функции 236
Модуль mod_php 114
— mod_ssl 487

Н

Наследование 470

О

Обработчик сессии session_handler_close() 425
— session_handler_destroy() 426
— session_handler_gc() 426
— session_handler_open() 425
— session_handler_read() 425
— session_handler_write() 425
Объект 469
Объектно-ориентированное программирование 468
Оператор @ 161
Оператор -> 471
Оператор break 154
— continue 154
— echo 116
— If-else 149
— присваивания 128
— эквивалентности 146
Операторы сравнения 142
Операции битовые 142
— логические 142
— строковые 144
Операций приоритет 142
Операция «обращение к символу в строке» 144
— «остаток от деления» 141
— вычитания 141
— декремента 141
— деления 141
— инкремента 141
— сложения 141
— умножения 141
Основные понятия БД: База данных 300
— Вторичный индекс 300
— Запись 300

— Запрос 300
— Индекс 300
— Первичный индекс 300
— Поле 300
— Таблица 300

П

Пакет php-mysql 116
Параметр error_reporting 161
— session.use_cookies 421
— session.use_trans_sid 421
Переменная HTTP_COOKIE 183
Переменные окружения: AUTH_TYPE 61
— CONTENT_LENGTH 61
— CONTENT_TYPE 61
— DOCUMENT_ROOT 61
— GATEWAY_INTERFACE 61
— HTTP_ACCEPT 62
— HTTP_ACCEPT_CHARSET 62
— HTTP_ACCEPT_ENCODING 62
— HTTP_ACCEPT_LANGUAGE 62
— HTTP_CONNECTION 62
— HTTP_HOST 62
— HTTP_REFERER 62
— HTTP_USER_AGENT 62
— HTTP_X_FORWARDED_FOR 62
— PATH_INFO 61
— PATH_TRANSLATED 61
— QUERY_STRING 61
— REMOTE_ADDR 61
— REMOTE_HOST 61
— REMOTE_PORT 61
— REMOTE_USER 61
— SCRIPT_FILENAME 61
— SCRIPT_NAME 61
— SERVER_ADDR 61
— SERVER_ADMIN 61
— SERVER_NAME 61
— SERVER_PORT 61
— SERVER_PROTOCOL 61
— SERVER_SIGNATURE 61
— SERVER_SOFTWARE 61
Переменная $REMOTE_ADDR 179
Полиморфизм 470
Полномочия пользователей MySQL 112
Права доступа 263
Приоритет операций 142

Р

Работа с классами: class 470
— delete 550
— extends 473
— new 471
— var 471
— оператор -> 471
Режимы блокировки файлов 268
Режимы открытия файлов 245
Результаты функции stat() 261
Рекурсия 216

С

Сертификат 490
Сессии 420
Система PHP-Nuke 454
Скрипт голосования Advanced Poll 361
Скрипт гостевой книги GuestsHere 385
Скрипт новостной ленты FusionPHP 371
Скрипт форума PHPBB 390
Сокет 320
Специальные символы 144
Список 186
Стандартные обработчики сессий 425

Т

Технология SSI 431
Типы файлов 261

У

Управление каналами 275

Ф

Файл httpd.conf 72
— httpd.conf пример конфигурации 89
— php.ini 479
Файл php.ini, директива: error_reporting 480
— magic_quotes_gpc 480
— max_execution_time 480
— track_vars 481
Финализатор 484
Флаги полей MySQL 317
Форма multipart 433
Функция acos() 237
— addslashes() 229
— array() 188
— array_merge() 191
— array_pop() 198
— array_push() 198
— array_reverse() 194
— array_shift() 199
— array_slice() 196
— array_splice() 197
— array_unique() 199
— array_unshift() 199
— arsort() 193
— asin() 237
— asort() 193
— atan() 237
— atan2() 237
— base_convert() 237
— basename() 260
— bindec() 238
— chdir() 272
— checkdate() 235
— chop() 228
— chr() 226
— closedir() 273
— compact() 200
— convert_cyr_string() 230
— copy() 256
— cos() 237
— count() 186
— crc32() 232
— create_function() 482
— crypt() 232
— current() 192

