Максим Кузнецов
Игорь Симдянов

PHP

НА ПРИМЕРАХ

2-е издание

Санкт-Петербург
«БХВ-Петербург»
2012

УДК 681.3.068+800.92PHP
ББК 32.973.26-018.1
К89

Кузнецов, М. В.

К89 PHP на примерах / М. В. Кузнецов, И. В. Симдянов. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2012. — 400 с.: ил.

ISBN 978-5-9775-0445-4

Рассмотрены приемы программирования на PHP, позволяющие разрабатывать современные эффективные Web-приложения. Использованы многочисленные примеры, взятые из реальной практики. Первое издание книги под названием «PHP 5 на примерах» охватывало только пятую версию языка. Второе издание полностью обновлено и переработано. Рассмотрено взаимодействие PHP-приложений с Web-сервером Apache и СУБД MySQL, извлечение информации с удаленных серверов, взаимодействие с браузером посредством технологии AJAX, использование библиотеки jQuery. Показаны тонкости работы с HTTP-протоколом, ньюансы ReWrite-преобразований, особенности применения интеллектуальных агентов на PHP. Приведены примеры защиты Web-приложений, работы с графикой, Flash и PDF-документами, оптимизации кода и решения ряда других важных задач.

Для программистов и Web-разработчиков

УДК 681.3.068+800.92PHP
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Елена Кашлакова*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Подписано в печать 30.09.11.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 32,25.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

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

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

ISBN 978-5-9775-0445-4

Оглавление

Введение

Книги, посвященные языкам программирования, как правило, условно можно поделить на две большие группы: одни рассчитаны на читателя, не знакомого с языком программирования, другие излагают профессиональные особенности, ориентированные на программистов с базовыми основами. Данная книга относится к последнему типу (если какие-то конструкции вам покажутся слишком сложными или незнакомыми, вы всегда можете обратиться к нашему самоучителю по PHP 5/6, излагающему язык последовательно).

С одной стороны, это означает, что в книге вы не встретите последовательного изложения языка от самого простого к самому сложному. Зато независимо от того, начинаете вы постигать азы Web-программирования или уже освоили язык и ищете свежих идей, вы сможете начать использовать примеры из книги практически сразу. Подавляющее большинство примеров из книги взято из реальной практики создания коммерческих Web-приложений и может без адаптации быть использовано на вашем сайте. По мере чтения книги мы познакомимся с PHP, начиная с самых низкоуровневых операций, таких как работа с файлами, директориями, строками, и заканчивая использованием PHP в AJAX-приложениях.

Язык программирования не существует сам по себе, он является частью технологии создания Web-приложений. Современному разработчику необходимо разбираться в конфигурировании Web-сервера Apache и его модулей, управлении и проектировании базами данных, представлять особенности работы сети Интернет и сетевых протоколов, владеть регулярными выражениями, языком разметки HTML, каскадными таблицами стилей CSS и клиентским языком JavaScript. Создавая эту книгу, мы постарались на коротких примерах показать связь PHP со всеми этими технологиями.

Для того чтобы изучить конкретный язык программирования, его синтаксис и шаблоны, обычно уходит несколько месяцев. Для того чтобы овладеть технологией, позволяющей создавать современные приложения в той или иной области, могут потребоваться годы. Данная книга предназначена для того, чтобы сократить это время и показать PHP в окружении других языков программирования и технологий.

Где искать помощи

Дополнительные материалы можно также найти на группе сайтов IT-студии SoftTime, руководителями которой являются авторы книги:

- **http://www.softtime.ru** — главный сайт;
- **http://www.softtime.org** — проекты студии;
- **http://www.softtime.info** — консультации авторов книг;
- **http://www.softtime.biz** — услуги студии.

Если по мере чтения у читателей возникнут технические вопросы, обсудить их можно на PHP-форуме авторов **http://www.softtime.ru/forum/**. Вопросы правового характера обсуждаются на форуме **http://www.softtime.org/forum/**.

Осветить все функции, расширения PHP, а также смежные дисциплины, необходимые для создания Web-приложений, в одной книге невозможно — объем их чрезвычайно велик. Поэтому мы регулярно выпускаем книги по Web-программированию, затрагивающие самые разнообразные вопросы, начиная с проблем безопасности и заканчивая психологическими аспектами профессии программиста. С полным списком наших книг можно ознакомиться по ссылке **http://www.softtime.ru/php5/index.php**.

Благодарности

Авторы выражают признательность сотрудникам издательства "БХВ-Петербург", благодаря которым наша рукопись увидела свет.

Авторы также благодарны всем читателям, приславшим отзывы на наши предыдущие книги, а также всем участникам форума SoftTime.

ГЛАВА 1

Установка Web-сервера Apache, интерпретатора PHP и СУБД MySQL

Для того чтобы создать удобную среду программирования, недостаточно установить инсталляционные пакеты, потребуется их тщательное конфигурирование. Данная глава посвящена устройству рабочего места Web-разработчика.

1.1. Что нужно, чтобы запустить PHP-скрипт

Программирование на PHP отличается от программирования на других языках достаточно сложной системой настройки среды. Первым шагом в освоении Web-технологий является воспроизведение рабочей среды на компьютере разработчика. Для этого потребуется запустить и настроить несколько программных компонентов.

Доступность Web-приложений и HTML-файлов в сети Интернет обеспечивается Web-сервером, который по протоколу HTTP выдает их любому клиенту, правильно оформившему запрос. Наиболее популярным сервером, используемым совместно с PHP, долгие годы остается Web-сервер Apache.

Интерпретатор PHP представляет собой либо внешнюю CGI-программу, либо динамическую библиотеку, которую необходимо подключить к Web-серверу, чтобы вместо кода PHP-скриптов клиенту выдавались результаты его выполнения. Ситуация осложняется тем, что к PHP-интерпретатору могут подключаться различные расширения, также оформленные в виде динамических библиотек. Для большинства современных приложений требуется довольно много внешних расширений, поэтому без ручного редактирования конфигурационных файлов довольно трудно обойтись.

Практически ни одна крупная программа не обходится без использования базы данных. Традиционно совместно с Web-сервером Apache и интерпретатором PHP используется СУБД MySQL.

Так как и язык программирования PHP, и Web-сервер Apache, и MySQL-сервер первоначально разработаны для UNIX-подобных операционных систем, их настройка и администрирование сводятся к редактированию конфигурационных файлов и работе в командной строке. Такой подход часто сбивает с толку программистов, не имеющих опыта работы в UNIX.

В данной главе рассматриваются приемы создания удобного рабочего места на локальной машине, которое позволит тестировать сайты без размещения их в Интернете на серверах хост-компаний.

1.2. Можно ли обойтись без утомительной настройки серверов и PHP

Конфигурирование серверов и PHP, настройка режима их работы являются важными разделами, позволяющими увереннее чувствовать себя Web-разработчикам, быстро осуществлять локализацию ошибок, попыток взлома, настраивать наиболее эффективную работу Web-приложений. Однако даже локальная настройка Web-серверов, а тем более настройка их для работы в сети Интернет, является отдельной областью деятельности, требующей зачастую не меньшей отдачи, чем программирование. Поэтому Web-разработчики зачастую отказываются от изучения конфигурирования серверов и их взаимодействия, оставляя эту область системным администраторам. Чтобы не завязнуть в многочисленных настройках серверов, правилах их связывания друг с другом, Web-разработчики прибегают к готовым пакетам, где вся настройка выполнена профессиональными администраторами. Такой пакет остается только загрузить и установить, после чего можно сразу приступить к работе с языком программирования. Наиболее популярным на сегодняшний день пакетом, объединяющим PHP, Web-сервер Apache и СУБД MySQL, является Denwer, загрузить который можно по ссылке **http://www.denwer.ru/**.

> ***Замечание***
>
> Перед установкой сверьтесь с регулярно обновляемыми деталями по адресу **http://www.softtime.ru/article/index.php?id_page=9**. По ссылке вы также можете загрузить готовые конфигурационные файлы httpd.conf (для Apache) и php.ini (для PHP). Также вы получите ответы на свои вопросы в форуме **http://www.softtime.ru/forum/index.php?id_forum=5**.

1.3. Где взять дистрибутивы

Поиски дистрибутивов следует начинать с официальных сайтов данных программных продуктов (табл. 1.1).

Таблица 1.1. *Официальные сайты PHP, Apache и MySQL*

Программный продукт	Web-сайт
Интерпретатор PHP	**http://www.php.net**
Web-сервер Apache	**http://www.apache.org**
СУБД MySQL	**http://dev.mysql.com**

Поскольку версии программных продуктов постоянно меняются, и указать постоянную ссылку невозможно, приведем алгоритм загрузки свежей версии каждого из программных продуктов.

1.3.1. Дистрибутив PHP

Загрузив страницу **http://www.php.net**, необходимо перейти на страницу **downloads**, которая на момент написания книги имела адрес **http://www.php.net/downloads.php**. На данной странице PHP доступен в формате исходных кодов **Complete Source Code**, запуск которого потребует предварительной компиляции. Для того чтобы загрузить предкомпилированный вариант **Windows Binaries**, необходимо проследовать по ссылке **http://windows.php.net/download/**. Для каждой из версий доступны два варианта компиляции в меню **Which version do I choose?** при помощи Visual Studio 6 и Visual Studio 9. Для установки на Web-сервер Apache нам потребуется версия VC6.

> ***Замечание***
>
> Версия VC9 используется для Web-сервера Windows IIS.

Необходим также выбор из вариантов **Non Thread Safe** и **Thread Safe**. Thread Safe используется для установки PHP в качестве модуля, Non Thread Safe — в качестве внешнего CGI-приложения. В дальнейшем повествование будет вестись в предположении, что выбрана версия для установки PHP в качестве модуля (**Thread Safe**).

Для загрузки лучше выбрать автоматический установщик (**Installer**), который содержит наиболее полную версию PHP. Переход по ссылке приведет на страницу со списком зеркал, откуда можно загрузить текущую версию PHP. На данной странице можно выбрать любую ссылку; разница в них заключается лишь в географическом расположении серверов. Разумнее выбрать сервер, который расположен в Российской Федерации (**Russian Federation**). Ссылки на такие серверы помечены значком российского флага.

> ***Замечание***
>
> С сайта **http://www.php.net** можно загрузить справочник по языку PHP, в том числе частично переведенный на русский язык. Для этого необходимо перейти на страницу **documentation** (**http://www.php.net/docs.php**) и там выбрать ссылку **downloads** (**http://www.php.net/download-docs.php**). На странице загрузки будет представлена таблица, позволяющая загрузить справочник в трех форматах для каждого из доступных языков. Для пользователей Windows лучше воспользоваться chm-форматом, удобным для поиска в справочнике.

1.3.2. Дистрибутив Apache

Для того чтобы тестировать и просматривать сайты на локальной машине, необходимо установить Web-сервер Apache. Поиск дистрибутива следует начать с официальной страницы **http://www.apache.org**, выбрав ссылку **Download** (**http://www.apache.org/dyn/closer.cgi**). На открывшейся странице будет представлен список HTTP- и FTP-серверов, откуда можно загрузить Web-сервер Apache.

При поиске следует помнить, что Apache может также называться httpd, по имени его UNIX-демона. На зеркалах обычно много различных файлов, например:

- httpd-2.2.15.tar.gz — для Linux, в котором программы принято распространять в исходных кодах;

- httpd-2.2.15-win32-x86-openssl-0.9.8m-r2.msi — установщик бинарных файлов под архитектуру x86 для Windows (win32) с поддержкой SSL (openssl-0.9.8m-r2) сервер Apache (httpd) версии 2.2.15 — именно его и следует загрузить.

> ***ЗАМЕЧАНИЕ***
> Скачать дистрибутивы Apache для Windows можно по адресу **http://www.sai.msu.su/apache/httpd/binaries/win32/**.

1.3.3. Дистрибутив MySQL

Дистрибутив MySQL можно загрузить со страницы **http://dev.mysql.com/downloads/**. На момент написания книги для загрузки были доступны две версии СУБД MySQL:

- MySQL 5.1 — рекомендуемая версия MySQL;
- MySQL 5.5 — разрабатываемая версия, находящаяся в стадии бета-тестирования.

Как правило, для загрузки доступны две-три версии сервера, из которых следует выбрать рекомендуемый (Generally Available Release) дистрибутив. На открывшейся странице будет представлен выпадающий список с операционными системами (по умолчанию открывается страница для Windows), для которых выводится список доступных дистрибутивов. Для Windows дистрибутивы представлены в трех вариантах (каждый из которых откомпилирован для 32- и 64-битной архитектуры):

- MSI Installer Essentials — урезанная версия дистрибутива, из которой удалены все вспомогательные утилиты;
- MSI Installer — полная версия дистрибутива MySQL, включающая автоматический установщик;
- ZIP Archive — полная версия MySQL без автоматического установщика.

Рекомендуется выбрать дистрибутив MSI Installer несмотря на то, что по объему он превышает все остальные дистрибутивы из Windows-серии. Наше изложение материала будет основано именно на этом дистрибутиве.

Помимо самого дистрибутива MySQL имеет смысл загрузить графические клиенты для работы с MySQL-сервером, которые свободно распространяются на сайте **http://dev.mysql.com/downloads/gui-tools/5.0.html**.

1.4. Установка Web-сервера Apache

Apache под Windows доступен в исходных кодах и в виде откомпилированного пакета. Исходные коды могут понадобиться в том случае, если вы решили при установке перекомпилировать Apache. Перекомпилирование исходных кодов необходимо, когда нужно исключить из исполняемой версии неиспользуемые функции или включить поддержку функций, не входящих в стандартную поставку. В этом случае необходимо наличие установленной среды разработки Microsoft Visual Studio. Если стандартная компиляция сервера вас устраивает, можно приступить к установке.

После двойного щелчка на имени файла httpd_версия_win32_*_.msi запустится Microsoft Installer (рис. 1.1). Для того чтобы начать процесс установки, нажмите кнопку **Next**, после чего появится окно с лицензионным соглашением (рис. 1.2).

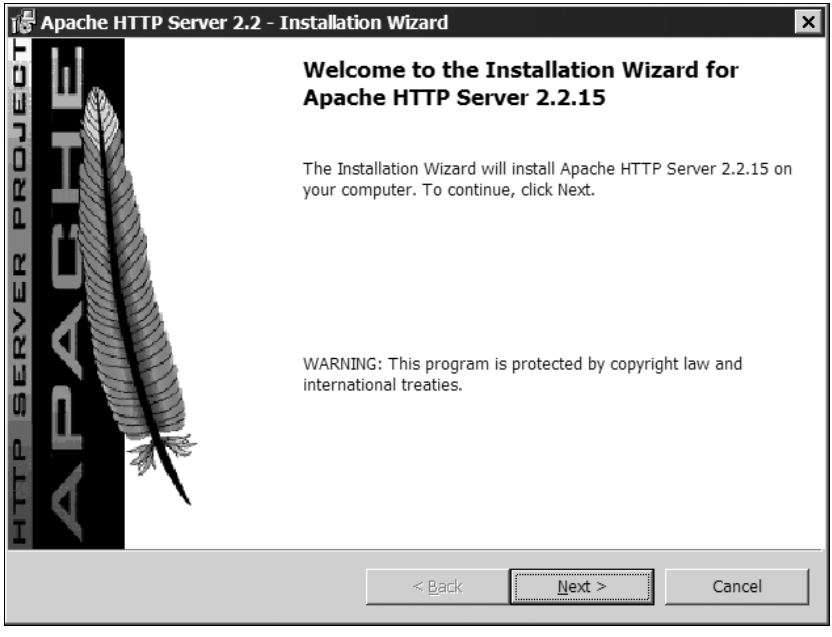


Рис. 1.1. Начало установки Web-сервера Apache

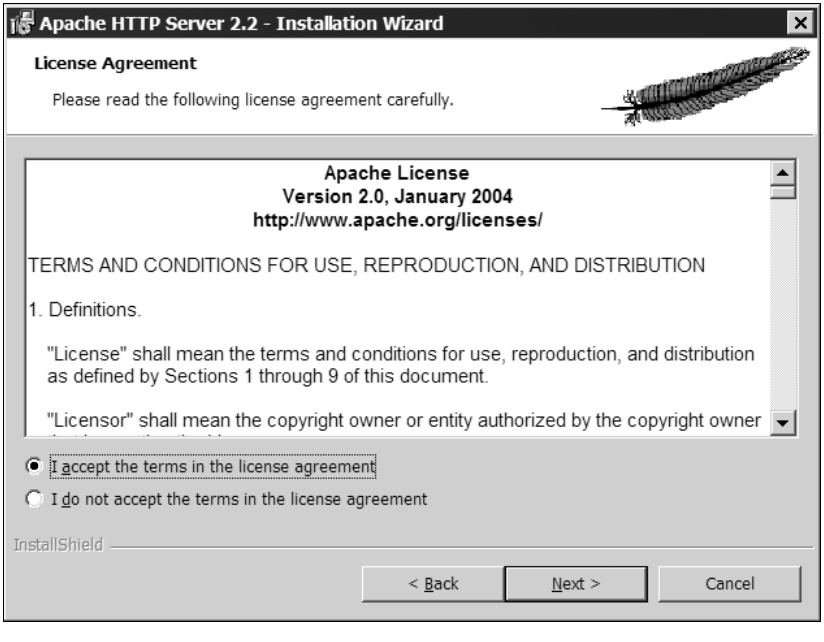


Рис. 1.2. Страница с лицензионным соглашением

После принятия лицензионного соглашения нужно перейти к следующему окну с краткой информацией о нововведениях текущий версии Apache и ссылках, по которым можно ознакомиться с ними подробнее (рис. 1.3).

Рис. 1.3. Страница с описанием нововведений

Следующее окно, показанное на рис. 1.4, позволяет ввести информацию о сервере: доменное имя сервера, имя сервера и адрес электронной почты администратора. Если установка происходит на локальную машину, то в поля для доменного имени и имени сервера следует ввести `localhost`. Очень важно заполнить эти реквизиты, иначе вместо IP-адреса 127.0.0.1 в конфигурационный Apache-файл httpd.conf будет записан IP-адрес 0.0.0.0, и сервер не сможет запуститься. В нижней части окна предлагается выбрать номер порта, по которому сервер будет принимать запросы (80 или 8080).

> ***Замечание***
>
> localhost — это псевдоним для IP-адреса 127.0.0.1, который зарезервирован для локального использования и применяется для назначения хостам в сети. Для IP-адреса 127.0.0.1 можно ввести произвольное количество псевдонимов *(см. разд. 1.5)*.

Обычно используется стандартный для протокола HTTP порт 80, однако, если для запуска не хватает прав доступа, следует выбрать порт 8080. Порт 8080 следует выбрать также в том случае, если порт 80 уже занят другим Web-сервером или программой (например, Skype).

Далее будет предложен способ установки (рис. 1.5): типичный **Typical** или выборочный **Custom**, позволяющий выбрать компоненты сервера вручную. Следующее окно (рис. 1.6) позволяет выбрать каталог установки сервера, по умолчанию это

C:\Program Files\Apache Group. Для удобства и надежности конфигурирования сервера рекомендуется изменить этот путь на любой другой, не содержащий пробелов, например, C:\www\Apache2.2 (рис. 1.7).

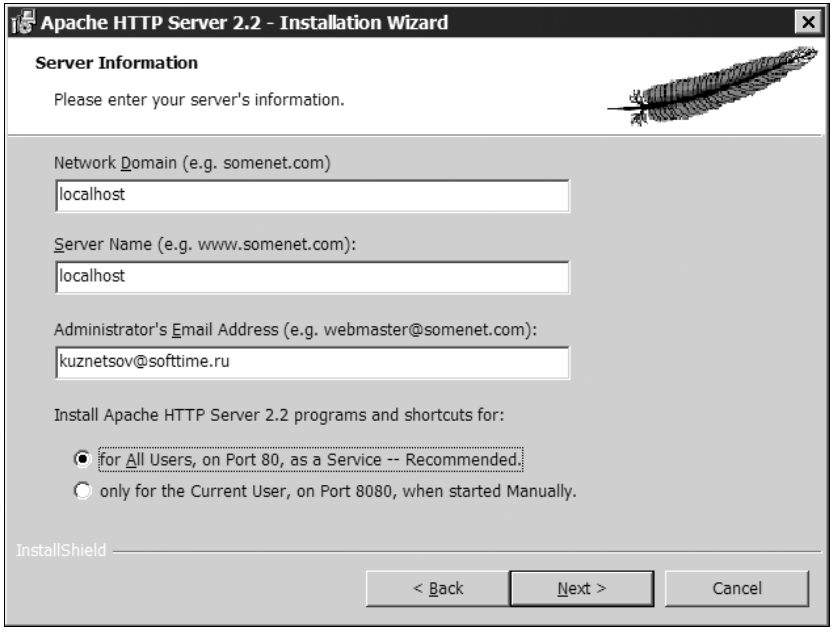


Рис. 1.4. Настройка параметров сервера

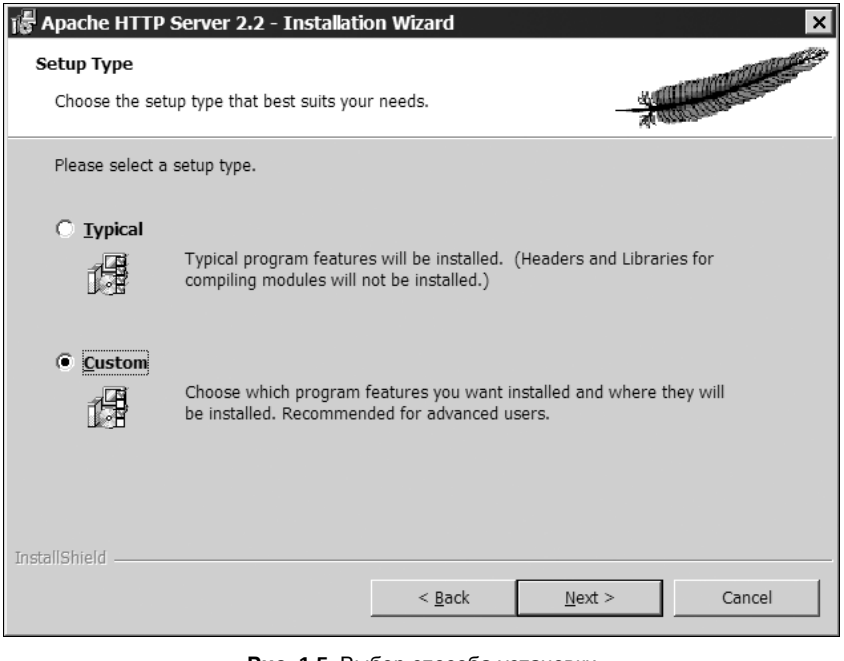


Рис. 1.5. Выбор способа установки

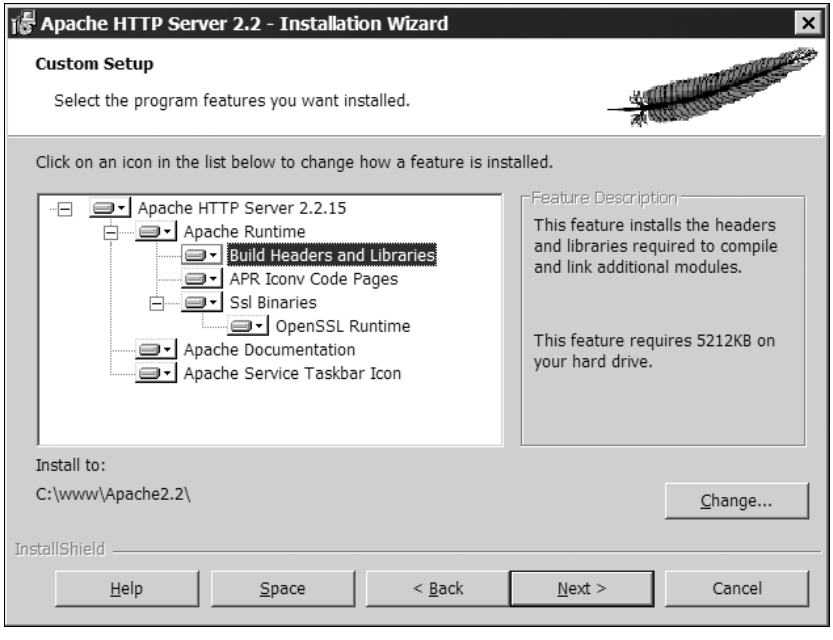


Рис. 1.6. Выбор списка компонентов и каталога установки Web-сервера

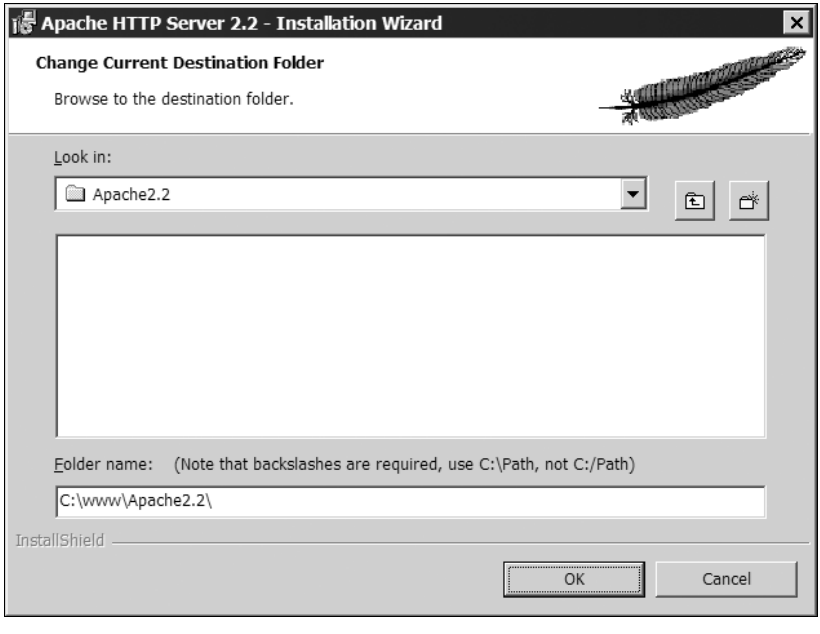


Рис. 1.7. Выбор нового каталога установки Web-сервера

Затем мастер установки сообщит о готовности к процессу установки, и после нажатия кнопки **Install** будет произведено копирование файлов сервера. Если установка прошла успешно, Windows автоматически запустит Apache. По умолчанию вместе с сервером запускается утилита мониторинга работы сервера ApachMonitor.

Замечание

Работа утилиты ApachMonitor не влияет на работу сервера, и ее можно отключить. Если далее она потребуется, ее можно найти в подкаталоге bin каталога установки Apache.

Проверить работоспособность сервера можно, набрав в браузере **http://localhost**. В случае успешной установки вы увидите фразу "It works!" (рис. 1.8).

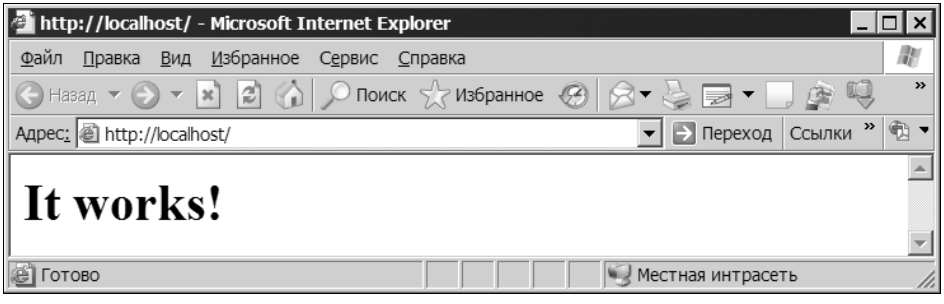


Рис. 1.8. Приветственная страница Web-сервера Apache

1.5. Настройка виртуальных хостов

После того как установка Web-сервера Apache успешно завершена, необходимо создать виртуальный хост. Виртуальный хост позволяет указать каталог, где будут располагаться HTML- и PHP-файлы. Кроме этого, виртуальные хосты позволяют перенести HTML- и PHP-файлы в другой раздел диска, что может быть удобно при резервном копировании данных.

Виртуальный хост позволяет связать Web-адрес с IP-адресом, причем каждый из виртуальных хостов можно связать с отдельной папкой, расположенной в произвольном месте файловой системы. Например, вместо **http://localhost** можно использовать имена в вымышленной DNS-зоне **http://site.dev**, **http://project.dev** и т. п. Можно также использовать имена в зоне RU (или любой другой реальной зоне) **http://site.ru** или **http://project.ru**, однако следует помнить, что при этом теряется возможность обращаться к удаленным сайтам с такими именами.

Традиционно для локальных виртуальных хостов выбирается IP-адрес 127.0.0.1, однако в случае операционных систем Windows Vista и Windows 7 настроить виртуальные хосты на этом IP-адресе не удается из-за использования его внутри системы. В случае этих операционных систем везде в конфигурационных файлах лучше использовать IP-адрес 127.0.0.2.

В первую очередь необходимо прописать имена виртуальных хостов в операционной системе в файле C:/Windows/system32/drivers/etc/hosts. Файл является обычным текстовым файлом, в котором IP-адресу сопоставляется доменное имя (листинг 1.1).

Листинг 1.1. Возможное содержимое файла hosts

```
# (C) Корпорация Майкрософт (Microsoft Corp.), 1993-1999
#
# Это образец файла HOSTS, используемый Microsoft TCP/IP для Windows.
#
# Этот файл содержит сопоставления IP-адресов именам узлов.
# Каждый элемент должен располагаться в отдельной строке. IP-адрес должен
# находиться в первом столбце, за ним должно следовать соответствующее
# имя. IP-адрес и имя узла должны разделяться хотя бы одним пробелом.
#
# Кроме того, в некоторых строках могут быть вставлены комментарии
# (такие, как эта строка), они должны следовать за именем узла и
# отделяться от него символом '#'.
#
# Например:
#
#      102.54.94.97     rhino.acme.com          # исходный сервер
#       38.25.63.10     x.acme.com              # узел клиента x

127.0.0.1 site.dev
127.0.0.1 www.site.dev
127.0.0.1 project.dev
127.0.0.1 www.project.dev
```

В файл host можно добавить произвольное количество доменных имен. Далее необходимо отредактировать конфигурационные файлы Apache, чтобы он получил возможность прослушивать и сортировать запросы к разным виртуальным хостам.

> ***ЗАМЕЧАНИЕ***
>
> Известный префикс www на самом деле является доменом третьего уровня, и если не создать для него привязки к IP-адресу и не настроить Apache на его отображение, он будет недоступен.

Открыв каталог, в который был установлен Web-сервер Apache, можно обнаружить подкаталог conf, который содержит глобальные конфигурационные файлы. Главным конфигурационным файлом является httpd.conf. Любая строка в файле, начинающаяся с символа диеза #, является комментарием и не влияет на работоспособность сервера. В конце конфигурационного файла необходимо найти строки из листинга 1.2 и убрать символ диеза напротив строки, начинающейся с `Include`. Этим самым мы подключаем дополнительный конфигурационный файл httpd-vhosts.conf, который можно обнаружить в подкаталоге extra.

Листинг 1.2. Фрагмент конфигурационного файла httpd.conf

```
# Virtual hosts
Include conf/extra/httpd-vhosts.conf
```

Файл httpd-vhosts.conf содержит пример создания виртуальных хостов. Сначала при помощи директивы `NameVirtualHost` требуется указать, какой IP-адрес и порт используются для виртуального хоста:

```
NameVirtualHost 127.0.0.1:80
```

> ***ЗАМЕЧАНИЕ***
>
> В конце IP-адреса указывает порт, который будет прослушивать сервер Apache. Так, если необходимо назначить Web-серверу альтернативный порт, можно указать вместо 80 порт 8080. В этом случае при обращении к серверу придется явно указывать порт в адресе: **http://localhost:8080/index.html**, поскольку если порт не указывается, браузер автоматически обращается по порту 80.

Далее необходимо создать контейнер `<VirtualHost>`, который будет определять конфигурацию виртуального хоста (листинг 1.3).

Листинг 1.3. Контейнер `<VirtualHost>`

```
<VirtualHost 127.0.0.1:80>
    ServerAdmin kuznetsov@softtime.ru
    DocumentRoot D:/data
    ServerName site.dev
    ServerAlias www.site.dev
    ErrorLog logs/dummy-host.example.com-error_log
    CustomLog logs/dummy-host.example.com-access_log common
</VirtualHost>
```

Адрес в контейнере виртуального хоста должен совпадать с адресом, указанным в директиве `NameVirtualHost`. В листинге 1.3 директива `ServerAdmin` определяет адрес электронной почты администратора. Именно этот адрес будет выводиться при генерации Web-сервером страниц с сообщениями об ошибках.

Директива `DocumentRoot` определяет физическое расположение виртуального хоста на жестком диске. Теперь файлы можно размещать в каталоге D:\data\, и они будут доступны при обращении к адресу **http://site.dev/**.

> ***ЗАМЕЧАНИЕ***
>
> Если в пути к каталогу виртуального хоста встречаются пробелы, путь следует заключать в двойные кавычки.

Директива `ServerName` задает имя сервера, которое в данном случае может быть любым. Директива `ServerAlias` позволяет задать псевдонимы для текущего виртуального хоста, именно эта директива наиболее удобна для связывания виртуального хоста с доменом третьего уровня, обеспечивающего адресу сайта префикс www. Директивы `ErrorLog` и `CustomLog` определяют путь к файлам журналов (log-файлам), в которые Web-сервер Apache помещает ошибки и фиксирует обращения к страницам сайта.

Для того чтобы виртуальные хосты смогли работать в других разделах, в конфигурационном файле httpd.conf необходимо найти контейнер `<Directory />` и исправить его в соответствии с листингом 1.4.

Листинг 1.4. Изменения, вносимые в контейнер `<Directory />`

```
<Directory />
    Options All
    AllowOverride All
    Order deny,allow
</Directory>
```

Чтобы изменения вступили в силу, необходимо перезагрузить Web-сервер *(см. разд. 1.6–1.7)*.

Для проверки работоспособности виртуального хоста необходимо создать каталог D:\data\ и разместить в нем файл index.html следующего содержания (листинг 1.5).

Листинг 1.5. Содержимое файла index.html

```
Это файл index.html виртуального хоста www.site.dev
```

Теперь, если перезагрузить сервер и набрать в адресной строке адрес **http://www.site.dev**, можно увидеть страницу, представленную на рис. 1.9.

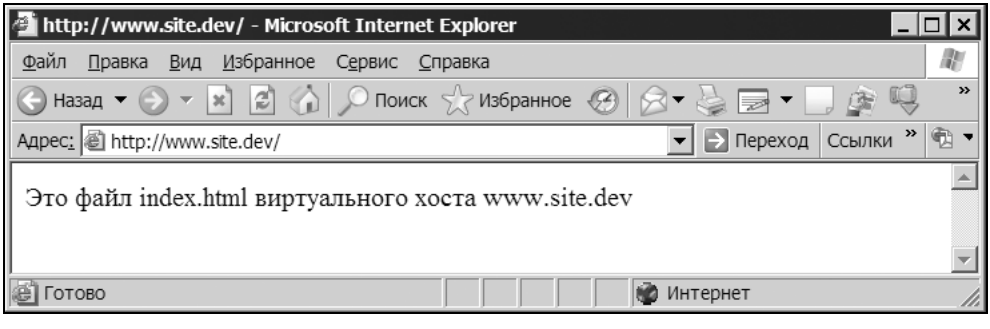


Рис. 1.9. Страница виртуального хоста **www.site.dev**

1.6. Управление запуском и остановкой Web-сервера Apache

Если при установке сервера был выбран порт 80 (см. рис. 1.4), допускается запуск Apache в качестве сервиса. Для запуска консоли управления сервисами выполните команду **Пуск | Настройка | Панель управления | Администрирование | Службы**. В появившемся окне консоли, приведенном на рис. 1.10, следует выбрать сервис Apache2.2. Контекстное меню позволяет осуществлять запуск, остановку и перезапуск сервиса.

Службы Windows позволяют производить запуск фоновых приложений при старте системы. Для этого необходимо перейти в окно **Свойства**, выбрав в контекстном меню сервиса пункт **Свойства** (рис. 1.10), и в появившемся окне (рис. 1.11) в выпадающем списке **Тип запуска** выбрать пункт **Авто**.

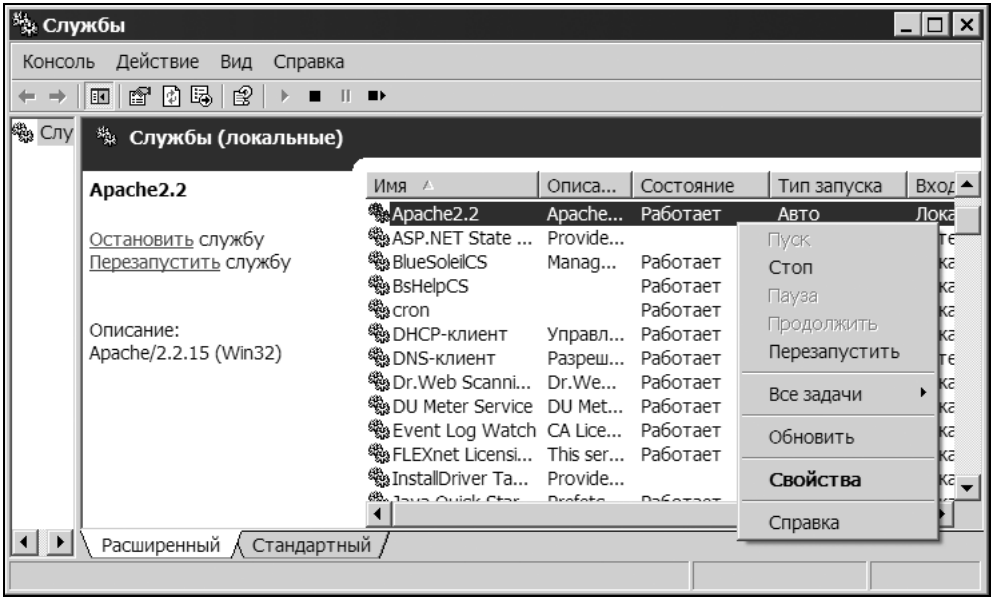


Рис. 1.10. Службы Windows

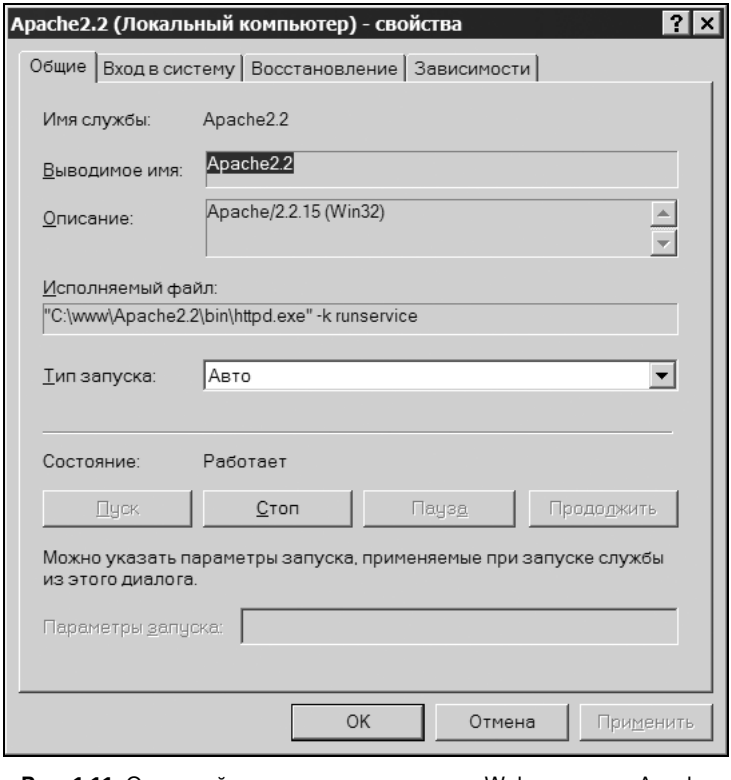


Рис. 1.11. Окно свойства сервиса управления Web-сервером Apache

1.7. Управление Apache из командной строки

В Windows запускать, останавливать и перезапускать сервер Apache из командной строки можно при помощи следующих команд:

- `Apache -k start` (запуск);
- `Apache -k restart` (перезапуск);
- `Apache -k stop` или `Apache -k shutdown` (остановка).

Все команды следует выполнять из каталога bin сервера Apache (C:\www\Apache2.2\bin\). Команды с ключом `-k` являются управляющими командами сервера Apache. Так команды `Apache -k install` и `Apache -k uninstall` позволяют установить и удалить сервис Apache2.2. Получить полный список команд управления с их кратким описанием можно командой `Apache -help` или в документации к серверу Apache.

> ***ЗАМЕЧАНИЕ***
>
> Запустить командную строку можно, выбрав пункт меню **Пуск | Программы | Стандартные | Командная строка**. Выбор диска осуществляется как `D:`, смена текущего каталога — командой `cd`, например, `cd C:\www\Apache2.2\bin`.