— date() 234
— date_sunrise() 546
— date_sunset() 546
— decbin() 238
— dechex() 238
— decoct() 238
— define() 135
— defined() 136
— die() 246
— dirname() 260
— disableCashe() 431
— end() 192
— error_reporting() 480
— escapeshellcmd() 241
— eval() 483
— exec() 240
— exp() 237
— extract() 200
— fclose() 254
— feof() 252
— fflush() 250
— fgetcsv() 257
— fgets() 254
— file() 255
— file_exists() 259
— fileatime() 262
— filectime() 263
— filemtime() 263
— filesize() 244
— filetype() 260
— flock() 268
— fopen() 245
— fputs() 254
— fread() 248
— fseek() 250
— fsockopen() 321
— ftell() 253
— func_get_arg() 213
— func_get_args() 213
— func_num_args() 213
— fwrite() 249
— get_headers() 546
— getcwd() 272
— gethostbyaddr() 325
— gethostbyname() 326
— gethostbynamel() 326
— getlastmod() 486
— gettype() 134
— header() 430
— hexdec() 238
— htmlspecialchars() 230
— idate() 546
— ImageArc() 288
— ImageColorAllocate() 286
— ImageColorAt() 287
— ImageColorClosect() 287
— ImageColorTotal() 287
— ImageColorTransparent() 287
— ImageCopy() 296
— ImageCopyResized() 295
— ImageCreate() 284
— ImageCreateFromGif() 283
— ImageCreateFromJpeg() 283
— ImageCreateFromPng() 283
— ImageDashedLine() 288
— ImageDestroy() 283
— ImageFill() 289
— ImageFilledPolygon() 288
— ImageFilledRectangle() 288
— ImageFillToBorder() 289
— ImageGif() 283
— ImageJpeg() 283
— ImageLine() 288
— ImageLoadFont() 291
— ImagePng() 283
— ImagePolygon() 288
— ImageRectangle() 288
— ImageSetPixel() 287
— ImageString() 290
— ImageStringUp() 292
— ImageSX() 285
— ImageSY() 285
— ImageTTFbox() 295
— ImageTTFtext() 292
— is_array() 133
— is_bool() 134
— is_dir() 274
— is_double() 133
— is_executable() 267
— is_file() 274
— is_float() 133
— is_int() 133
— is_integer() 133
— is_object() 134
— is_redable() 267
— is_string() 133
— is_writeable() 267
— isset() 139
— join() 255
— key() 192
— krsort() 194
— ksort() 194
— list() 188
— log() 237
— lstat() 262
— ltrim() 228
— max() 236
— md5() 231
— min() 236
— mkdir() 272
— mt_rand() 235
— mt_srand() 235
— mysql_close() 313
— mysql_connect() 312
— mysql_errno() 314
— mysql_error() 314
— mysql_fetch_array() 316
— mysql_fetch_assoc() 316
— mysql_fetch_row() 316
— mysql_field_flags() 317
— mysql_field_name() 317
— mysql_field_type() 317
— mysql_list_tables() 319
— mysql_num_fields() 315

— mysql_pconnect() 313
— mysql_query() 314
— mysql_result() 315
— mysql_select_db() 313
— next() 192
— nl2br() 224
— octdec() 238
— opendir() 273
— ord() 226
— passthru() 240
— phpinfo() 485
— phpinfo() 486
— phpversion() 486
— pi() 237
— popen() 278
— pow() 237
— prev() 192
— range() 201
— rawurlDecode() 229
— rawurlEncode() 229
— readdir() 273
— realpath() 260
— register_shutdown_function() 484
— rename() 258
— reset() 192
— rmdir() 272
— serialize() 232
— session_id() 423
— session_is_registered() 423
— session_name() 423
— session_register() 422
— session_save_path() 424
— session_set_save_handler() 426
— session_start() 422
— session_unregister 423
— session_unset() 423
— set_file_buffer() 251
— setcookie() 182
— setlocale() 227
— settype() 134
— shuffle() 195
— sin() 237
— sleep() 486
— socket_set_blocking() 321
— sort() 194
— sqlite_array_query() 558
— sqlite_changes() 559
— sqlite_close() 556
— sqlite_column() 557
— sqlite_current() 555
— sqlite_error_string() 559
— sqlite_fetch_array() 555
— sqlite_fetch_single() 557
— sqlite_field_name() 559
— sqlite_has_more() 559
— sqlite_last_error() 559
— sqlite_last_insert_rowid() 555
— sqlite_libvertion() 556
— sqlite_next() 558
— sqlite_num_fields() 559
— sqlite_num_rows() 559
— sqlite_open() 556
— sqlite_popen() 556
— sqlite_query() 557
— sqlite_rewind() 559
— sqlite_seek() 559
— sqrt() 237
— stat() 260
— str_replace() 224
— str_split() 546
— strcasecmp() 224
— strcmp() 224
— stream_copy_to_stream() 545
— stream_get_line() 545
— stream_set_blocking() 321
— stream_socket_accept() 545
— stream_socket_client() 545
— stream_socket_get_name() 545
— stream_socket_server() 545
— strip_tags() 225
— stripslashes() 229
— strlen() 223
— strpbrk() 546
— strpos() 223
— strrpos() 226
— strtolower() 226
— strtoupper() 226
— substr() 224
— system() 240
— tan() 237
— tempname() 260
— time() 235
— tmpfile() 248
— touch() 263
— trim() 227
— uasort() 196
— uksort() 195
— unlink() 259
— unserialize() 233
— unset() 140
— urldecode() 228
— urlencode() 228
— usleep() 486
— usort() 196
— virtual() 432
— wordwrap() 225