Команда `Apache -t` позволяет проверить конфигурационные файлы Apache на предмет наличия синтаксических ошибок. В случае их отсутствия выдается строка `"Syntax OK"`. Если же в конфигурационных файлах имеются ошибки, то в результате тестирования программа выдаст сообщение об ошибке, например:

```
Syntax error on line 57 of C:/www/Apache2.2/conf/ httpd.conf:
ServerRoot takes one argument, Common directory of server-related files.
```

Если сервис Apache2.2 успешно установлен, как это описано в предыдущем разделе, то управлять запуском и остановкой Web-сервера можно при помощи системной команды `NET`:

- `NET START Apache2.2` — запуск сервиса;
- `NET STOP Apache2.2` — остановка сервиса.

1.8. Установка PHP

После двойного щелчка на файле php-версия-win32-installer.msi запустится Microsoft Installer (рис. 1.12).

Для того чтобы начать процесс установки, нажмите кнопку **Next**, после чего появится окно с лицензионным соглашением (рис. 1.13). Следующая страница предлагает выбрать путь установки PHP (рис. 1.14).

Осуществив переход к следующей странице, можно обнаружить список Web-серверов и способов взаимодействия с ними, среди которых следует выбрать пункт Apache 2.2.x Module (рис. 1.15). Следующая страница позволяет указать расположение Web-сервера Apache (рис. 1.16).

Рис. 1.12. Начало установки PHP

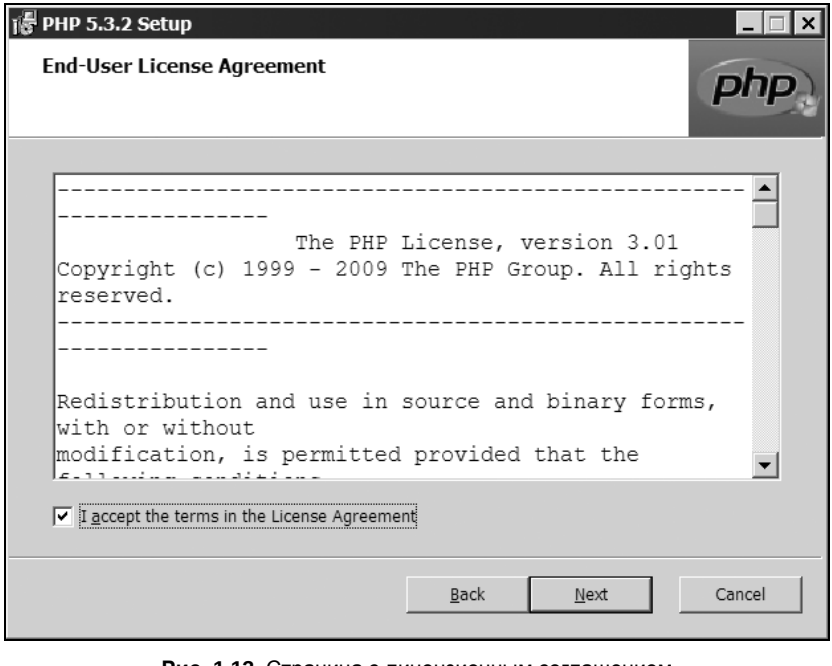


Рис. 1.13. Страница с лицензионным соглашением

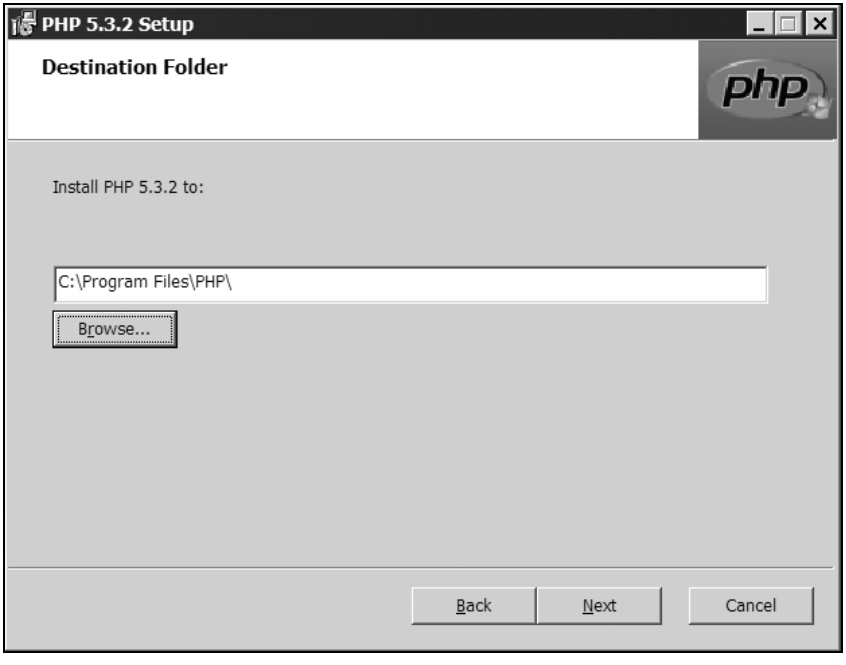


Рис. 1.14. Выбор пути установки PHP

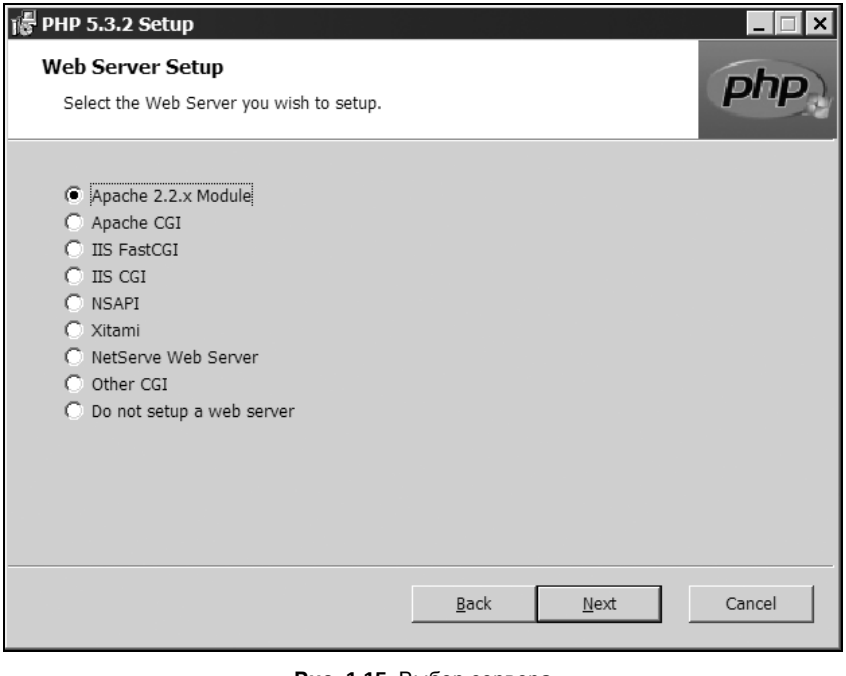


Рис. 1.15. Выбор сервера

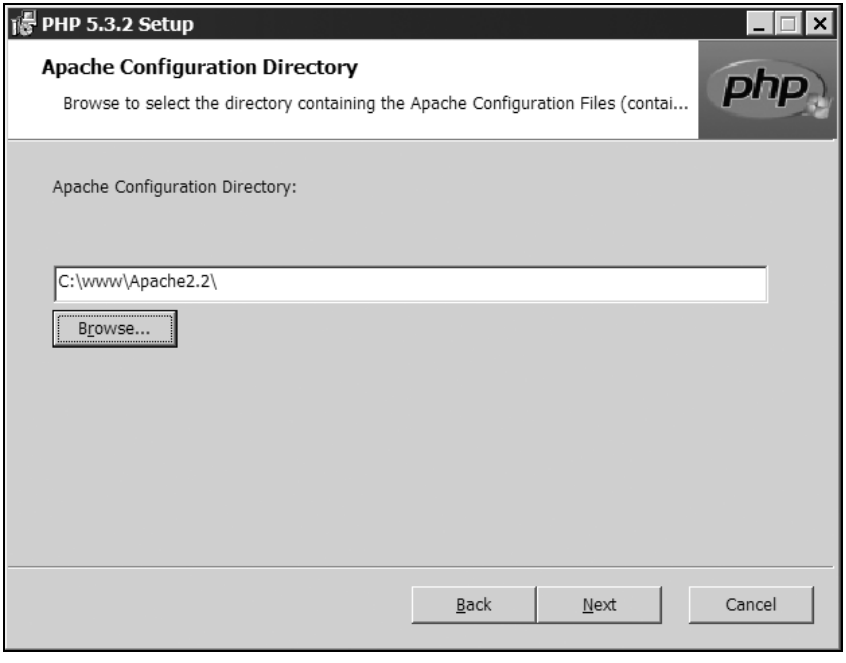


Рис. 1.16. Страница, запрашивающая расположение корневого каталога Web-сервера Apache

На следующей странице отображается список компонентов, доступных для установки по умолчанию (рис. 1.17). Рекомендуется оставить настройки на данной странице без изменения.

Замечание

Теоретически, на данной странице можно выбрать набор расширений, которые будут доступны для работы из PHP, на практике настройку расширений лучше выполнить вручную через конфигурационный файл PHP — php.ini.

Затем мастер установки сообщит о готовности к процессу установки, и после нажатия кнопки **Install** будет произведено копирование файлов PHP и редактирование конфигурационного файла httpd.conf. Если сравнить данные файлы до и после установки, можно заметить, что в него добавляется всего лишь три строки, представленные в листинге 1.6.

Замечание

В конфигурационных файлах программ, портированных из UNIX, в пути традиционно используется прямой /, а не обратный слэш \.

Листинг 1.6. Подключение PHP в httpd.conf

```
AddType application/x-httpd-php .php
PHPIniDir "C:/Program Files/PHP/"
LoadModule php5_module "C:/Program Files/PHP/php5apache2_2.dll"
```

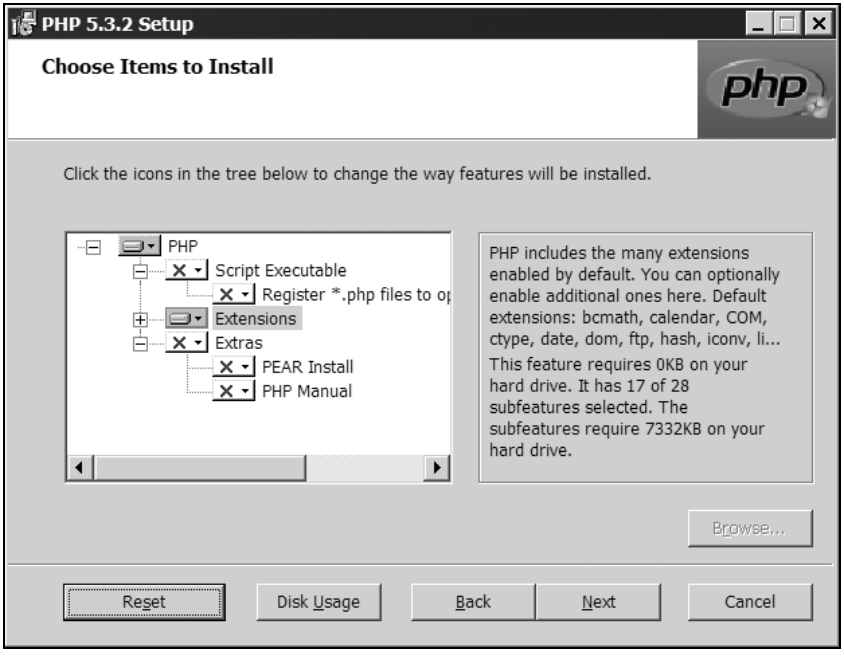


Рис. 1.17. Выбор компонентов для установки

Первая директива `AddType` связывает файлы с расширением php и PHP-интерпретатор. Если помимо расширения php необходимо, чтобы и другие файлы рассматривались как PHP-файлы, их расширения можно добавить в этой строке через пробел. Вторая директива `PHPIniDir` указывает путь к местоположению конфигурационного файла PHP — php.ini, третья директива `LoadModule` подключает интерпретатор PHP для Web-сервера Apache 2.2.

Индексный файл — это точка входа для папки. Создав такой файл, можно обращаться по сокращенному пути, например, **http://site.dev/**, вместо **http://sive.dev/index.php**. По умолчанию индексным файлом является index.html. Чтобы добавить новые индексные файлы, включая index.php, найдите в конфигурационном файле httpd.conf директиву `DirectoryIndex` и исправьте ее в соответствии с листингом 1.7.

Замечание

Изменения в конфигурационном файле httpd.conf вступают в силу после перезагрузки сервера.

Листинг 1.7. Настройка индексных файлов

```
DirectoryIndex index.php index.html index.htm
```

В директиве `DirectoryIndex` можно указать любые другие индексные файлы, например, index.htm, index.php3, index.phtml и т. п.

Для проверки работоспособности PHP-файла в каталоге виртуального хоста необходимо создать PHP-файл index.php следующего содержания (листинг 1.8).

Листинг 1.8. Проверочный PHP-скрипт

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

Функция `phpinfo()` выводит в окно браузера отчет о конфигурации PHP. Если PHP успешно связан с Web-сервером, то результат может выглядеть, как на рис. 1.18.

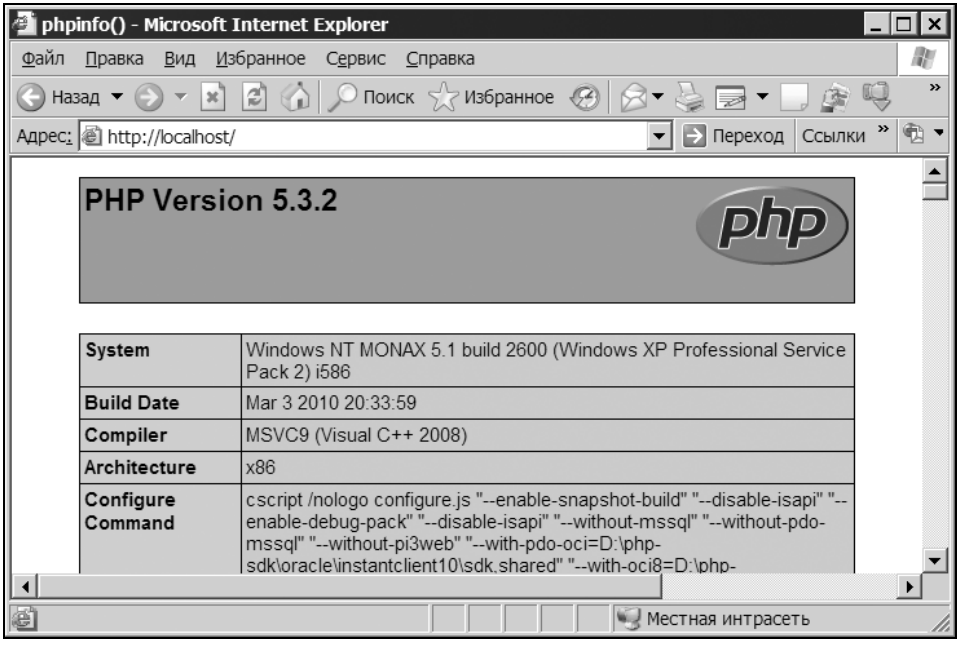


Рис. 1.18. Отчет функции `phpinfo()`

Следует отметить, что отчет, предоставляемый функцией `phpinfo()`, указывает на путь к конфигурационному файлу php.ini. Его можно выяснить в строке `Loaded Configuration File`. Этот путь может быть полезным, если имеется подозрение, что редактируется не та копия, которая используется PHP (например, внесенные изменения не вступают в силу).

ЗАМЕЧАНИЕ

Если при установке сервера MySQL у вас возникли затруднения, вы можете обратиться на форум по адресу **http://www.softtime.ru/forum/index.php?id_forum=5**.

1.9. Что предпринять, если Web-сервер не запускается

Конфигурационные файлы удобны для ручного редактирования, и очень часто настройка при помощи автоматического установщика может приводить к тому, что

Web-сервер перестает запускаться. Главная причина обычно (если не допущена синтаксическая ошибка в одной из директив) заключается в том, что Web-сервер не может обнаружить динамических библиотек расширений или связанных с ними вспомогательных библиотек.

В первую очередь следует заглянуть в конфигурационный файл php.ini, в конце которого можно обнаружить список расширений, представленных в листинге 1.9.

Листинг 1.9. Фрагмент конфигурационного файла php.ini

```
[PHP_BZ2]
extension=php_bz2.dll
[PHP_CURL]
extension=php_curl.dll
[PHP_GD2]
extension=php_gd2.dll
[PHP_IMAP]
extension=php_imap.dll
[PHP_MBSTRING]
extension=php_mbstring.dll
[PHP_MYSQL]
extension=php_mysql.dll
[PHP_MYSQLI]
extension=php_mysqli.dll
[PHP_OPENSSL]
extension=php_openssl.dll
[PHP_PDO_MYSQL]
extension=php_pdo_mysql.dll
[PHP_PDO_ODBC]
extension=php_pdo_odbc.dll
[PHP_SOCKETS]
extension=php_sockets.dll
```

Если Web-сервер отказывается стартовать, закомментируйте все расширения, расположив в начале строки символ точки с запятой. Если после этого сервер стартует, последовательно убирайте комментарии с тех расширений, которые вам нужны в работе. Большинство полезных расширений будут рассмотрены далее в книге, поэтому подключать их можно по мере надобности.

Замечание

Более подробно директивы конфигурационного файла php.ini рассматриваются в *главе 2*.

1.10. Установка СУБД MySQL

При работе в Windows необходимо зарегистрироваться в системе с привилегиями администратора и запустить установочный файл. В результате появится окно мастера установки, показанное на рис. 1.19.

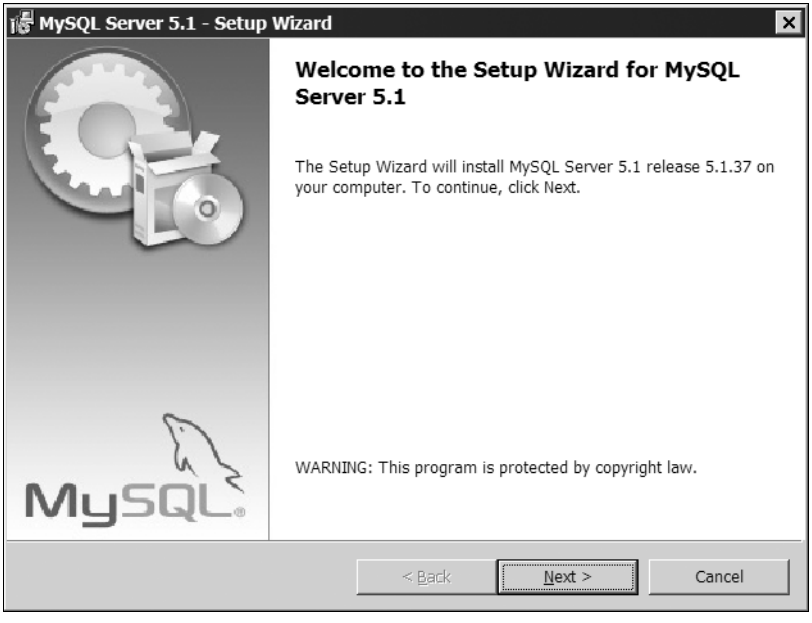


Рис. 1.19. Стартовое окно автоматического установщика

ЗАМЕЧАНИЕ

Перед установкой сервера MySQL рекомендуется отключить Firewall, если он имеется в системе, и включить его только после того, как сервер MySQL успешно установлен. Если после этого сервер MySQL прекратит принимать запросы, в Firewall следует открыть порт 3306 (или другой порт, если значение порта будет изменено во время установки).

Для продолжения установки следует нажать кнопку **Next**, после чего откроется окно, показанное на рис. 1.20, в котором предлагается выбрать способ установки:

- **Typical** (Стандартный);
- **Complete** (Полный);
- **Custom** (Выборочный).

В первом случае устанавливаются стандартные компоненты, во втором — все возможные компоненты, в третьем — по выбору пользователя.

В режиме **Custom** нажмите кнопку **Next** и в появившемся окне (рис. 1.21) выберите необходимые компоненты. Компоненты, которые по умолчанию отключены, перечеркнуты красным крестиком. Для того чтобы установить предлагаемые компоненты, необходимо выделить нужный компонент и выбрать в выпадающем меню пункт **This feature will be installed on local hard drive.** Если у вас отсутствует опыт установки MySQL, при первой установке лучше ничего не менять.

После того как все компоненты выбраны, в диалоговом окне, показанном на рис. 1.21, можно изменить каталог установки, нажав кнопку **Change.**

ЗАМЕЧАНИЕ

По умолчанию установка производится в каталог C:\Program Files\MySQL\MySQL Server 5.1\.

Рис. 1.20. Выбор режима установки MySQL

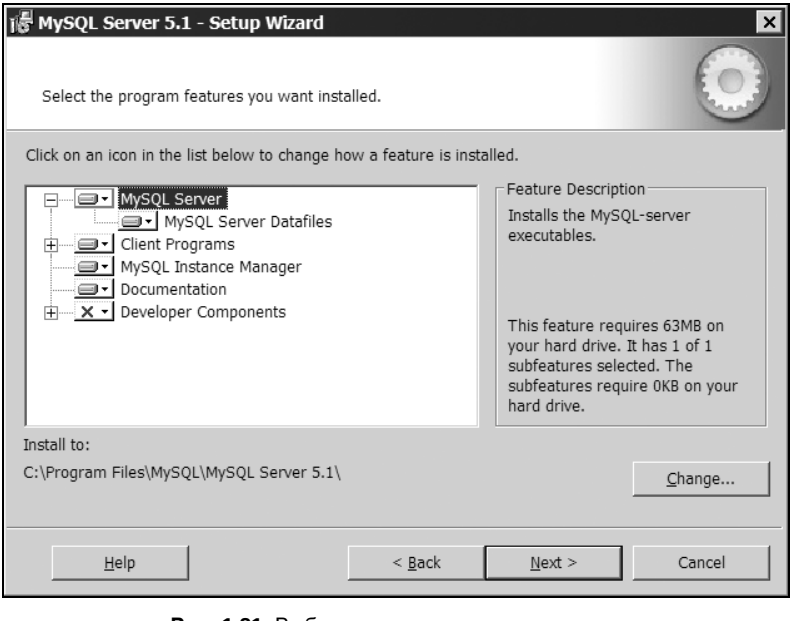


Рис. 1.21. Выбор компонентов для установки

Рекомендуется изменить путь по умолчанию на более короткий, например, C:\MySQL\. Нажав кнопку **Change**, введите путь `C:\MySQL\` (рис. 1.22).

Если вы устанавливаете MySQL поверх старой копии, будьте осторожны. Такой способ часто применяется, чтобы базы данных старой версии перешли в новую ин-

сталляцию. Однако следует помнить, что системная база данных `mysql` в настоящий момент несовместима с предыдущими версиями. Поэтому если вы производите обновление MySQL, помимо остановки сервера следует удалить подкаталог C:\mysql\data\mysql системной базы данных `mysql`. При первой инсталляции проблем, связанных с несовместимостью, возникать не должно.

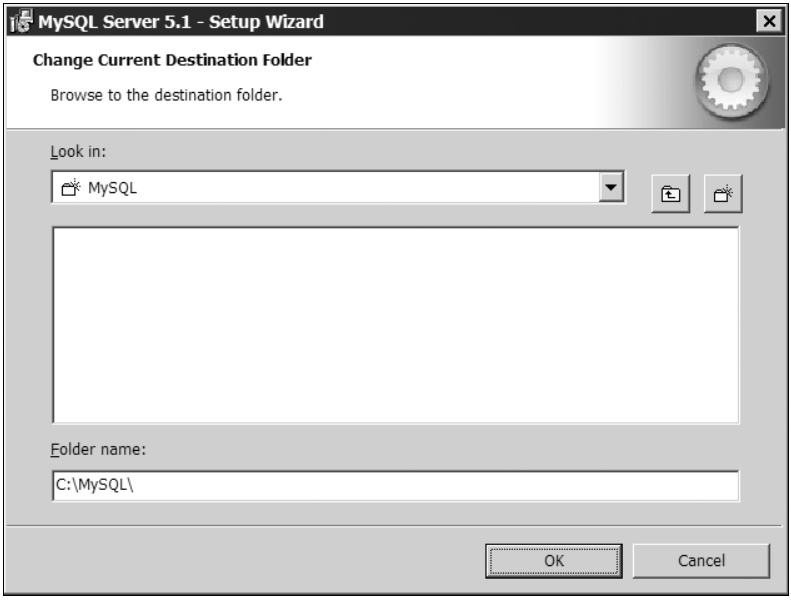


Рис. 1.22. Выбор каталога, в который будет установлен MySQL-сервер

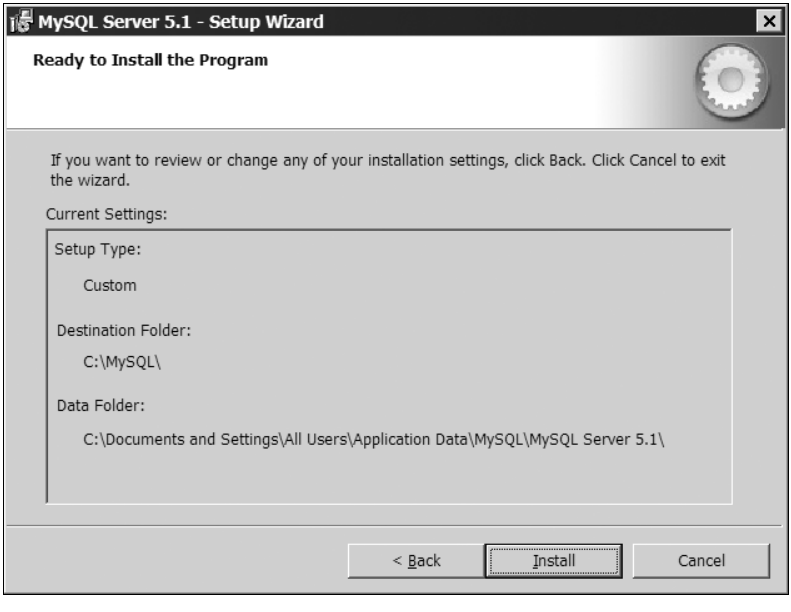


Рис. 1.23. Завершающее окно перед процессом инсталляции

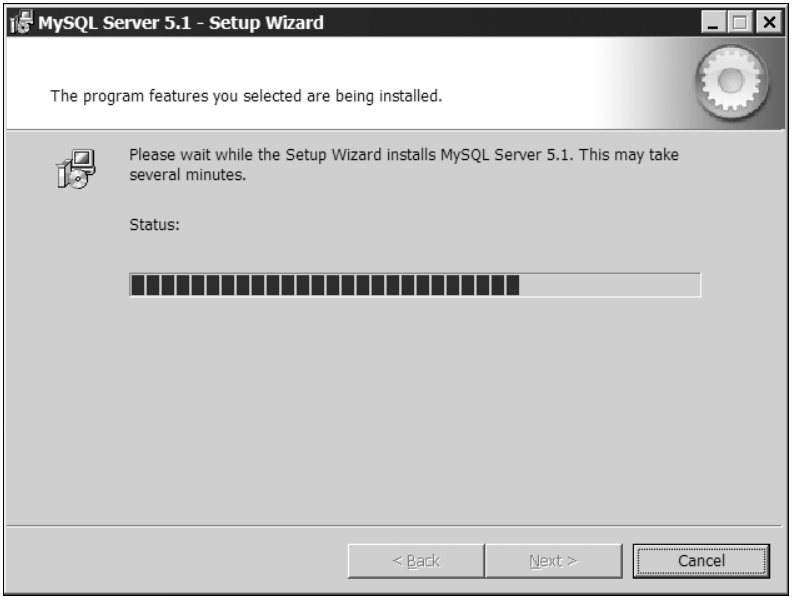


Рис. 1.24. Копирование файлов

После завершения настройки выводится окно, представленное на рис. 1.23. Нажатие кнопки **Install** начинает процесс копирования файлов (рис. 1.24).

После копирования файлов мастер установки предлагает зарегистрировать сервер и выполнить послеустановочную настройку (рис. 1.25). Регистрация не является обязательной процедурой, однако выполнить настройку сервера рекомендуется.

Рис. 1.25. Предложение зарегистрироваться на сайте **http://www.mysql.com**

1.11. Послеустановочная настройка MySQL

Сразу после установки MySQL запускается утилита послеустановочной настройки. Данный этап можно пропустить, учитывая, что к настройке всегда можно вернуться, выбрав пункт системного меню **Пуск | Программы | MySQL | MySQL Server 5.1 | MySQL Server Instance Config Wizard**. Тем не менее, рекомендуется сразу произвести настройку системы.

Настройка начинается со стартового окна (рис. 1.26). После нажатия кнопки **Next** открывается окно (рис. 1.27), в котором предлагается выбрать режим настройки:

- **Detailed Configuration** (Подробный);
- **Standard Configuration** (Стандартный).

Рис. 1.26. Настройка MySQL при помощи мастера MySQL Server Instance Configuration Wizard

Для более гибкой настройки системы следует выбрать первый пункт (**Detailed Configuration**). После нажатия кнопки **Next** открывается окно настройки производительности MySQL (рис. 1.28) со следующими опциями:

- **Developer Machine** (Машина разработчика);
- **Server Machine** (Сервер);
- **Dedicated MySQL Server Machine** (Выделенный сервер).

Все три опции различаются по интенсивности использования процессора, объему требуемой оперативной памяти и жесткого диска. Следует выбрать первый пункт, т. к. в этом случае MySQL занимает наименьший объем оперативной памяти, не мешая работе других приложений. Второй пункт предназначен для сервера, на котором, помимо MySQL, будут работать другие серверы, например, Web-сервер или транспортный почтовый агент. Третий пункт предназначен для выделенного сервера MySQL, на котором не будут выполняться никакие другие приложения.

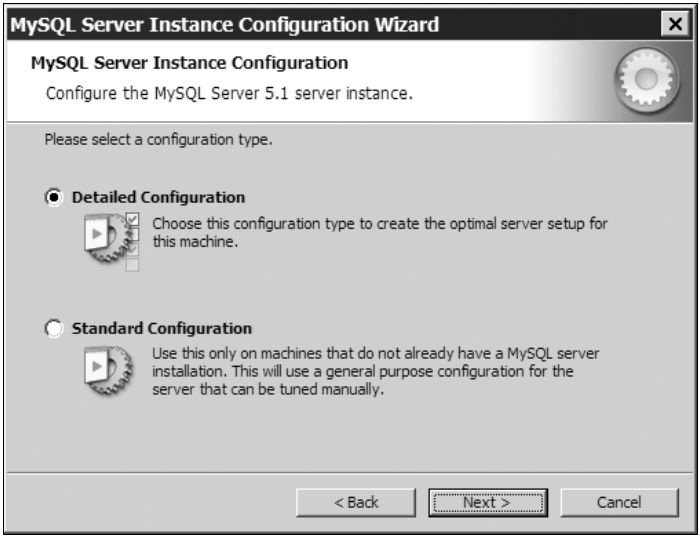


Рис. 1.27. Выбор режима настройки

Рис. 1.28. Настройка производительности сервера MySQL

Окно, представленное на рис. 1.29, позволяет выбрать для таблиц предпочтительный тип, который назначается по умолчанию. Следует оставить первый пункт.

Замечание

Результатом работы утилиты MySQL Server Instance Configuration Wizard является конфигурационный файл C:\MySQL5\my.ini, который позже можно отредактировать вручную.

В окне на рис. 1.29 предлагается выбор, по сути, между двумя типами таблиц: MyISAM и InnoDB. Главное различие заключается в том, что под каждую таблицу MyISAM создается отдельный файл, который хранится в соответствующей базе

данных каталога, а все таблицы InnoDB хранятся в едином табличном пространстве, под которое выделяется один файл. Кроме этого, скорость работы с MyISAM в несколько раз превышает скорость работы с таблицами типа InnoDB. Однако таблицы InnoDB поддерживают транзакции на уровне строк, более надежны и позволяют работать с таблицами большого объема. При первом знакомстве рекомендуется выбрать более простые в обслуживании и настройке таблицы типа MyISAM.

Выбор диска для хранения этого файла и пути, куда он будет помещен, осуществляется в следующем окне, представленном на рис. 1.30.

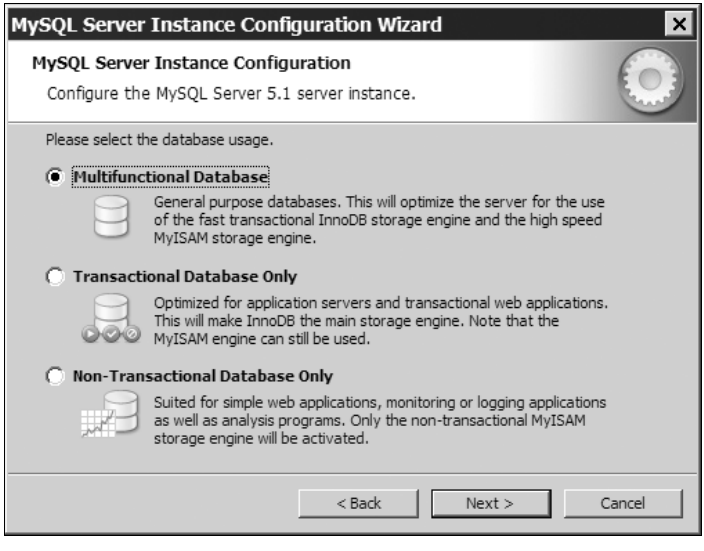


Рис. 1.29. Выбор типа таблиц по умолчанию

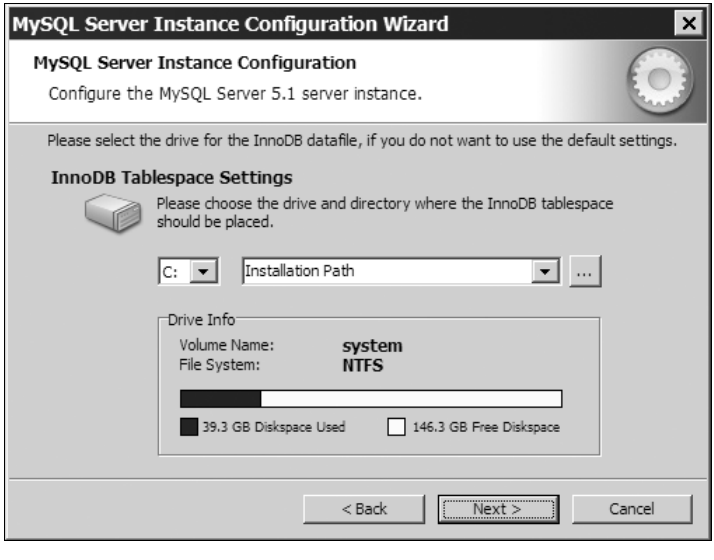


Рис. 1.30. Выбор пути для хранения файла под таблицы InnoDB

Следующее окно (рис. 1.31) предлагает указать максимальное количество клиентов, которые смогут одновременно подключаться к серверу. Первый пункт (рекомендуется выбрать именно его) предполагает, что количество таких соединений не будет превышать 20, второй пункт допускает 500 соединений с сервером, третий позволяет назначить собственное ограничение количества активных соединений.

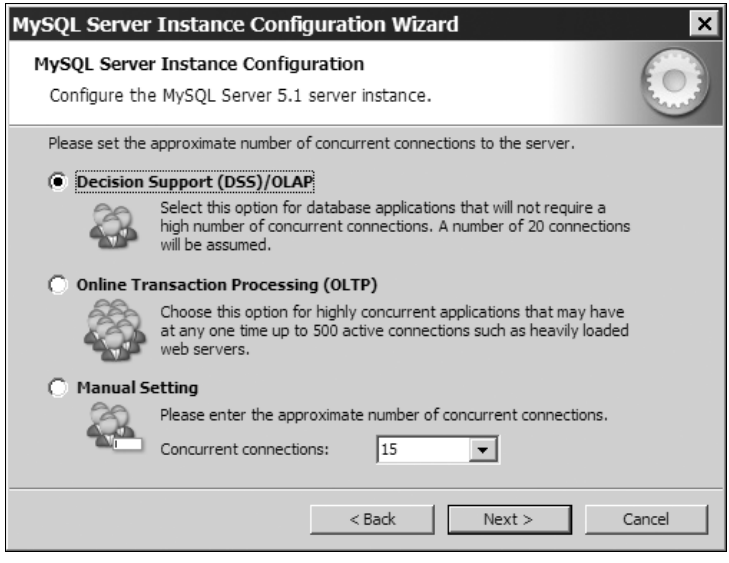


Рис. 1.31. Выбор максимального количества активных соединений с сервером

На реальных серверах не следует устанавливать это число более 200, обычно, если оперативной памяти выделено достаточно и производительность процессора (процессоров) высока, запросы выполняются очень быстро, не достигая этого предела. Как только один из компонентов сервера начинает потреблять лишние ресурсы, все остальные процессы ожидают выполнения и накапливаются в очереди. Так как в памяти накапливается множество невыполненных запросов, ситуация еще больше усугубляется. Ограничение количества одновременных запросов позволяет при такой критической ситуации отбрасывать все новые запросы. В результате после того, как пиковая нагрузка минует, сервер выполняет все накопившиеся в очереди запросы, и ситуация нормализуется. При отсутствии такого ограничения накапливающие запросы приведут к зависанию сервера и необходимости ручного вмешательства для его перезапуска.

Замечание

Если ситуация с превышением максимального количества одновременных соединений воспроизводится регулярно и вызвана не утечками памяти или перегрузкой процессора, следует увеличить ограничение на количество одновременно выполняющихся запросов.

В следующем окне (рис. 1.32) устанавливается номер порта, по которому будет происходить соединение клиентов с MySQL-сервером (по умолчанию — 3306). Если на компьютере не имеется других версий MySQL, рекомендуется сохранить значение по умолчанию, т. к. порт 3306 является стандартным для MySQL.

Примечание

В том случае, если необходимо запускать MySQL-сервер совместно с другими версиями MySQL, можно выбрать другой номер порта, отличный от тех, по которым происходит обращение к прочим MySQL-серверам.

В следующем окне предлагается указать кодировку по умолчанию (рис. 1.33). Необходимо выбрать ручной выбор кодировки и в выпадающем списке выделить пункт **cp1251**, соответствующий русской Windows-кодировке.

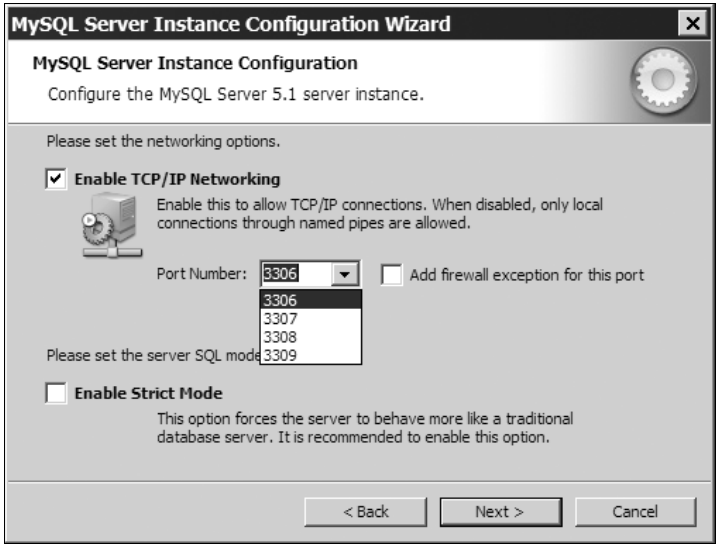


Рис. 1.32. Выбор порта, по которому сервер MySQL будет слушать клиентские запросы

Рис. 1.33. Выбор кодировки на MySQL-сервере по умолчанию

При работе в среде Windows можно установить MySQL в качестве сервиса, что обеспечит запуск сервера mysqld.exe при старте системы и корректное завершение работы сервера при выключении компьютера.

Окно, представленное на рис. 1.34, предназначено для настройки такого сервиса. Установка флажка **Install As Windows Service** позволяет установить сервис с именем, которое следует выбрать в выпадающем списке **Service Name**. Можно изменить имя сервиса, особенно если в системе присутствует инсталлированный MySQL-сервер более ранней версии, это позволит избежать конфликтов при запуске серверов как MySQL 5, так и более ранней версии. Отметка флажка **Launch the MySQL Server automatic** позволяет настроить сервис на автоматический режим работы, когда запуск сервера производится со стартом системы, а остановка — при завершении работы с системой. В противном случае запуск и остановку сервера придется выполнять вручную. Как будет показано далее, можно изменить режим работы сервиса в любой момент в консоли управления сервисами.

Флажок **Include Bin Directory in Windows PATH** позволяет прописать путь к каталогу C:\MySQL\bin в системной переменной `PATH`, что может быть удобным при частом использовании утилит из этого каталога.