Ц

Цикл do while 151
— for 152
— foreach 155
— while 150

Ч

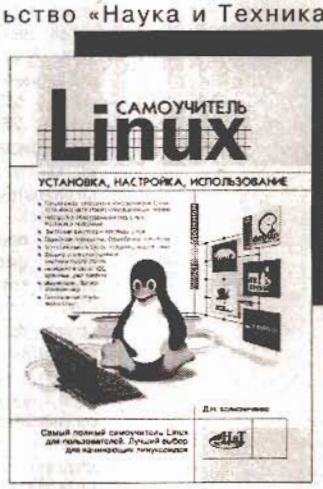
Чат CHAT.PHP.SPB.RU 352
Чат SimpleChat 346

Издательство «Наука и Техника»

Д.Н. Колисниченко

Самоучитель LINUX

Установка, настройка, использование



высылает книги почтой

ISBN: 5-94387-137-3
Формат: 70x100/16
Объем: 640 с.: ил.

Издание 2-е, переработанное и дополненное

Данная книга является самым полным руководством пользователя Linux. Предназначена для всех, кто хочет эффективно использовать у себя на компьютере эту систему. Изложение материала происходит последовательно, начиная с этапа установки и заканчивая специальными технологиями Linux.

Подробно описаны внутреннее устройство Linux, настройка оборудования и оптимизация системы. Обсуждаются возможные проблемы. Особое внимание уделено защите и восстановлению Linux после сбоев.

В книге детально рассмотрен весь круг задач, который может стоять перед пользователем: от редактирования текста, работы в Интернете и запуска Windows-игр, до компилирования ядра, администрирования системы и записи компакт-дисков. Учитываются особенности разных дистрибутивов.

Подробно описаны мультимедиа-инструменты Linux: фотомонтаж, просмотр и редактирование видео, настройка TV-тюнера, обработка звука и многое другое. Рассмотрено множество программ и приложений под Linux: OpenOffice, KOffice, The Gimp, MainActor, WineX, VMWare, Lan Billing, и т.д.

Автор книги — опытный и известный консультант по использованию Linux. Книга написана простым и понятным языком. Лучший выбор для начинающих линуксоидов.

серия — просто о сложном — серия

Издательство «Наука и Техника»

Н.А. Кузнецова

Установка и переустановка **Windows**

высылает книги почтой

ISBN: 5-94387-088-1
Формат: 60x88/16
Объем: 128 с.: ил.

Издание 2-е, переработанное и дополненное

Эта книга позволяет любому, даже самому начинающему пользователю компьютера, переустановить, установить и настроить Windows. Для этого в книге приведены пошаговые описания всех необходимых действий. Причем каждый шаг сопровождается иллюстрацией (копией экрана) и ясными указаниями, что нужно делать. Отдельно рассмотрена переустановка, установка и настройка Windows 98, Windows XP и 2000.

Кроме этого, в книге описано, как можно автоматизировать установку Windows — сделать ее автоматической. Отдельно показано, как установить несколько операционных систем на одном компьютере, а также рассмотрена технология восстановления Windows XP без переустановки.

Приведено описание специальных режимов установки Windows. Разобраны возможные проблемы и показаны пути их решения.