Рис. 1.34. Настройка сервиса для автоматического запуска MySQL-сервера

В следующем окне (рис. 1.35) производится настройка учетных записей. Если вы не знакомы с системой авторизации MySQL и осуществляете установку первый раз, рекомендуется снять флажок **Modify Security Settings** и оставить данные настройки по умолчанию. Два текстовых поля позволяют задать пароль для суперпользователя `root` (в противном случае в качестве пароля будет выступать пустая строка). Флажок **Create An Anonymous Account** позволяет создать анонимного пользователя. Такой пользователь обладает минимальными правами, а в качестве имени и пароля у него выступает пустая строка.

Рис. 1.35. Настройка учетных записей

Последняя страница утилиты настройки MySQL-сервера MySQL Server Instance Configuration Wizard представлена на рис. 1.36.

Рис. 1.36. Последняя страница утилиты MySQL Server Instance Configuration Wizard

После нажатия кнопки **Execute** будет создан конфигурационный файл C:\MySQL\my.ini и запущен сервер MySQL.

ЗАМЕЧАНИЕ

Если при установке сервера MySQL у вас возникли затруднения, вы можете обратиться на форум по адресу **http://www.softtime.ru/forum/index.php?id_forum=3**.

1.12. Проверка работоспособности MySQL

После того как установка и конфигурирование MySQL завершены, необходимо убедиться в работоспособности сервера MySQL. Для этого следует открыть окно для работы с командной строкой, выбрав в системном меню **Пуск | Программы | MySQL | MySQL Server 5.1 | MySQL Command Line Client**. Открывшееся окно может выглядеть так, как это показано на рис. 1.37. На предложение ввести пароль нажмите клавишу <Enter>. После того как появилось приглашение `mysql>`, введите команду:

```
SELECT VERSION();
```

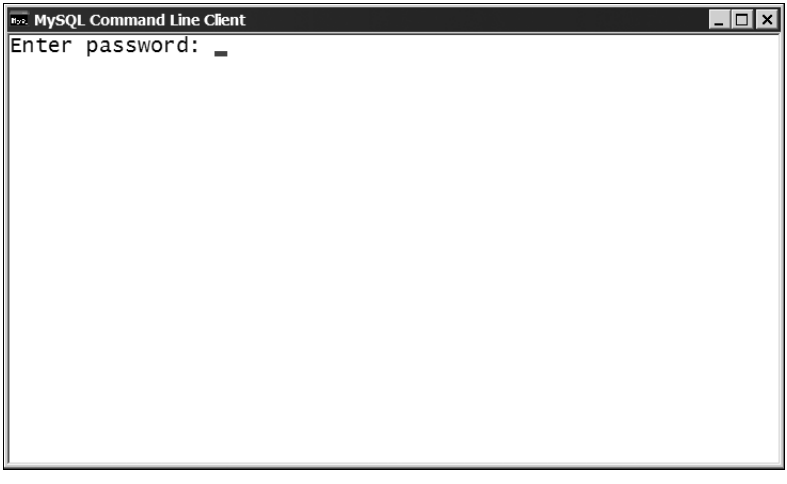


Рис. 1.37. Командная строка утилиты MySQL Command Line Client

В результате сервер должен показать текущую версию сервера, в нашем случае это `5.1.37-community` (рис. 1.38).

> ***ЗАМЕЧАНИЕ***
> Для выхода введите команду `EXIT`.

Если при установке MySQL кодировкой по умолчанию была выбрана cp1251, соответствующая русской кодировке Windows-1251, то утилита MySQL Command Line Client может не запуститься. Для исправления ситуации потребуется отредактировать конфигурационный файл my.ini, который можно обнаружить в корневом каталоге MySQL. Если при установке MySQL вы следовали инструкциям *разд. 1.10–1.11*, обнаружить конфигурационный файл можно по пути C:\MySQL\my.ini.

> ***ЗАМЕЧАНИЕ***
> Более подробно структура конфигурационного файла my.ini описывается в *разд. 1.14*.

Чтобы восстановить работоспособность утилиты MySQL Command Line Client, необходимо отредактировать файл my.ini таким образом, чтобы директива `default-character-set` присутствовала только в секции `[mysqld]` и отсутствовала в секциях `[mysql]` и `[client]`.

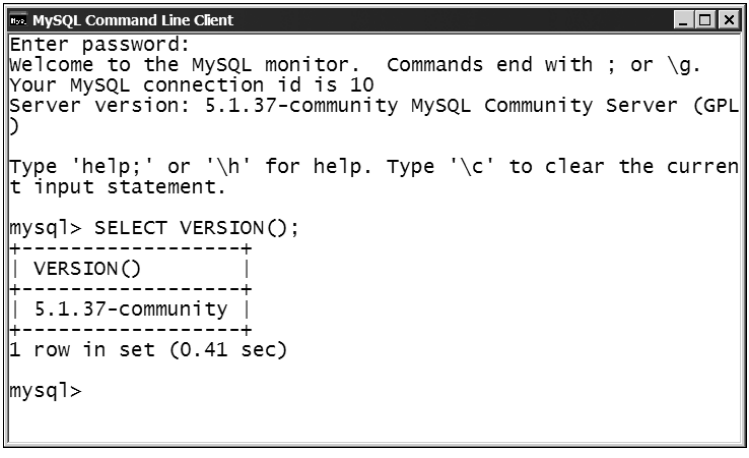


Рис. 1.38. Результат выполнения команды `SELECT VERSION()`

1.13. Управление запуском и остановкой MySQL

Запуск и остановка сервера MySQL, как и в случае Apache *(см. разд. 1.6)*, осуществляется через сервисы, консоль которых можно открыть, выполнив команду **Пуск | Настройка | Панель управления | Администрирование | Службы**. В появившемся окне консоли следует выбрать сервис с именем MySQL или тот, который был выбран на этапе конфигурирования (см. рис. 1.34). Контекстное меню позволяет осуществлять запуск, остановку и перезапуск сервиса.

В консоли управления сервисами необходимо найти сервис MySQL (рис. 1.39). Если поле **Состояние** данного сервиса пусто, то он не запущен. Для его запуска

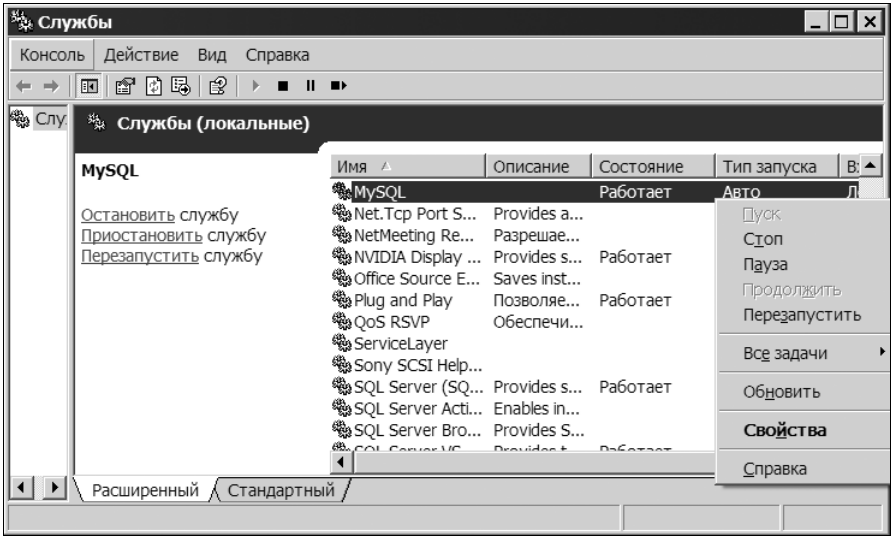


Рис. 1.39. Консоль управления сервисами

следует выбрать в контекстном меню пункт **Пуск**. Для остановки сервиса необходимо выбрать пункт **Стоп**. Обратите внимание на столбец **Тип запуска**: значение **Авто** сообщает Windows о необходимости запускать сервис при старте операционной системы, значение **Вручную** означает, что запуск выполняется пользователем через консоль управления сервисами. Можно изменить режим запуска сервиса, выбрав в контекстном меню сервиса пункт **Свойства**.

Замечание

Проверить, запущен MySQL-сервер или нет, можно также по наличию процесса mysqld.exe в Диспетчере задач.

1.14. Конфигурационный файл my.ini

Сервер MySQL имеет переменные состояния, которые определяют его настройки. Некоторые из переменных состояния могут быть изменены при помощи параметров запуска. При просмотре команды запуска сервера MySQL в свойствах сервиса или при помощи команды `ps -Af` можно заметить, что им передаются параметры запуска. *Параметром* называют ключевое слово, которое следует после имени процесса. Например, параметр `--defaults-file` в листинге 1.10 сообщает MySQL-серверу, что конфигурационный файл my.ini следует искать по адресу C:\mysql5\my.ini.

Замечание

Для того чтобы запустить MySQL-сервер в обход сервисов, необходимо воспользоваться параметром `--standalone`, чтобы процесс, обеспечивающий работу сервера, не был закрыт с закрытием консольного окна.

Листинг 1.10. Параметр запуска `--defaults-file`

```
C:\mysql\bin\mysqld.exe --defaults-file="C:\mysql5\my.ini" --standalone
```

Параметров сервера MySQL достаточно много, поэтому их может накапливаться изрядное количество. Просмотреть их полный список можно, передав серверу MySQL параметры `--verbose --help`. Некоторые параметры имеют короткие сокращения и начинаются с одного дефиса, а не с двух. Например, параметр `--user` имеет сокращение `-u`.

Параметры запуска можно не прописывать непосредственно в строке запуска сервера, а поместить в конфигурационный файл my.ini, из которого сервер прочитает их при старте. Если вы уже открывали файл my.ini на предыдущих этапах, то уже заметили, что по умолчанию файл содержит некоторое количество параметров.

Следует отметить, что в конфигурационном файле my.ini параметры указываются без предваряющих дефисов, т. к. каждый параметр располагается на отдельной строке. Если строка начинается с символа диеза `#` или точки с запятой `;`, то строка является комментарием. Параметры в конфигурационном файле называют *директивами*.

В операционной системе Windows конфигурационный файл может иметь как расширение ini (такой файл может располагаться в каталогах C:\Windows\ и в C:\mysql5\), так и расширение cnf (обычно располагается в корне диска C:). В UNIX-подобных операционных системах конфигурационный файл имеет, как правило, расширение cnf.

Конфигурационный файл влияет на работу не только сервера MySQL, но и вспомогательных утилит, таких как консольный клиент mysql, утилита создания SQL-дампов mysqldump и т. п., более того, один конфигурационный файл может управлять работой нескольких серверов MySQL. Поэтому содержимое конфигурационного файла разделено на секции, которые имеют вид `[имя_секции]`. Имя секции определяет утилиту или сервер, к которым будут относиться перечисленные далее директивы, до тех пор, пока не встретится новая секция или конец файла. В табл. 1.2 перечислены наиболее типичные секции.

Таблица 1.2. *Секции конфигурационного файла my.ini или my.cnf*

Имя секции	Описание
`[mysqld]`	Сервер MySQL
`[server]`	Сервер MySQL
`[mysqld-4.0]`	Сервер MySQL версии 4.0
`[mysqld-4.1]`	Сервер MySQL версии 4.1
`[mysqld-5.0]`	Сервер MySQL версии 5.0
`[mysqld-5.1]`	Сервер MySQL версии 5.1
`[mysqld_safe]`	Утилита запуска `mysqld_safe`
`[safe_mysqld]`	Утилита запуска `mysqld_safe`
`[mysql.server]`	Скрипт запуска `mysql.server`
`[mysqld_multi]`	Утилита запуска `mysqld_multi`
`[client]`	Любая клиентская утилита, обращающаяся к серверу
`[mysql]`	Консольный клиент `mysql`
`[mysqldump]`	Утилита создания SQL-дампов `mysqldump`
`[mysqlhotcopy]`	Утилита горячего копирования бинарных файлов базы данных

Наличие секции `[mysqld]` и специальных секций для разных версий обусловлено тем, что с каждой новой версией появляется все больше и больше параметров запуска, и если конфигурационный файл управляет несколькими серверами, то некоторые директивы будут одинаковыми для всех серверов, а другие уникальны для каждой из версий.

Точно такая же ситуация сложилась с секцией `[client]` и секциями для каждой из утилит. Дело в том, что все утилиты обладают сходными параметрами (например,

параметры соединения с серверами у всех одинаковые), и в то же время каждая из них имеет уникальные параметры, характерные только для ее функциональных возможностей.

В листинге 1.11 приводится пример конфигурационного файла my.ini, характерного для операционной системы Windows.

Листинг 1.11. Типичный конфигурационный файл my.ini

```
# Секция MySQL-сервера
[mysqld]
# Порт TCP/IP, который прослушивает MySQL-сервер, порт 3306 является
# стандартным, однако можно назначить другой незанятый порт
port=3306

# Путь установки MySQL-сервера, все остальные пути,
# если не указано другое, вычисляются относительно этого пути
basedir="C:/mysql5/"

# Путь к каталогу данных, в случае данного конфигурационного файла
# эту директиву можно не указывать, т. к. путь будет правильно
# вычислен относительно директивы basedir
datadir="C:/mysql5/Data/"

# В качестве кодировки по умолчанию для новых таблиц и баз данных
# будет выступать русская кодировка Windows-1251,
# в случае отсутствия этой директивы по умолчанию будет назначена
# кодировка latin1 (русский текст будет сортироваться неправильно)
default-character-set=cp1251

# Объем оперативной памяти, которая отводится на кэш ключей
key_buffer_size = 128M
# Максимальный размер запроса, который посылает клиент серверу
max_allowed_packet = 8M
# Объем оперативной памяти, которая отводится на кэш запросов
query_cache_size = 64M
# Максимальное количество одновременных подключений
max_connections = 200


# Секция для общих директив клиентов MySQL
[client]
# На одном IP-адресе может быть много серверов, чтобы различать, какие
# IP-пакеты предназначены тому или иному серверу, за каждым из них
# закрепляется порт — уникальный номер, который не может занимать другой
# сервер, для Apache это 80 порт (или 8080), для MySQL, как правило, 3306
port=3306
```

```
# Секция MySQL-сервера
[mysqld]
# Порт TCP/IP, который прослушивает MySQL-сервер, порт 3306 является
# стандартным, однако можно назначить любой другой незанятый порт
port=3306

# Путь установки MySQL-сервера, все остальные пути,
# если не указано другое, вычисляются относительно этого пути
basedir="C:/MySQL/"

# Путь к каталогу данных, где хранятся базы данных
datadir="C:/Documents and Settings/All Users/Application Data/MySQL/MySQL
Server 5.1/Data/"

# В качестве кодировки по умолчанию для новых таблиц и баз данных
# будет выступать русская кодировка Windows-1251,
# в случае отсутствия этой директивы по умолчанию будет назначена
# кодировка latin1 (русский текст будет сортироваться неправильно)
default-character-set=cp1251

# MySQL поддерживает большое количество таблиц с разной
# функциональностью, некоторые поддерживают уникальные возможности вроде
# полнотекстового поиска или внешних ключей, некоторые очень быстрые, так
# как располагаются в оперативной памяти, некоторые позволяют
# манипулировать частями в разных разделах и даже хостах. Следующая
# директива задает тип таблицы по умолчанию
default-storage-engine=MYISAM

# Объем оперативной памяти, которая отводится на кэш ключей
key_buffer_size = 128M

# Максимальный размер запроса, который посылает клиент серверу
max_allowed_packet = 8M

# Объем оперативной памяти, которая отводится на кэш запросов
query_cache_size = 64M

# Максимальное количество одновременных подключений
max_connections = 200
```

В конфигурационном файле my.ini, приведенном в листинге 1.11, представлена лишь небольшая часть директив. Полный список можно найти в официальной документации.

1.15. Связывание PHP и MySQL

После того как на компьютер установлены Web-сервер Apache, MySQL-сервер и PHP, можно приступить к связыванию PHP и MySQL. Для того чтобы в PHP-скрипте были доступны библиотеки, позволяющие отправлять запросы MySQL, необходимо подключить соответствующие расширения. В настоящий момент PHP поддерживает две библиотеки доступа к СУБД MySQL:

- php_mysql — традиционная процедурная библиотека;
- php_mysqli — объектно-ориентированная библиотека.

Для того чтобы подключить их, в конфигурационный файл php.ini необходимо добавить строки, представленные в листинге 1.12. Если эти строки уже присутствуют в файле и предваряются точкой с запятой, следует их раскомментировать.

Листинг 1.12. Подключение MySQL-расширений

```
...
extension=php_mysql.dll
extension=php_mysqli.dll
...
```

После внесения изменений необходимо перезагрузить Web-сервер Apache. Динамические библиотеки php_mysql.dll и php_mysqli.dll являются зависимыми от библиотеки libmysql.dll в корне каталога, в который установлен PHP. Если Web-сервер Apache не может перезапуститься после редактирования конфигурационного файла php.ini, одной из причин может быть невозможность обнаружить библиотеку libmysql.dll самостоятельно. В этом случае необходимо скопировать ее в один из системных каталогов, например, в C:\Windows\system32.

Для проверки правильности подключения расширения php_mysql можно воспользоваться скриптом из листинга 1.13. В случае успешной настройки связки PHP и MySQL, скрипт вернет текущую версию MySQL-сервера.

Листинг 1.13. Проверка работоспособности расширения php_mysql

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "test";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";

  // Устанавливаем соединение с базой данных
  $dbcnx = mysql_connect($dblocation, $dbuser, $dbpasswd);
```

```
  if (!$dbcnx) {
   exit( "<p>В настоящий момент сервер базы данных недоступен,
             поэтому корректное отображение страницы невозможно.</p>" );
  }
  // Выбираем базу данных
  if (! mysql_select_db($dbname, $dbcnx) ) {
    exit( "<p>В настоящий момент база данных недоступна,
              поэтому корректное отображение страницы невозможно.</p>" );
  }
  // Выполняем SQL-запрос
  $query = "SELECT VERSION()";
  $res = mysql_query($query);
  // Проверяем правильность выполнения запроса
  if(!$res) exit("Ошибка выполнения запроса — ".mysql_error());
  // Получаем результат выполнения запроса
  echo mysql_result($res, 0);
?>
```

Для проверки правильности подключения расширения php_mysqli можно воспользоваться скриптом из листинга 1.14. Как и в случае предыдущего скрипта, он также вернет текущую версию MySQL-сервера.

Листинг 1.14. Проверка работоспособности расширения php_mysqli

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "test";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";
  // Объявляем объект класса mysqli, который устанавливает
  // соединение с сервером
  $mysqli = new mysqli($dblocation, $dbuser, $dbpasswd, $dbname);
  if (mysqli_connect_errno())  exit("Ошибка установки соединения");

  // Выводим версию сервера
  echo $mysqli->server_info;

  // Закрываем соединение с сервером
  $mysqli->close();
?>
```

1.16. Настройка командной строки для mysql

Сервер MySQL выполняет команды, которые ему отправляет клиент. СУБД MySQL необходима Web-разработчикам, чтобы PHP-скрипт, выступая таким клиентом, имел возможность помещать информацию в таблицы базы данных, а затем извлекать ее по мере надобности. Существует большое количество платных и свободных клиентов для работы с MySQL. Самым распространенным клиентом является утилита mysql, с которой мы уже познакомились в *разд. 1.12* при тестировании работоспособности MySQL-сервера. Этот консольный клиент устанавливается вместе с дистрибутивом MySQL. Для его запуска необходимо выбрать следующий пункт меню: **Пуск | Программы | MySQL | MySQL Server 5.1 | MySQL Command Line Client**.

При запуске утилиты mysql с помощью MySQL Command Line Client нет возможности вводить какие-либо параметры. Кроме того, для работы с другими утилитами, входящими в поставку MySQL (например, утилитой mysqldump, позволяющей создавать SQL-дампы), необходим доступ к командной строке. Получить его можно, обратившись к следующему пункту: **Пуск | Программы | Стандартные | Командная строка**. В результате будет открыто окно, представленное на рис. 1.40.

ЗАМЕЧАНИЕ

В данной книге цвет окна и его размеры выбраны отличными от тех, что выставляются системой по умолчанию. Можно самостоятельно изменить цветовую схему командной строки и ее размеры в свойствах системного меню окна, щелкнув мышью пиктограмму окна вверху слева и выбрав в контекстном меню пункт **Свойства**.

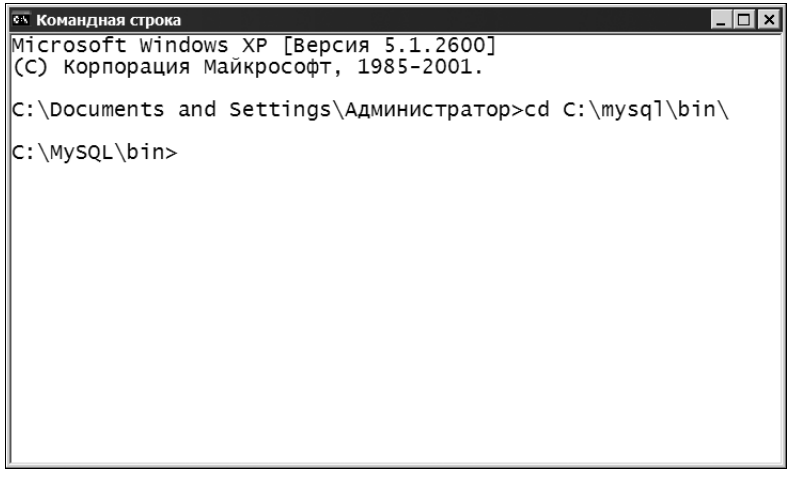


Рис. 1.40. Командная строка

После запуска командной строки необходимо перейти в подкаталог bin корневого каталога MySQL. Для этого следует набрать команду:

```
cd C:\mysql5\bin
```

В ответ будет выведено приглашение командной строки:

```
C:\mysql5\bin>
```

> ***Замечание***
>
> Если корневой каталог MySQL расположен на другом диске, необходимо перед выполнением команды `cd` сменить диск, например, `D:`. Кроме этого, создав ярлык для командной строки, в свойствах ярлыка можно выставить в качестве рабочего каталога путь к каталогу bin, для того чтобы не вводить всякий раз команду `cd`.

Теперь, находясь в каталоге bin, можно запускать расположенные в нем утилиты. Для этого достаточно набрать в командной строке имя утилиты и, если необходимо, параметры. Список всех файлов и подкаталогов текущего каталога можно отобразить, вызвав команду `dir`.

Каждый раз набирать команды перехода в каталог bin достаточно утомительно, особенно если сервер MySQL был установлен в каталог C:\Program Files. Имеется несколько возможностей для автоматизации загрузки консольного клиента mysql. Первая из них заключается в создании ярлыка командной строки на рабочем столе. Далее в контекстном меню ярлыка следует выбрать пункт **Свойства**, а в открывшемся диалоге вкладку **Ярлык**. В поле **Рабочая папка** следует ввести путь к каталогу bin (на рис. 1.40 это путь C:/mysql/bin), после чего нажать кнопку **OK** или **Применить**.

Теперь при вызове командной строки при помощи ярлыка в качестве текущего каталога изначально будет выставлен каталог C:\mysql\bin.

Второй путь автоматизации процесса запуска утилиты mysql заключается в том, чтобы прописать каталог C:\mysql\bin в переменной окружения `PATH`. Это позволит запускать утилиты из каталога bin из любого другого каталога компьютера.

> ***Замечание***
>
> В UNIX-подобных операционных системах все утилиты помещаются либо в каталог /bin, либо в каталог /usr/sbin, которые указаны в переменной окружения `PATH` по умолчанию. Если по какой-то причине утилиты MySQL расположены в других каталогах, следует либо указывать полный путь к ним, либо прописать каталог в переменной окружения `PATH`.

Переменная окружения — это параметр, который позволяет настроить поведение операционной системы: где искать исполняемые файлы, где хранить временные файлы, где расположен системный каталог и т. п. На рис. 1.41 в поле **Объект** используется конструкция %SystemRoot% — это один из примеров переменной окружения, которая хранит путь к каталогу Windows. Отдельные программы могут создавать собственные переменные окружения, однако некоторые из переменных окружения являются стандартными. К стандартным переменным окружения относится переменная `PATH`, которая сообщает операционной системе, в каких каталогах следует искать исполняемые файлы.

В операционной системе Windows для доступа к переменным окружения следует открыть диалог **Свойства системы**, выбрав в контекстом меню значка **Мой компьютер** пункт **Свойства**, или посредством команд: **Пуск | Настройка | Панель управления | Система**. В результате будет открыто диалоговое окно, изображенное на рис. 1.42.

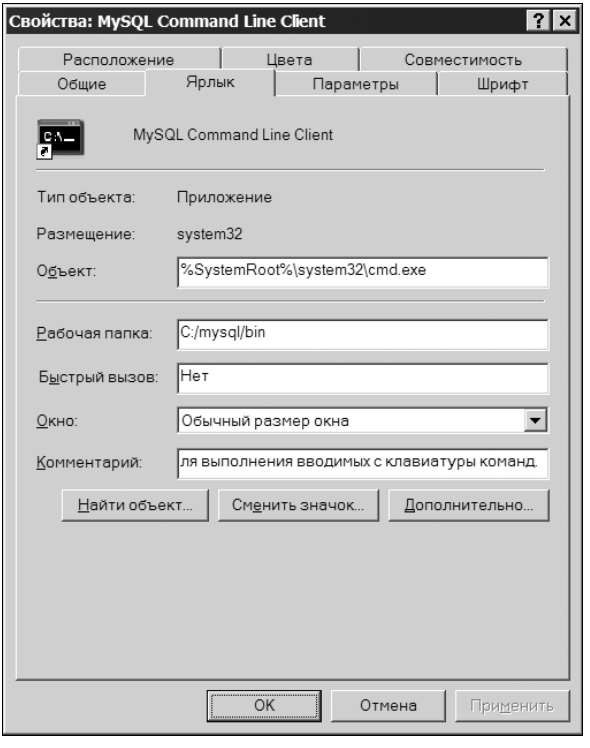


Рис. 1.41. Свойства ярлыка для командной строки

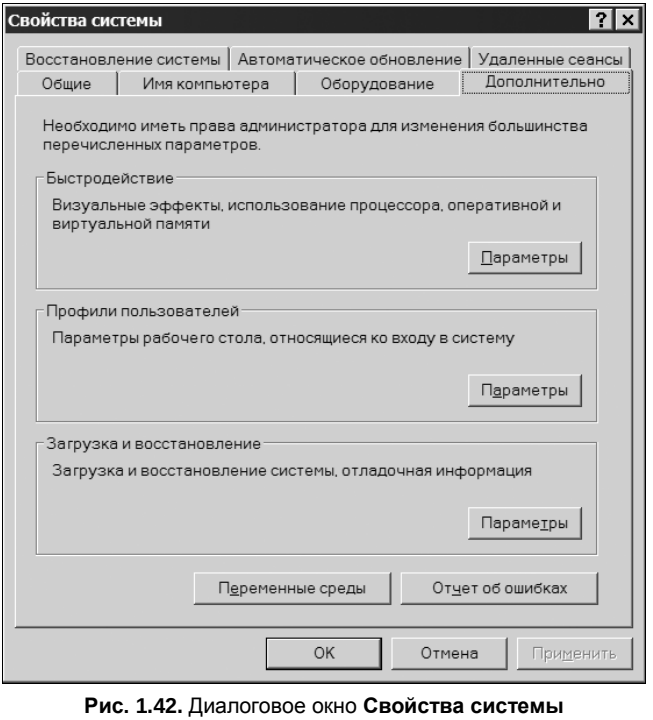


Рис. 1.42. Диалоговое окно **Свойства системы**

На вкладке **Дополнительно** необходимо нажать кнопку **Переменные среды**, что приведет к открытию диалогового окна, представленного на рис. 1.43.

В разделе **Системные переменные** следует отыскать переменную окружения `PATH` и дополнить ее путем к каталогу bin СУБД MySQL. Отдельные пути в значении переменной `PATH` разделяются точкой с запятой (в конце всей строки точка с запятой не требуется). Новое значение переменной окружения `PATH` вступает в силу после перезагрузки компьютера.

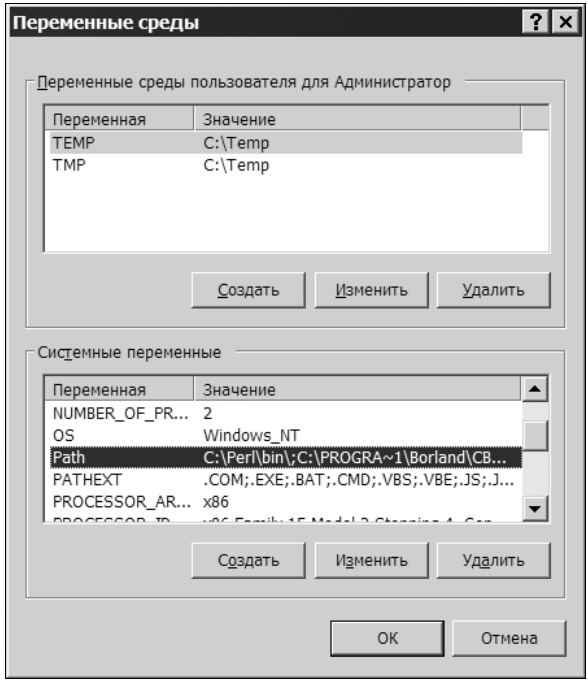


Рис. 1.43. Диалоговое окно **Переменные среды**

1.17. Поддержка русского языка

Еще одним важным параметром командной строки является кодировка. По умолчанию набранный в консоли русский текст будет помещен в базу данных MySQL в кодировке cp866 (DOS), что может не входить в ваши планы. Для смены кодировки в консоли используется команда `chcp`, запуск которой без параметров сообщает текущую кодировку, передача команде в качестве параметра номера кодировки приводит к смене кодировки консоли. Например, для того чтобы установить кодировку Windows-1251, следует выполнить команду:

```
chcp 1251
```

Точечные шрифты в консоли не поддерживают русский текст и выводят его в искаженном виде. Поэтому следует сменить в свойствах консоли шрифт на Lucida Console.

Кроме того, следует учитывать, что MySQL-сервер по умолчанию ожидает данные в кодировке latin1, что приводит к искажению вводимых данных. Чтобы русский текст, набираемый в консоли, помещался в базу данных без искажений, сразу после установки соединения с MySQL-сервером необходимо выполнить SQL-запрос, представленный в листинге 1.15.

Листинг 1.15. Настройка кодировки соединения

```
SET NAMES cp1251;
```

Настройка кодировки соединения должна осуществляться не только в консольном режиме, но и в любом другом клиенте, в том числе и в PHP-скрипте. В листинге 1.16 представлен типичный скрипт для установки соединения с СУБД MySQL.

Листинг 1.16. Установка соединения с СУБД MySQL

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "test";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";

  // Устанавливаем соединение с базой данных
  $dbcnx = mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx) {
   exit( "<p>В настоящий момент сервер базы данных недоступен,
             поэтому корректное отображение страницы невозможно.</p>" );
  }
  // Выбираем базу данных
  if (! mysql_select_db($dbname, $dbcnx) ) {
    exit( "<p>В настоящий момент база данных недоступна,
              поэтому корректное отображение страницы невозможно.</p>" );
  }
  // Настраиваем кодировку соединения
  @mysql_query("SET NAMES cp1251");
?>
```

ГЛАВА 2

Хитрости конфигурирования среды

Если в первой главе рассказывалось о том, как создать удобную среду программирования Web-приложений, то в данной главе мы еще больше погрузимся в хитрости конфигурирования и использования возможностей PHP, Web-сервера Apache и СУБД MySQL.

2.1. PHP

2.1.1. Структура конфигурационного файла php.ini

Конфигурационный файл php.ini состоит из секций и директив. Секции заключаются в квадратные скобки, например, `[PHP]`, после которых следуют директивы, имеющие такой формат:

```
directive = value
```

Здесь `directive` — это название директивы, а `value` — ее значение. Все строки, в начале которых располагается точка с запятой (`;`), считаются комментариями и игнорируются.

Допускается не указывать значение `value`. В этом случае директива инициализируется пустой строкой. Этого же результата можно добиться, присвоив ей значение `none`.

Файл php.ini начинается с директивы `[PHP]`, которая позволяет настроить параметры ядра, после чего следуют директивы расширений: внешние библиотеки (такие как `[MySQL]` или `[PostgresSQL]`) и расширения, включенные в состав ядра (например, `[Date]` или `[Session]`, настраивающие порядок работы с датой и сессиями).

Если PHP подключен к Web-серверу в качестве модуля, изменения в конфигурационном файле php.ini вступают в силу только после перезагрузки Web-сервера. В случае если PHP подключен как внешнее CGI-приложение, изменения вступают в силу немедленно.

По умолчанию интерпретатор PHP последовательно ищет конфигурационный файл в нескольких местах:

- по пути, указанному в директиве PHPIniDir конфигурационного файла httpd.conf Web-сервера Apache (директива действует, начиная с Apache 2.2);
- по пути, указанному в переменной окружения PHPRC (начиная с версии PHP 5.2);
- в текущем каталоге (если скрипт выполняется под управлением Web-сервера);
- в каталоге C:\Windows\ в случае Windows и в каталоге компиляции в случае любой другой операционной системы.

Если PHP-скрипт запускается вне среды Web-сервера, указать путь к конфигурационному файлу php.ini можно при помощи параметра -c, например:

```
php.exe -c C:/PHP/php.ini D:/scripts/file.php
```

В следующих разделах будут рассмотрены директивы ядра [PHP].

2.1.2. Параметры языка *PHP*

В табл. 2.1 представлены наиболее часто используемые директивы группы управления параметрами языка. В скобках указывается значение, которое может принимать директива.

Таблица 2.1. *Директивы управления параметрами языка*

Директива	Описание
engine	Включает (On) или выключает (Off) выполнение PHP-скриптов под управлением Web-сервера Apache
short_open_tag	Включает (On) или выключает (Off) возможность использования короткого тега <? вместо <?php
asp_tags	Включает (On) или выключает (Off) возможность использования ASP-тегов <% и %> вместо <?php и ?>. Директива исключена в PHP 6
precision	Указывает количество знаков, которое отводится под дробное число в случае, если вывод не форматируется при помощи специальных функций, таких как printf(), sprintf()
output_buffering	Включает (On) или выключает (Off) автоматическую буферизацию вывода. В качестве значения директива может принимать число байтов, по достижению которых буфер автоматически очищается, а его содержимое отправляется клиенту

Директива engine по умолчанию принимает значение On, установка данной директивы в значение Off приводит к тому, что PHP-скрипты перестают интерпретироваться под управлением Web-сервера. Попытка обратиться к PHP-скрипту вместо выдачи HTML-результата приведет к выводу окна, предлагающего сохранить файл (рис. 2.1).

Директивы short_open_tag и asp_tags определяют доступность различных тегов, обрамляющих PHP-скрипт. Традиционно директива short_open_tag, позволяющая использовать короткий вариант <?, включена (On). Директиву имеет смысл отключить, если PHP-скрипты используются совместно с XML-файлами, которые используют теги <?xml и ?>.

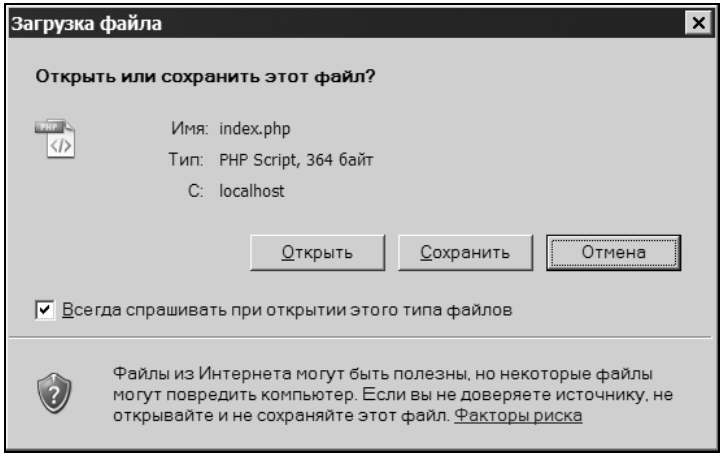


Рис. 2.1. Отключение интерпретации PHP-файла

Директива `asp_tags`, позволяющая использовать теги языка ASP `<%` и `%>`, отключена. Начиная с версии PHP 6, возможность использования ASP-тегов и сама директива `asp_tags` исключаются из языка.

Директива `precision` позволяет задать количество знаков, которое отводится под вывод вещественного числа. В листинге 2.1 дан скрипт, выводящий два вещественных числа в разных форматах.

> ***Замечание***
>
> Директива `precision` не влияет на точность вычисления, она определяет только количество символов после запятой в случае неформатного вывода вещественного числа.

Листинг 2.1. Вывод вещественных чисел

```
<?php
  echo 10.23456;
  echo "<br />";
  echo 10.23456E+20;
?>
```

Если значение директивы `precision` равно 4, скрипт выведет следующую пару чисел:

```
10.23
1.023E+21
```

Если значение директивы `precision` превышает количество цифр в числе, они выводятся без изменений:

```
10.23456
1.023456E+21
```

Директива `output_buffering`, отключенная по умолчанию, позволяет включить буферизацию вывода. По умолчанию PHP-интерпретатор отправляет клиенту данные

сразу после их обработки, такой вывод может порождать некоторые проблемы. С одной стороны, данные отправляются небольшими порциями, что может приводить к замедлению вывода, с другой стороны, это жестко регламентирует отправку HTTP-заголовков, которые должны отправляться всегда перед данными. Рассмотрим работу директивы `output_buffering` на примере скрипта из листинга 2.2.

Листинг 2.2. Отправка HTTP-заголовков после вывода данных