Книга написана простым и доступным языком. Будет очень полезна начинающим пользователям ПК, но и опытные пользователи найдут для себя много интересного.

серия — просто о сложном — серия

Издательство «Наука и Техника»

И.Н. Подольский

Печать на ПК

слепым десятипальцевым методом

высылает книги почтой

ISBN: 5-94387-111-X
Формат: 60x88/16
Объем: 80 с.: ил.

Издание 2-е, переработанное и дополненное

Это новое издание популярной книги предназначено для самостоятельного изучения приемов работы на клавиатуре ПК «слепым» десятипальцевым методом на русском и английском языках.

Приведенные упражнения позволят вам достичь мастерства и высокой скорости печати в самые сжатые сроки. Описываемая методика испытана годами и отработана автором при обучении многих специалистов.

Также в книге даны основные правила по оформлению документов, правки текста, полезные советы по организации правильной работы на ПК.

Имеющаяся схема клавиатуры в натуральную величину позволит вам заниматься самообучением и без использования компьютера. Поэтому, если вы только планируете приобретать ПК или работать на нем, то эта книга также для вас.

Книга предназначена для широкого круга пользователей ПК.

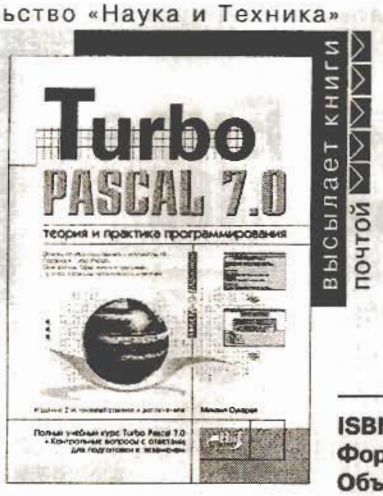
серия — просто о сложном — серия

Издательство «Наука и Техника»

М. Сухарев

Turbo Pascal 7.0

Теория и практика программирования



ISBN: 5-94387-126-8
Формат: 70x100/16
Объем: 640 с.: ил.

Издание 2-е, переработанное и дополненное

Книга представляет собой подробное учебное пособие по языку Turbo Pascal 7.0 и общим принципам программирования. Изложение материала построено от простого к сложному. Используется большое количество примеров: от работы с текстом и рисования простых графических объектов, до спрайтовой анимации, обработки драйвера мыши и построения СУБД.

Подробно разбирается код каждого примера и обсуждаются возможные ошибки. Отдельно рассмотрены специальные приемы программирования, как, например, рекурсия.

Кроме того, в книге приведены все необходимые сведения, позволяющие принципиально понять, как создается и работает программа, осуществляется адресация и взаимодействие с памятью, обрабатывается информация, функционирует компьютер. То есть все то, без чего невозможно полноценно изучить программирование. В последней главе описано, как правильно и профессионально оформлять программы, изложены положения Венгерской нотации. Завершает книгу справочный материал по языку Turbo Pascal 7.0.

Книга предназначена для школьников и студентов. Написана автором на основе многолетнего опыта преподавания программирования в Санкт-Петербургском Городском Дворце творчества юных. Особенно будет полезна для всех, кто хочет получить серьезные и глубокие знания. Адаптирована для самостоятельного изучения.

серия — просто о сложном — серия

Издательство «Наука и Техника»

В.В. Серогодский

Excel 2002

Популярный самоучитель

высылает книги почтой

ISBN: 5-94387-058-X
Формат: 70x100/16
Объем: 352 с.: ил.

Данная книга является превосходным самоучителем, позволяющим самостоятельно освоить Excel 2002. Благодаря ей вы научитесь создавать, редактировать и оформлять таблицы, готовить их к печати, выполнять вычисления, строить графики и диаграммы, а также многому другому.

В конце каждого раздела имеется подробный пошаговый пример. Каждое действие в нем сопровождается рисунком — копией окна Excel. Всего в книге около 70 таких примеров. Выполняя их под руководством автора, вы на практике будете закреплять пройденный материал. Таким образом, прочитав данную книгу, вы получите устойчивые навыки работы в Excel.

Отдельное внимание уделено использованию Excel 2002 для решения конкретных практических задач: составлению и оформлению отчетов, прайс-листов, простых баз данных, анализу результатов вычислений. Много полезного для себя найдут студенты, работники фирм (менеджеры, секретари, бухгалтера и т.п.), инженеры.

Книга отличается простотой и ясностью изложения материала. Написана автором на основе многолетнего опыта преподавания на компьютерных курсах. Лучший выбор для начинающих.

серия — просто о сложном — серия

Издательство «Наука и Техника»

Ю.А. Резник

Графика, звук, видео на ПК

Популярный самоучитель

высылает книги почтой

ISBN: 5-94387-073-3
Формат: 70x100/16
Объем: 336 с.: ил.

Данная книга посвящена компьютерному творчеству. Она научит вас делать на компьютере множество полезных и интересных вещей. Охвачен самый широкий круг популярных проектов: поздравительная открытка и карикатура, фотомонтаж и видеомонтаж, анимационный клип и видеоэффекты, трехмерные пейзажи и обработка звука, изготовление визитки, брошюры, Web-страницы и многое другое. При этом изначально от вас не требуется никаких специальных знаний и навыков.

В отличие от других эта книга не заставляет вас от начала до конца изучать разные программы, а потом самим разбираться, как их использовать на практике для решения конкретных задач. Здесь подробно рассмотрены реальные примеры и описано только то, что действительно необходимо.

В каждом проекте используется новый прием, набор инструментов или технология. Материал изложен последовательно, от простого к сложному. Благодаря этому, прочитав книгу, вы достаточно хорошо освоите различные программные продукты (Photoshop, CorelDRAW, Premiere, 3D MAX, SoundForge и др.). Причем сделаете это на практике.

Книга предназначена для широкого круга читателей. Написана простым и понятным языком. Лучший выбор для начинающих.

серия — просто о сложном — серия

Книги и Наборы схем ПОЧТОЙ

Издательство «Наука и Техника» принимает заказы на продажу печатной продукции по почте наложенным платежом.

Оплата производится на почте при получении книг, для организаций возможна оплата по безналичному расчету после выставления счета.
Для этого Вам необходимо оформить бланк заказа и отправить его нам.

Для жителей России:
193029 Санкт-Петербург, а/я 44,
ООО «Наука и Техника»
тел/факс (812)-567-70-26, 567-70-25
E-mail: nit@mail.wplus.net

Для жителей Украины:
02166 Киев-166, ул. Курчатова, 9/21,
«Наука и Техника»
тел/факс (044)-516-38-66, 559-27-40
E-mail: nits@voliacable.com

Перечень рассылаемых книг размещен на сайте издательства: www.nit.com.ru

Вы также можете приобрести книги других издательств по тематикам:

Радиоэлектроника ..... ☐
Компьютерные технологии ..... ☐
Техника безопасности на предприятиях ..... ☐
Транспорт ..... ☐

Примечание. Заказанные бесплатные каталоги высылаются по электронной почте или вкладываются в посылку с книгами ***(отдельно письмом каталоги не рассылаются).***

Заказ на каталоги

Заполняйте поля аккуратно большими отдельными буквами.

Информация для приобретения книг почтой частными лицами

1. Фамилия, имя, отчество _____
2. Почтовый адрес: индекс _____ страна _____
область _____ город, поселок _____
улица _____ дом _____ корпус _____ кв. _____
телефон (_____) _____
адрес электронной почты (если он у Вас есть) : E-mail: _____

Информация для выставления счета организациям

Название _____ ИНН _____
Телефон/факс (_____) _____
Контактное лицо _____
Адрес для отправки заказа: индекс _____ страна _____
область _____ город, поселок _____
улица _____ дом _____
адрес электронной почты (если он у Вас есть) : E-mail: _____

БЛАНК ЗАКАЗА

(принимаются ксерокопии)