```
<?php
  echo "hello world!";
  session_start();
?>
```

Здесь осуществляется попытка отправки данных перед HTTP-заголовком (напомним, что SID-сессии передаются через cookies, что устанавливаются при помощи HTTP-заголовка `Set-Cookie`). При отключенной буферизации попытка выполнения скрипта приводит к ошибке (рис. 2.2).

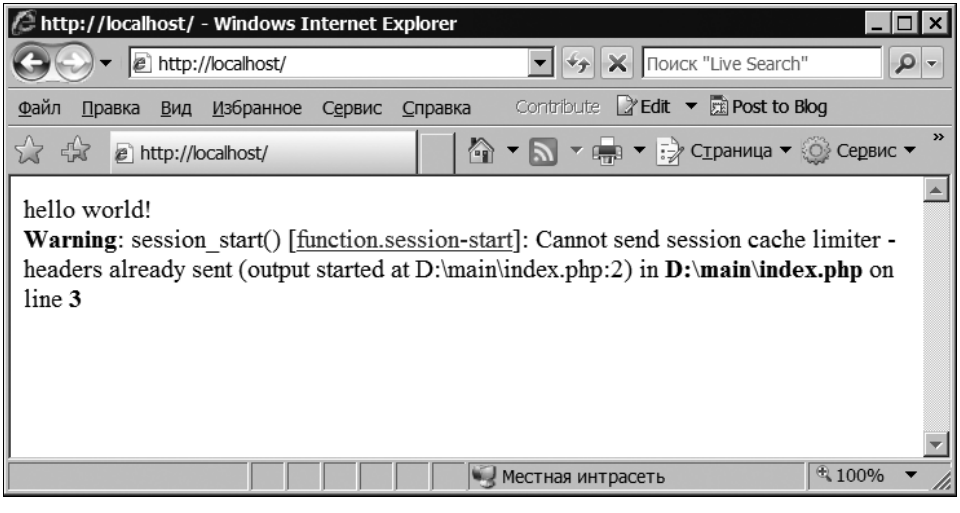


Рис. 2.2. HTTP-заголовки должны отправляться перед данными

Включив буферизацию (`output_buffering = On`) в конфигурационном файле php.ini, можно заставить помещать все данные в буфер, тогда интерпретатор получит возможность сначала отправить HTTP-заголовки и лишь потом данные, независимо от порядка их вывода в скрипте (рис. 2.3).

Вместо значения `On` директива `output_buffering` может принимать максимальное количество байтов в буфере, после чего буфер автоматически очищается, а его содержимое отправляется клиенту. Строка "hello world!" в однобайтовой кодировке занимает 12 байтов, поэтому при значении `output_buffering` больше 12 скрипт из листинга 2.2 выполняется без ошибок, при значениях директивы меньше 12 — с ошибкой.

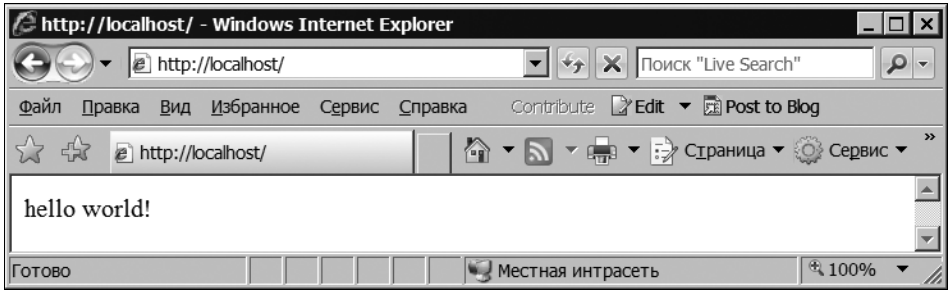


Рис. 2.3. Буферизация вывода позволяет использовать HTTP-заголовки в произвольном месте

2.1.3. Директивы безопасности

Один Web-сервер может поддерживать множество сайтов. Такое соседство порождает потенциальную опасность: скрипты одного сайта могут намеренно или случайно обращаться к скриптам соседнего сайта. Задачу разграничения прав традиционно решают средствами операционной системы и Web-сервера. Однако PHP имеет некоторые директивы, позволяющие упростить процесс такой настройки.

Безопасный режим для ограничения доступа PHP к файлам использует UNIX-права доступа. UNIX — многопользовательская операционная система, в которой может быть несколько пользователей, которые объединяются в группы (любой пользователь входит минимум в одну группу). Все файлы и каталоги снабжаются двумя атрибутами: владелец файла и группа пользователей, которые могут работать с файлом. Поэтому права доступа для каждого файла или каталога выставляются для трех категорий пользователей:

- владелец файла;
- группа владельцев;
- остальные пользователи.

В рамках каждой из трех групп существуют три вида разрешений:

- `r` — право чтения данного файла;
- `w` — право записи/изменения данного файла;
- `x` — право выполнения данного файла, если он является сценарием или программой (для каталога это право его просмотра).

Права доступа могут выглядеть так: `rwxr--r--`. Такая запись означает, что владелец файла может читать файл, записывать и выполнять, а все остальные пользователи только читать. UNIX-права доступа можно задавать восьмеричным числом. При этом праву чтения соответствует цифра 4, праву записи — 2, а исполнению — 1. Общие права для групп задаются суммой этих чисел: так 6 (4 + 2) обеспечивает возможность чтения и записи, а 7 (4 + 2 + 1) предоставляет полный доступ к файлу или каталогу. Тогда для каталога восьмеричное число 0755 означает, что владелец каталога имеет полный доступ (`rwx`), а все остальные имеют право читать файлы в нем и просматривать содержимое каталога (`r-x`).

Безопасный режим сначала проверяет, совпадают ли владельцы скрипта и файла или каталога (которыми скрипт оперирует). При помощи директивы `safe_mode_gid` можно смягчить это ограничение: присвоив данной директиве значение `On`, можно потребовать проверять совпадение лишь группы владельцев.

Следует отметить, что ограничение на чтение и модификацию файлов распространяется не только на файловые функции `file_get_contents()`, `put_get_contents()`, `fopen()`, `fwrite()`, `fread()`, но и на директивы `include`, `include_once`, `require` и `require_once`. Такое поведение может быть не очень удобным, особенно если общие для множества проектов включаемые файлы хранятся в отдельном каталоге (как библиотека PEAR с готовыми PHP-решениями). Для того чтобы снять ограничение на включаемые файлы, используется директива `safe_mode_include_dir`: на каталоги, указанные в этой директиве, не распространяется проверка владельца пользователя и группы владельцев.

PHP-скрипт может не только самостоятельно обратиться к чужому файлу, но и попытаться изменить его при помощи средств операционной системы, используя функции `system()`, `exec()`, `shell_exec()` (рис. 2.4).

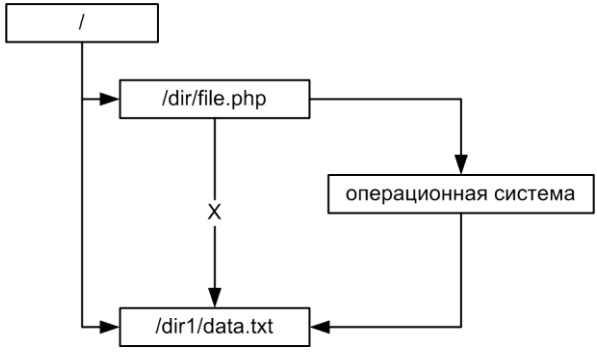


Рис. 2.4. Попытка обращения к чужому файлу через команды операционной системы

Безопасный режим предотвращает модификацию чужих файлов, в том числе и при помощи функций, способных выполнять команды операционной системы, что может создавать определенные неудобства. При помощи директивы `safe_mode_exec_dir` можно указать каталог с исполняемыми файлами, на которые защищенный режим не будет распространяться.

Еще одним важным ресурсом являются переменные окружения, при помощи которых операционная система и среда выполнения (в нашем случае Web-сервер) могут обмениваться с исполняемыми файлами информацией. Операционная система или Web-сервер могут заполнять переменные окружения, а исполняемый модуль (например, PHP) может читать и использовать эту информацию. В качестве примера можно привести обработку GET-запроса. Получив GET-параметры, Web-сервер создает переменную окружения `QUERY_STRING` и помещает в нее GET-параметры. Теперь PHP получает возможность прочитать значение из переменной окружения и заполнить суперглобальные массивы `$_SERVER` и `$_GET`. Переменных окружения мо-

жет быть достаточно много, сам PHP-скрипт, как будет показано далее в этой главе, также имеет возможность изменять переменные окружения (рис. 2.5).

ЗАМЕЧАНИЕ

Запрет на изменение переменных окружения необходим для таких важных параметров, как объем памяти и время выполнения, выделяемые на один экземпляр скрипта.

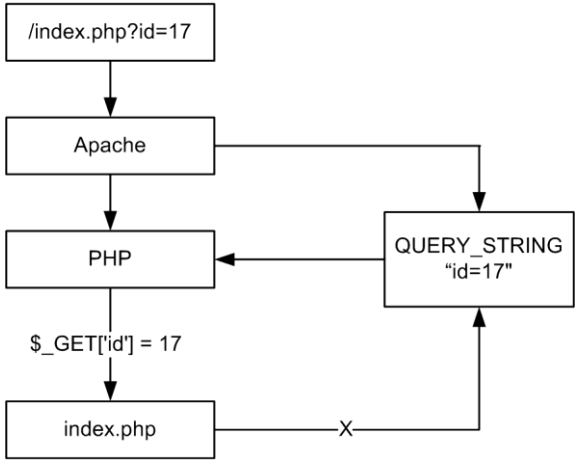


Рис. 2.5. Взаимодействие PHP и Apache при помощи переменных окружения

В безопасном режиме разрешается изменять лишь те переменные окружения, которые начинаются с префикса, указанного в директиве `safe_mode_allowed_env_vars`. По умолчанию директива разрешает изменять переменные окружения, начинающиеся с префикса `PHP_`:

```
safe_mode_allowed_env_vars = PHP_
```

При необходимости директивой `safe_mode_protected_env_vars` можно запретить изменять любые переменные окружения, даже разрешенные директивой `safe_mode_allowed_env_vars` (при этом переменные отделяются друг от друга двоеточием).

В табл. 2.2 представлен полный список директив, позволяющих настраивать защищенный режим PHP.

Таблица 2.2. *Директивы защищенного режима*

Директива	Описание
`safe_mode`	Включает (`On`) или выключает (`Off`) защищенный режим, ограничивающий доступ PHP к чужим файлам и некоторым функциям
`safe_mode_gid`	Вариант защищенного режима: вместо проверки пользователя файла проверяется группа-пользователь
`safe_mode_include_dir`	На каталоги, указанные в данной директиве, не распространяется проверка пользователя

***Таблица 2.2** (окончание)*

Директива	Описание
safe_mode_exec_dir	Исполняемые файлы в этом каталоге разрешено выполнять при помощи функций system(), exec() и т. п. В остальных каталогах исполняемые файлы в защищенном режиме будут игнорироваться
safe_mode_allowed_env_vars	Префикс переменных окружения, которые могут быть изменены из PHP-скрипта в защищенном режиме. Если директива принимает пустую строку, разрешается изменение любых переменных окружения
safe_mode_protected_env_vars	Запрет модификации в защищенном режиме переменных окружения, даже если она разрешена директивой safe_mode_allowed_env_vars

В качестве демонстрации защищенного режима рассмотрим запрет на выполнение команд операционной системы. Попробуем прочитать содержимое текущего каталога при помощи команды dir (для Windows) или ls -l (для Linux). Проще всего воспользоваться функцией shell_exec(), которая выполняет команду и возвращает результат вывода команды в стандартный поток (листинг 2.3).

Листинг 2.3. Выполнение команды операционной системы

```
<?php
  $text = shell_exec("dir");
  echo "<pre>";
  echo htmlspecialchars(convert_cyr_string($text, 'd', 'w'));
  echo "</pre>";
?>
```

Формат вывода команды предназначен для командной строки, поэтому просто вывода echo shell_exec("dir") недостаточно. В браузере переводы строк рассматриваются как обычные пробельные символы, поэтому их следует либо заменить на тег
 , либо обрамить вывод в теги <pre> и </pre>. Кроме того, результат выполнения функции возвращается в кодировке cp866 (DOS). Поэтому кодировка вывода меняется при помощи функции convert_cyr_string(). В выводе команды dir для обозначения каталога используется последовательность <DIR>, которая воспринимается браузером как тег и в конечном итоге приводит к ступенчатому отображению результата. Поэтому результат пропускается через функцию htmlspecialchars() для того, чтобы предотвратить интерпретацию последовательности <DIR>. На рис. 2.6 представлен результат выполнения скрипта в штатном режиме.

Если в конфигурационном файле php.ini включить директиву safe_mode, вместо содержимого текущего каталога скрипт из листинга 2.3 выдаст предупреждение, сообщающее о невозможности выполнения функции shell_exec() в безопасном режиме (рис. 2.7).

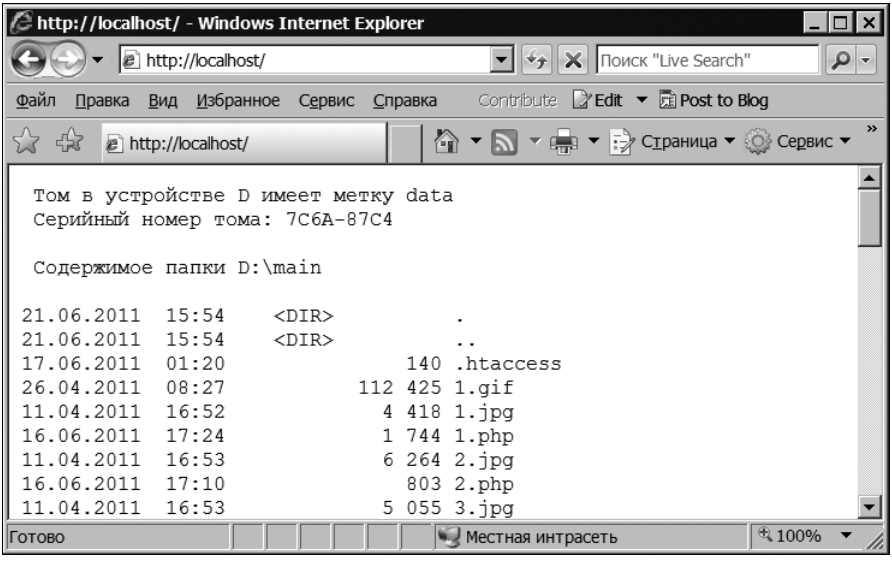


Рис. 2.6. Результат выполнения команды `dir`

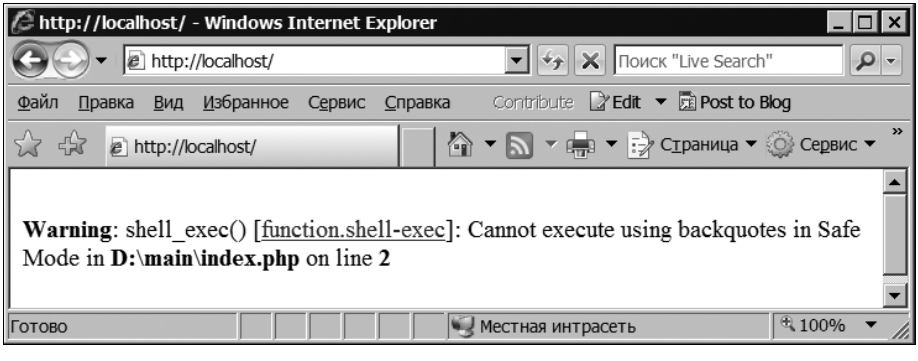


Рис. 2.7. Выполнение команд операционной системы запрещено в защищенном режиме

ЗАМЕЧАНИЕ

В защищенном режиме использование обратных кавычек также запрещено.

PHP предоставляет администраторам ряд директив, которые работают независимо от того, включен защищенный режим или нет (табл. 2.3).

Таблица 2.3. *Директивы безопасности*

Директива	Описание
`open_basedir`	Позволяет задать список каталогов, в которых разрешено выполнение PHP-скриптов и доступ к файлам
`disable_functions`	Позволяет задать список функций, запрещенных к выполнению
`disable_classes`	Позволяет задать список классов, запрещенных к использованию

Директива open_basedir позволяет задать список каталогов, в которых разрешено выполнение PHP-скриптов и обращение к файлам при помощи файловых функций (разрешение автоматически распространяется и на подчиненные каталоги). Указанные в директиве каталоги выступают в качестве префикса. Например,

```
open_basedir = "D:/main/pro"
```

Путь "D:/main/pro" разрешает доступ как в каталог D:/main/pro/, так и в D:/main/projects/ и D:/main/progress/. Для того чтобы ограничить доступ только каталогом pro, следует в конце строки явно указать слэш "D:/main/pro/".

> ***Замечание***
>
> Используя директиву open_basedir совместно с виртуальными хостами Web-сервера, можно добиться разграничения PHP по виртуальным хостам таким образом, чтобы скрипты одного сайта не могли получить доступ к файлам другого сайта *(см. разд. 2.1.15)*.

Директива disable_functions позволяет задать список функций, запрещенных к использованию. Например, следующее значение запретит выполнение функций set_time_limit() и putenv():

```
disable_functions  = "set_time_limit,putenv"
```

2.1.4. Настройка подсветки PHP-кода

В распоряжение разработчиков PHP предоставляет функции, обеспечивающие подсветку PHP-кода (табл. 2.4).

Таблица 2.4. *Подсветка PHP-кода*

Функция	Описание
highlight_string($code [, $return])	Функция подсвечивает синтаксис содержимого PHP-скрипта $code и выводит в окно браузера. Если необязательный параметр $return принимает значение true, вместо непосредственного вывода функция возвращает строку с результатом
highlight_file($filename [, $return])	Функция открывает PHP-файл $filename, подсвечивает его содержимое и выводит в окно браузера. Если необязательный параметр $return принимает значение true, вместо непосредственного вывода функция возвращает строку с результатом

Пусть имеется файл test.php, который необходимо вывести с подсветкой синтаксиса (листинг 2.4).

Листинг 2.4. Файл test.php

```
<?php
  if (!$flag)
  {
    // Пишем любой код
    echo "Hello";
```

```
    $var = 1;
  }
  else echo "Ошибка";
?>
```

В листинге 2.5 приводится пример использования функции `highlight_file()`, которая извлекает содержимое файла test.php и выводит его в окно браузера.

Листинг 2.5. Использование функции `highlight_file()`

```
<?php
  highlight_file("test.php");
?>
```

На рис. 2.8 демонстрируется результат выполнения скрипта из листинга 2.5.

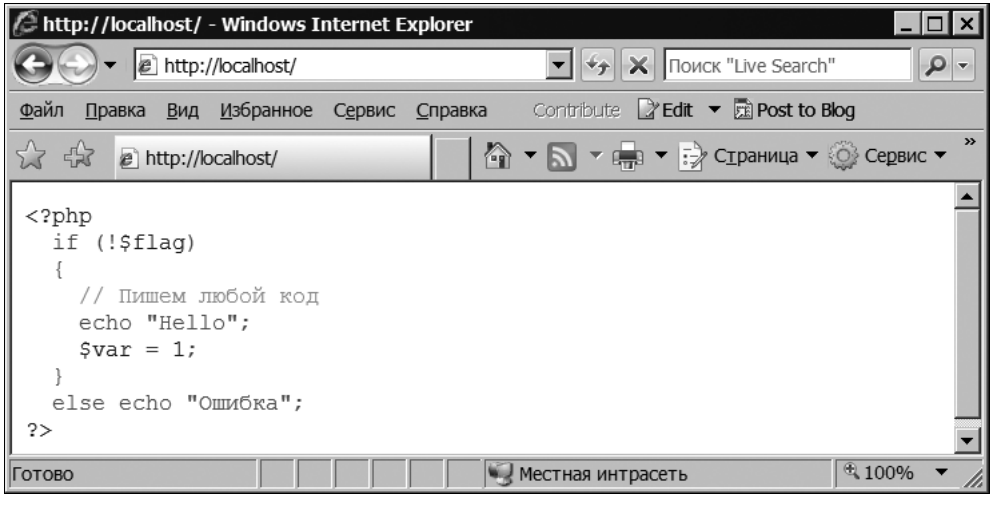


Рис. 2.8. Функция `highlight_file()` подсвечивает синтаксис PHP-скрипта

Если вывод результата функции непосредственно в месте ее вызова нецелесообразен (или необходима предварительная обработка получившегося HTML-кода), можно передать второму необязательному параметру функции значение `true`, в этом случае результат можно сохранить в строке (листинг 2.6).

Листинг 2.6. Использование второго параметра функции `highlight_file()`

```
<?php
  $code = highlight_file("test.php", true);
  // Промежуточный код
  ...
  // Вывод результата
  echo $code;
?>
```

Единственное неудобство в использовании функции заключается в том, что цвета программных элементов жестко заданы: комментарии — оранжевый, переменные и числа — синий, строки — красный, ключевые слова — зеленый и т. д. Изменить цвета по умолчанию можно в конфигурационном файле php.ini при помощи директив, представленных в табл. 2.5.

ЗАМЕЧАНИЕ

Директива `highlight.bg` исключена, начиная с PHP 6.0.

Таблица 2.5. *Директивы подсветки PHP-кода*

Директива	Описание
`highlight.string`	Цвет строк, по умолчанию это красный цвет (#DD0000)
`highlight.comment`	Цвет комментариев, по умолчанию оранжевый цвет (#FF9900)
`highlight.keyword`	Цвет ключевых слов, по умолчанию это зеленый цвет (#007700)
`highlight.bg`	Цвет фона, по умолчанию это белый цвет (#FFFFFF)
`highlight.default`	Цвет остальных элементов PHP-кода, по умолчанию это синий цвет (#0000BB)
`highlight.html`	Цвет HTML-кода, по умолчанию это черный цвет (#000000)

Цвет задается в HTML-формате, т. к. подсвечиваемые элементы обрамляются `span`-тегами вида