Автор	Название	Цена Россия (руб.)	Цена Украина (грн.)	Год	Объем	Заказ (экз.)
Профи						
Вебер	Knowledge-технологии в консалтинге и управлении предприятием + CD	127,00	25,00	2003	С_176	
Гургенидзе	Мультисервисные сети и услуги широкопол. доступа	143,00	29,00	2003	С_400	
Кучеров	Источники питания ПК и периферии, изд. 2-е	108,00	22,00	2002	С_384	
Кучерявый	Пакетная сеть связи общего пользования	132,00	27,00	2004	С_272	
Назаров	Нейросетевые алгоритмы прогноз. и оптимизации	160,00	32,00	2003	С_384	
Никамин	Цифровая звукозапись. Технологии и стандарты	76,00	15,00	2002	М_256	
Щеглов	Защита компьютерной информации от несанкционированного доступа	154,00	31,00	2004	С_384	
Секреты мастерства						
Колисниченко	Linux-сервер своими руками, изд. 2-е, перераб. и доп.	193,00	39,00	2004	С_704	
Финков	Интернет. Шаг второй: от польз. к проф. + CD	88,00	18,00	2002	С_768	
Просто о сложном						
Колисниченко	Самоучитель Linux. Установка, настройка, использование	152,00	30,00	2004	С_640	
Колисниченко	Самоучитель PHP 5	165,00	33,00	2004	С_576	
Кузнецова	Установка и переустановка Windows, изд. 2-е	39,00	6,00	2004	М_128	
Подольский	Печать на ПК слепым десятипальцевым методом, изд. 2-е + макет клавиатуры	39,00	8,00	2003	М_80	
Пестриков	Turbo Pascal 7.0. Изучаем на примерах	109,00	21,00	2004	С_368	
Резник	Графика, звук, видео на ПК. Популярный самоучитель	110,00	22,00	2003	С_336	
Серогодский	Excel 2002. Популярный самоучитель	110,00	22,00	2003	С_352	
Сухарев	Turbo Pascal 7.0. Теор. и практ. программир., изд. 2-е	165,00	33,00	2004	С_640	
Конспект программиста						
Будилов	JavaScript, XML и объектная модель документа	28,00	5,00	2001	С_352	
Будилов	Практические занятия по HTML. Краткий курс + дск	55,00	10,00	2001	С_256	
Будилов	Практические занятия по PHP4 + CD	44,00	9,00	2001	С_352	
Картузов	Программирование на языке Java: краткий курс	18,00	4,00	2001	С_192	
Костельцев	Построение интерпретаторов и компиляторов + CD	44,00	9,00	2001	М_224	
Куправа	Самоучитель Access 97/2000 + дск	63,00	10,00	2001	С_144	
Николенко	MIDI – язык богов + CD	39,00	8,00	2000	С_144	
Николенко	Практические занятия по JavaScript + дск	18,00	4,00	2000	С_128	
Николенко	Самоучитель по Visual C++6	105,00	15,00	2001	С_368	
Цеховой	Web: дизайн и коммерция	22,00	4,00	2001	С_190	
Библиотека пользователя						
Будилов	Работаем с Finale 2001 + CD	50,00	5,00	2001	С_240	
Серогодский	Просто о сложном. Windows 98	43,00	9,00	2001	С_144	
Серогодский	Просто о сложном. Word 2000	57,00	9,00	2001	М_224	
Цеховой	Энциклопедия дизайнера CorelDRAW10 + CD	44,00	9,00	2001	С_512	
Популярный учебник						
Есипов	Информатика (учебник), изд. 3	102,00	20,00	2003	М_400	
Есипов	Информатика (задачник)	102,00	20,00	2001	М_368	
Радиомастер						
Авраменко	Схемотехника CD-проигрывателей + схемы	102,00	20,00	2003	С_192	
Виноградов	Источники питания видеомагнитофонов	68,00	14,00	2001	Б_256	
Виноградов	Сервисные режимы телевизоров-1	55,00	10,00	2002	С_208	
Виноградов	Сервисные режимы телевизоров-2	55,00	10,00	2001	С_208	
Виноградов	Сервисные режимы телевизоров-3	55,00	10,00	2001	С_208	
Виноградов	Сервисные режимы телевизоров-4	55,00	10,00	2002	С_208	
Корякин-Черняк	Сервисные режимы телевизоров-5	55,00	10,00	2002	С_208	
Корякин-Черняк	Сервисные режимы телевизоров-6	55,00	10,00	2002	С_208	
Корякин-Черняк	Сервисные режимы телевизоров-7	55,00	10,00	2002	С_208	
Виноградов	Сервисные режимы телевизоров-8	55,00	10,00	2002	С_208	
Корякин-Черняк	Сервисные режимы телевизоров-9	55,00	10,00	2002	С_208	
Виноградов	Сервисные режимы телевизоров-10	55,00	10,00	2002	С_208	
Корякин-Черняк	Сервисные режимы телевизоров-11	55,00	10,00	2002	С_208	
Корякин-Черняк	Сервисные режимы телевизоров-12	55,00	10,00	2002	С_208	
Журавлев	Телевизионные процессоры системы управления	83,00	15,00	2001	С_512	
Заикин	Современные радиотелефоны + схемы	175.00	36,00	2003	С_352	
Корякин-Черняк	Абонентские телефонные аппараты	121,00	24,00	2003	С_368	

(принимаются ксерокопии)

Автор	Название	Цена Россия (руб.)	Цена Украина (грн.)	Год	Объем	Заказ (экз.)
Корякин-Черняк	АОН в телефонных аппаратах	127,00	25,00	2003	С_288	
Корякин-Черняк	Автосигнализации от А до Z, изд. 2-е, перераб. и доп.	127,00	25,00	2003	С_336	
Котенко	Электронные телефонные аппараты, изд. 3-е, перераб.	106,00	21,00	2003	С_272	
Кучеров	Источники питания мониторов	66,00	10,00	2001	С_240	
Мухин	Энциклопедия мобильной связи. Том 1	63,00	10,00	2001	С_240	
Партала	Видеокамеры + (вклейки: 13 схем)	56,00	10,00	2000	С_192	
Партала	Цифровая электроника, изд. 2-е, перераб. и доп.	70,00	10,00	2001	С_224	
Пашкевич	Модернизация телевизоров 3..5УСЦТ, изд. 2-е	99,00	20,00	2001	С_320	
Пестриков	Домашний электрик и не только... Книга 1	76,00	15,00	2003	С_208	
Пестриков	Домашний электрик и не только... Книга 2	76,00	15,00	2003	С_208	
Пестриков	Радиоэлектроника в конструкциях и увлечениях	87.00	18,00	2004	С_240	
Пестриков	Энцикл. радиолюбителя. Работаем с компьютером	87.00	18,00	2004	С_272	
Янковский	Видеомагнитофоны серии ВМ + (вклейки: 18 схем)	66,00	13,00	2001	Б_272	
Телемастер и энциклопедия телемастера						
Безверхний	Телевизоры DAEWOO и SAMSUNG + схемы	149,00	30,00	2003	С_144	
Виноградов	Импульсные источники питания видеомагнитофонов	77,00	15,00	2003	С_160	
Корякин-Черняк	Применение телевизионных микросхем. Том 1	142,00	29,00	2004	С_320	
Корякин-Черняк	Применение телевизионных микросхем. Том 2	142,00	29,00	2004	С_304	
Пьянов	Телевизоры LG на шасии MC-51B/74A/991A + схемы	83,00	17,00	2003	С_160	
Пьянов	Видеопроцессоры семейства UOC + схемы	132,00	26,00	2003	С_144	
Саулов	Нов. телевизоры Horizont + (вклейки: 10 схем)	150,00	30,00	2002	С_480	
Саулов	Переносные телевизоры	154,00	30,00	2002	С_496	
Янковский	Импульсные источники питания телевизоров	127,00	25,00	2003	С_384	
Электронные компоненты						
Авраменко	Видеопроцессоры. Справочник	87,00	18,00	2004	С_256	
Герасимов	Интегральные УНЧ, изд. 2-е, перераб. и доп.	154,00	31,00	2003	М_528	
Никамин	Микросхемы для CD-проигрывателей. Сервосистемы	176,00	35,00	2003	С_272	
Партала	Цифровые КМОП-микросхемы	83,00	15,00	2001	С_400	
Ульрих	Микроконтроллеры PIC16X7XX	99,00	20,00	2002	М_320	
	Зарубежные микросхемы, тразисторы, диоды. A...Z	202,00	40,00	2003	М_768	
	Заруб. м/схемы, тразисторы, диоды. 0...9, изд. 2-е	164,00	33,00	2004	М_590	
Зарубежная электроника						
Авраменко	Ремонт и регулировка CD-проигр. + (вклейки: 26 схем)	58,00	12,00	1999	Б_160	
Брускин	Зарубежные резидентные радиотелефоны, изд. доп.	48,00	10,00	2000	Б_176	
Брускин	Схемотехника автоответчиков + (вклейки: 14 схем)	18,00	4,00	1999	Б_176	
Кульский	КВ-приемник мирового уровня (для радиолюбителей)	40,00	8,00	2000	С_352	
Куприянов	Тех. обеспечение цифровой обработки сигналов	66,00	13,00	2000	С_752	
Лаврус	Практика измерений в телевизионной технике	18,00	4,00	2000	Б_192	
Лукин	Телевизоры ближнего зарубежья	18,00	4,00	1998	Б_136	
Домашний мастер						
Балясников	Обувь. Выбор, уход, ремонт	72,00	14,00	2003	М_240	
Бирюков	Защита автомобиля от угона	61,00	12,00	2003	М_176	
Богданов	Умный дом	44,00	9,00	2003	М_112	
Корякин-Черняк	Справочник домашнего электрика	121,00	24,00	2003	С_432	
Корякин-Черняк	Холодильники от А до Я, изд. 2-е, перераб. и доп.	123,00	25,00	2003	С_416	
Корякин-Черняк	Стиральные машины от А до Я	105,00	21,00	2002	С_304	
Ландик	Современные холодильники NORD	72,00	14,00	2003	С_144	
Пестриков	Азбука сотового телефона, изд. 2-е, перераб. и доп.	112,00	23,00	2004	С_352	
Радиолюбитель						
Белов	Самоучитель по микропроцессорной технике	77,00	15,00	2003	М_223	
Виноградов	Микропроцессорное управление телевизорами	55,00	11,00	2003	М_144	
Петров	Звуковая схемотехника для радиолюбителей	109,00	22,00	2004	М_400	
Семьян	500 схем для радиолюбителей. Приемники	60,00	12,00	2004	М_192	
Справочник рабочего						
Шевченко	Чтение машиностроительных чертежей. Справ. пособие	83,00	15,00	2003	М_192	

Примечания.

1. В колонке «Объем» буквой указан размер книги в см: М — 14x21, С — 16x23, Б — 20x28.

*2. +**CD** — означает, что к книге прилагается компакт-диск. +**дск** — означает, что к книге прилагается дискета.*

***3. Цены розничные, указаны без учета почтовых расходов.** Цены приведены по состоянию на май 2004 года.*

Ваш лучший выбор

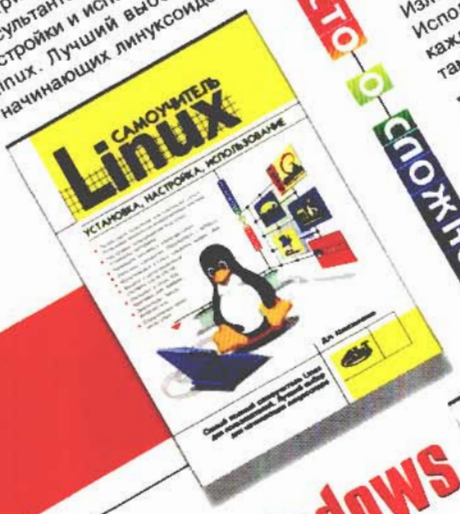
книги серии ПРОСТО О СЛОЖНОМ

Самоучитель Linux

«Самоучитель Linux. Установка, настройка, использование» позволяет пользователю в полном объеме освоить Linux и ее приложения. Книга написана известным и опытным консультантом по вопросам настройки и использования Linux. Лучший выбор для начинающих линуксоидов.

Turbo Pascal 7.0

Книга «Turbo Pascal 7.0. Теория и практика программирования» представляет собой подробный учебник по языку Turbo Pascal 7.0 и общим принципам программирования. Изложение материала построено от простого к сложному. Используется большое количество примеров. В конце каждой главы имеются контрольные вопросы с ответами для подготовки к экзаменам. Завершает книгу справочный материал по языку Turbo Pascal 7.0.



Установка и переустановка Windows

С помощью книги «Установка и переустановка Windows» каждый сможет установить, переустановить и настроить Windows. Для этого в книге приведены пошаговые описания всех необходимых действий. Разобраны возможные проблемы и показаны пути их решения. Помимо общей методики в книге описано как можно автоматизировать установку Windows. Также рассмотрена установка нескольких операционных систем на одном компьютере.

Россия: Санкт-Петербург,
пр. Обуховской обороны, 107,
(812)-567-70-25, 567-70-26
E-mail: nit@mail.wplus.net

Украина: Киев, ул. Курчатова, 9/21,
(044)-516-38-66, (044)-559-27-40
E-mail: nits@voliacable.com

Официальный сайт:
www.nit.com.ru