```
<span style="color: #0000BB">$var = 1;</span>
```

Может использоваться шестнадцатеричная нотация #RRBBGG, задающая красный (R), синий (B) и зеленый (G) цвета шестнадцатеричным числом (от 0 до 255). Кроме этого, могут использоваться предопределенные цвета HTML: `red`, `blue` и т. п.

2.1.5. Кэш файловой системы

Некоторые файловые функции кэшируют результаты своей работы. Директивы, представленные в табл. 2.6, позволяют управлять размером этого кэша и частотой его обновления.

ЗАМЕЧАНИЕ

Функция `clearstatcache()` позволяет очистить кэш досрочно.

Таблица 2.6. *Директивы управления кэшем файловой системы*

Директива	Описание
`realpath_cache_size`	Определяет размер кэша, по умолчанию 16 Кбайт
`realpath_cache_ttl`	Определяет частоту обновления кэша, по умолчанию 120 с

2.1.6. Взаимодействие с клиентом

Директивы данной группы определяют параметры взаимодействия с клиентом (табл. 2.7).

Таблица 2.7. *Директивы взаимодействия с клиентом*

Директива	Описание
`ignore_user_abort`	Если директива включена (`On`), скрипт не завершает работу после разрыва соединения с клиентом
`expose_php`	Разрешает (`On`) или запрещает (`Off`) сообщать клиенту, что сайт использует PHP при помощи HTTP-заголовка

2.1.7. Ограничение ресурсов

Сервер может поддерживать множество сайтов, которые в свою очередь могут обслуживать множество клиентов. Такое положение дел вынуждает ограничивать скрипты как по количеству используемой памяти, так и по времени выполнения.

ЗАМЕЧАНИЕ

Директивы в табл. 2.8 вводят ограничение на процессорное время. Это означает, что время ожидания ответа базы данных или ответа удаленного сервера не учитывается.

Таблица 2.8. *Директивы ограничения ресурсов*

Директива	Описание
`max_execution_time`	Определяет максимальное количество процессорного времени (в секундах), отводимого на выполнение скрипта. Если директива принимает значение 0, скрипт выполняется бессрочно. Значение по умолчанию 30 с
`max_input_time`	Максимальное количество процессорного времени (в секундах), отводимое на разбор GET-, POST-данных и загруженных файлов. Если директива принимает значение –1, по умолчанию принимается значение 60 с
`memory_limit`	Максимальное количество памяти, выделяемое под один экземпляр скрипта

В подавляющем большинстве случаев 30 с более чем достаточно для выполнения скрипта, задаваемого директивой `max_execution_time`. Если все же этот лимит превышается, работа скрипта останавливается, а в окно браузера выводится предупреждение (рис. 2.9).

Схожим образом действует ограничение на количество выделяемой памяти, задаваемое директивой `memory_limit`. Данное ограничение предназначено для того, чтобы неэффективные скрипты не могли в короткое время исчерпать память сервера.

ЗАМЕЧАНИЕ

Чаще всего с ограничениями по памяти разработчики сталкиваются при попытке прочитать гигантского размера файлы в переменную скрипта или при работе с объемными изображениями средствами библиотеки GDLib.

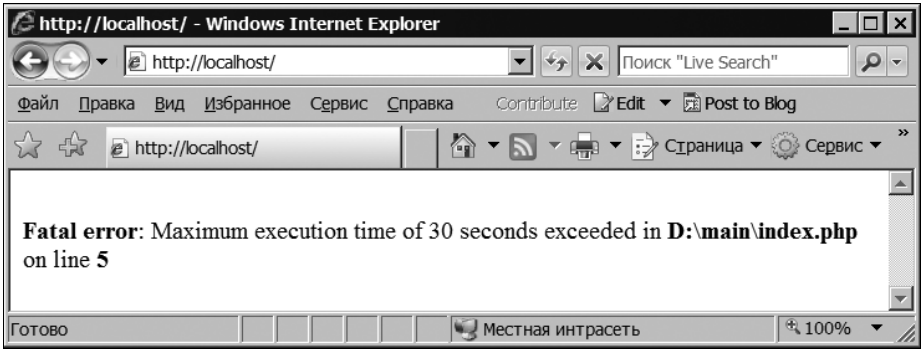


Рис. 2.9. Превышение ограничения процессорного времени

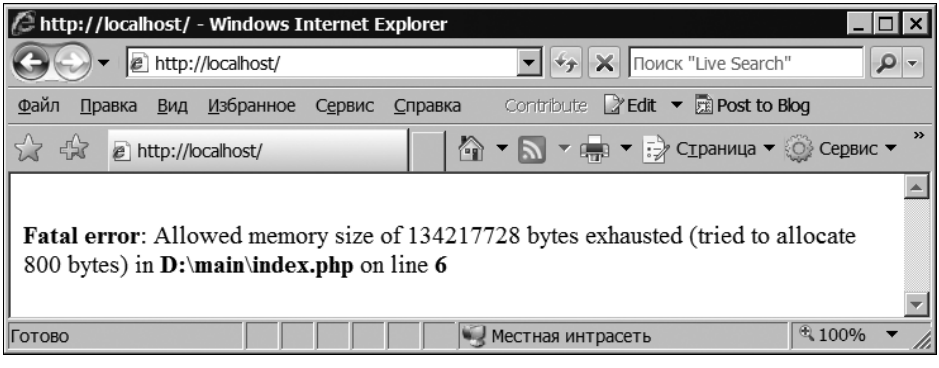


Рис. 2.10. Достижение предела выделенной оперативной памяти

На рис. 2.10 приводится типичный результат работы скрипта, исчерпавшего весь отведенный ему лимит оперативной памяти.

2.1.8. Обработка ошибок

Директивы группы обработки ошибок позволяют настроить уровень обработки ошибок и методы сообщения об ошибках интерпретатором PHP. В табл. 2.9 приводится список доступных директив.

Таблица 2.9. *Директивы обработки ошибок*

Директива	Описание
`error_reporting`	Устанавливает уровень обработки ошибок
`display_errors`	Включает (`On`) или отключает (`Off`) отображение ошибок в окно браузера
`display_startup_errors`	Включает (`On`) или отключает (`Off`) отображение ошибок запуска. Этот тип ошибок управляется отдельно от всех остальных типов ошибок
`log_errors`	Включает (`On`) или отключает (`Off`) сохранение ошибок в журнал, местоположение которого определяется директивой `error_log`
`log_errors_max_len`	Определяет максимальное количество байтов, которое сохраняется в журнал ошибок. Значение 0 снимает ограничение на размер сохраняемого сообщения

***Таблица 2.9** (окончание)*

Директива	Описание
`ignore_repeated_errors`	Включает (`On`) или отключает (`Off`) игнорирование повторяющихся ошибок (произошедших в том же самом файле, на той же самой строке)
`track_errors`	Если директива включена (`On`), сообщение о последней ошибке сохраняется в переменную `$php_errormsg`
`error_log`	Определяет местоположение журнала ошибок. Это может быть обычный текстовый файл или служба регистрации событий. Если директива принимает значение `syslog`, сообщения об ошибках сохраняются в журналах службы syslog в UNIX и журнале событий в Windows

Зачастую рекомендуется сохранять сообщения об ошибках не только на рабочих серверах, но и во время разработки. Последнее особенно полезно, если сообщение об ошибке скрывается дизайном или теряется в результате редиректа. Для того чтобы включить сохранение сообщений в журнал, необходимо произвести настройки конфигурационного файла php.ini в соответствии с листингом 2.7.

Листинг 2.7. Настройка директив обработки ошибок

```
; Реагируем на все сообщения об ошибках за исключением замечаний
error_reporting = E_ALL & ~E_NOTICE
; Включаем отображение ошибок в окне браузера
display_errors = On
; Включаем сохранение сообщений об ошибках
log_errors = On
; Ограничиваем длину сообщения об ошибках 1 Кбайт
log_errors_max_len = 1024
; Предписываем сохранение сообщений об ошибках в текстовый файл
error_log = "D:/php.log"
```

Теперь все ошибки, помимо отображения в окне браузера, будут дублироваться, в том числе в файл php.log; если директиве `error_log` присвоить значение `syslog`, сообщения будут сохраняться в службе регистрации событий (листинг 2.8).

Листинг 2.8. Регистрация ошибок операционной системой

```
; Реагируем на все сообщения об ошибках за исключением замечаний
error_reporting = E_ALL & ~E_NOTICE
; Включаем отображение ошибок в окне браузера
display_errors = On
; Включаем сохранение сообщений об ошибках
log_errors = On
; Ограничиваем длину сообщения об ошибках 1 Кбайт
log_errors_max_len = 1024
; Предписываем сохранение сообщений системной службой регистрации событий
error_log = syslog
```

Получить доступ к ошибкам можно при помощи консоли службы регистрации событий: **Пуск | Настройка | Панель управления | Администрирование | Просмотр событий**. В консоли следует выбрать раздел **Приложение** и ориентироваться на сообщения с источником PHP (рис. 2.11).

Двойной щелчок по событию позволяет извлечь сообщение об ошибке (рис. 2.12).

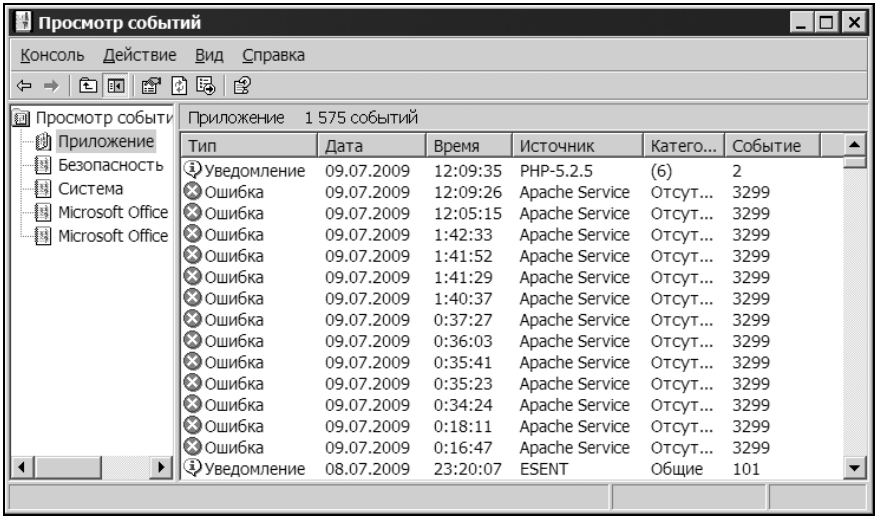


Рис. 2.11. Консоль службы регистрации событий

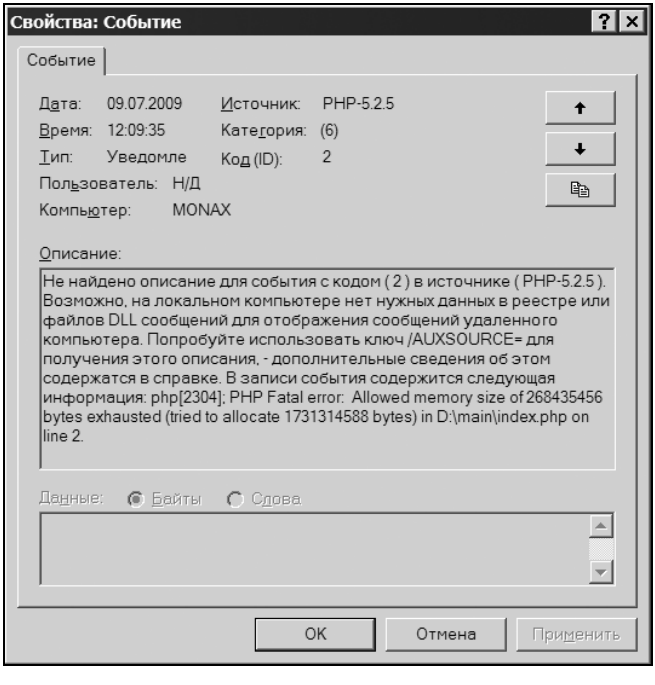


Рис. 2.12. Сообщение об ошибке

Рассмотрим более подробно директиву `error_reporting`. В отличие от большинства языков программирования, в PHP переменная создается сразу же при первом обращении. В ранних версиях языка это поощряло множество трудноулавливаемых ошибок и дыр в безопасности. Поэтому разработчиками принято решение ужесточить требование к переменным. Переменная по-прежнему создается при первом обращении к ней, однако, если она не была инициализирована при помощи оператора присваивания =, генерируется замечание (Notice). В листинге 2.9 приводится обращение к неинициализированной переменной `$user`, которое приводит к генерации замечания.

Листинг 2.9. Обращение к неинициализированной переменной `$user`

```
<?php
  echo $user;
?>
```

В результате обращения к неинициализированной переменной `$user` генерируется замечание "Notice: Undefined variable: user" ("Замечание: неопределенная переменная ") (рис. 2.13).

Рис. 2.13. Замечание, генерирующееся при обращении к неопределенной переменной

Такое поведение скрипта не всегда удобно, и любому разработчику предоставляется возможность самостоятельно настроить реакцию интерпретатора PHP. За чувствительность PHP к таким ситуациям несет ответственность директива `error_reporting` конфигурационного файла php.ini.

PHP имеет несколько уровней обработки ошибок: ошибки в ядре, при разборе скрипта, его интерпретации и пользовательские ошибки относятся к разным логическим уровням, и их смешение не желательно. Такое разделение ошибок на уровни позволяет более гибко настраивать интерпретатор. В табл. 2.10 приводится список уровней и соответствующие им предопределенные константы, используемые для настройки чувствительности интерпретатора PHP к ошибкам, как в конфигурационном файле php.ini, так и из скриптов.

Таблица 2.10. *Уровни обработки ошибок*

Константа	Значение	Описание
E_ERROR	1	Критическая ошибка исполнения, не совместимая с продолжением работы скрипта. Возникновение таких ошибок всегда сопровождается остановкой выполнения
E_WARNING	2	Предупреждения — некритическая ошибка исполнения скрипта. Возникновение таких ошибок не приводит к остановке выполнения
E_PARSE	4	Ошибка синтаксического разбора скрипта
E_NOTICE	8	Замечание — своеобразный совет по улучшению скрипта. Выдается, в частности, если встречается неинициализированная переменная
E_CORE_ERROR	16	Критическая ошибка, возникшая при инициализации ядра PHP
E_CORE_WARNING	32	Некритическая ошибка (предупреждение), возникшая при инициализации ядра PHP
E_COMPILE_ERROR	64	Критическая ошибка, возникшая во время компиляции PHP
E_COMPILE_WARNING	128	Некритическая ошибка (предупреждение), возникшая во время компиляции PHP
E_USER_ERROR	256	Критическая ошибка пользовательского уровня, генерируемая вызовом функции `trigger_error()`
E_USER_WARNING	512	Некритическая ошибка (предупреждение) пользовательского уровня, генерируемая вызовом функции `trigger_error()`
E_USER_NOTICE	1024	Замечание пользовательского уровня, генерируемое вызовом функции `trigger_error()`
E_STRICT	2048	Замечание, направленное на повышение переносимости PHP-приложений между различными операционными системами
E_RECOVERABLE_ERROR	4096	Критическая ошибка, которая может быть обработана при помощи пользовательского обработчика, назначенного функцией `set_error_handler()`. Если обработчик не назначен, такой вид ошибок получает статус `E_ERROR`
E_ALL	8191	Данный уровень включает все предыдущие ошибки

В листинге 2.10 приводится фрагмент конфигурационного файла php.ini, в котором PHP предписывается выводить все сообщения (`E_ALL`), кроме замечаний (`E_NOTICE`).

Замечания

Вместо констант из табл. 2.10 можно использовать их числовые значения, однако это не рекомендуется делать, т. к. они могут изменяться от версии к версии.

Листинг 2.10. Фрагмент конфигурационного файла php.ini

```
<?php
  ...
  error_reporting  =  E_ALL & ~E_NOTICE
  ...
?>
```

Примечание

Для комбинации нескольких констант можно использовать поразрядные операторы `|`, `~`, `!`, `^` и `&`.

Директива `error_reporting` устанавливает чувствительность PHP на уровне сервера. Установить собственный уровень для скрипта или Web-приложения можно при помощи одноименной функции, которая имеет следующий синтаксис:

```
error_reporting([$level])
```

Если указан необязательный параметр `$level`, функция устанавливает уровень чувствительности интерпретатора PHP. В качестве результата возвращает старый уровень (в виде числового значения).

В листинге 2.11 приводится пример использования функции `error_reporting()` для установки локального уровня чувствительности интерпретатора PHP.

Листинг 2.11. Использование функции `error_reporting()`

```
<?php
  error_reporting (E_ALL & ~E_NOTICE)
?>
```

Если в конфигурационном файле php.ini или в самом скрипте включено отображение сообщений об ошибке, детали выводятся в окно браузера, как, например, в вызове неизвестной функции (листинг 2.12).

Листинг 2.12. Вызов неизвестной функции, приводящий к сообщению об ошибке

```
<?php
  // Вызываем неизвестную функцию
  error1();
?>
```

Вызов неизвестной функции приводит к возникновению ошибки уровня `E_ERROR`, в результате чего в окно браузера выводится сообщение, представленное на рис. 2.14.

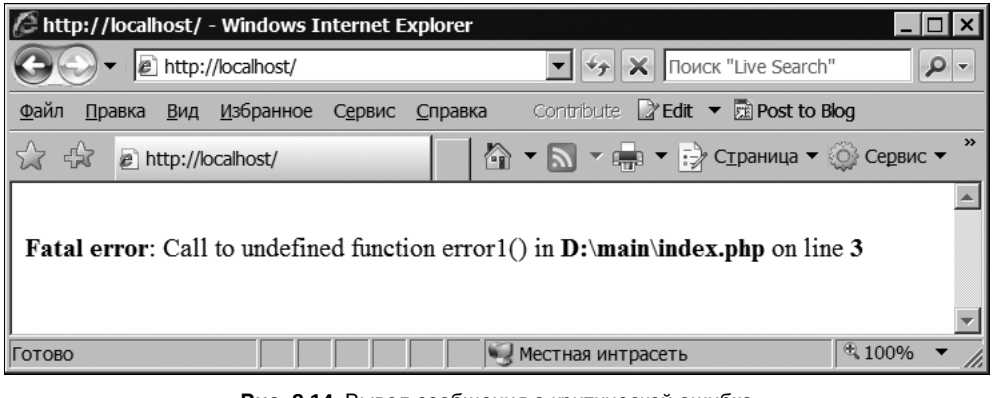


Рис. 2.14. Вывод сообщения о критической ошибке

2.1.9. Обработка входящих и исходящих данных

Директивы обработки данных влияют на предварительную обработку информации, поступающей извне, а также результата, сгенерированного скриптом, перед отправкой его клиенту. В табл. 2.11 представлен список директив данной группы.

ЗАМЕЧАНИЕ

Из PHP 6 исключен ряд устаревших директив, таких как `register_globals`, `register_long_arrays` и `magic_quotes_gpc`.

Таблица 2.11. *Директивы обработки данных*

Директива	Описание
`arg_separator.output`	Задает разделитель в адресах URL, которые генерируются PHP. В HTML для этого используется `&`, в XHTML — последовательность `&`
`arg_separator.input`	Задает разделитель, используемый для получения параметров из URL. Не рекомендуется задавать в этом параметре что-то, отличное от `&`, т. к. PHP может потерять возможность получать GET- и POST-параметры через соответствующие суперглобальные массивы `$_GET` и `$_POST`
`register_globals`	Включает (`On`) или выключает (`Off`) трансляцию переменных окружения, GET- и POST-данных, а также cookies в глобальные переменные
`variables_order`	Задает порядок трансляции внешних параметров в глобальные переменные при включенной директиве `register_globals`. По умолчанию директива принимает значение `EGPCS`: сначала обрабатываются переменные окружения операционной системы, потом GET-параметры, затем POST-параметры, cookies и переменные окружения Web-сервера (соответствующие суперглобальному массиву `$_SERVER`)
`request_order`	Задает порядок заполнения суперглобального массива `$_REQUEST` внешними параметрами. Так как параметры из разных источников могут обладать одинаковыми именами, элементу массива `$_REQUEST` с этим ключом присваивается значение из последнего источника. По умолчанию, директива принимает значение `GP`: сначала обрабатываются GET-, а затем POST-параметры
`register_long_arrays`	Включает (`On`) или отключает (`Off`) возможность использования длинных массивов, позволяет включить поддержку старых вариантов суперглобальных массивов
`register_argc_argv`	Включает (`On`) или выключает (`Off`) автоматическую регистрацию переменных `$argv` и `$argc`, содержащих соответственно массив с параметрами командной строки и число параметров. Такой режим очень удобен при использовании PHP в качестве интерпретатора системных скриптов, которые часто принимают дополнительные параметры. Однако при выполнении скрипта в условиях Web-сервера эти лишние переменные бесполезны и лишь расходуют ресурсы
`auto_globals_jit`	Если директива включена (`On`), суперглобальный массив `$_SERVER` создается только в момент обращения к нему из скрипта. В противном случае (`Off`) массив `$_SERVER` воссоздается каждый раз перед началом выполнения, независимо от того, используется он в скрипте или нет. Для работы включенного режима `auto_globals_jit` необходимо, чтобы были отключены директивы `register_globals`, `register_long_arrays` и `register_argc_argv`

Таблица 2.11 *(окончание)*

Директива	Описание
post_max_size	Задает максимальный размер POST-данных, которые может принять PHP-скрипт
magic_quotes_gpc	Включает (On) или выключает (Off) режим "магических кавычек" для внешних данных
magic_quotes_runtime	Включает (On) или выключает (Off) режим "магических кавычек" для данных, которые поступают от источников в момент выполнения скрипта (например, от СУБД или команды операционной системы, выполненной в обратных кавычках или функции system())
magic_quotes_sybase	Включает (On) или выключает (Off) режим "магических кавычек" с экранированием данных в стиле СУБД Sybase: кавычки ' экранируются удвоением '', вместо \'
auto_prepend_file	Задает имя файла, который автоматически добавляется в начало любого PHP-файла. Файл вызывается так, будто он был включен при помощи инструкции include
auto_append_file	Задает имя файла, который автоматически добавляется в конец любого PHP-файла. Файл вызывается так, будто он был включен при помощи инструкции include. Если скрипт останавливается при помощи exit(), файл не включается
default_mimetype	Задает MIME-тип в HTTP-заголовке Content-Type. По умолчанию принимает значение "text/html"
default_charset	Задает кодировку в HTTP-заголовке Content-Type. Чтобы отключить отправку данного HTTP-заголовка, необходимо установить пустое значение

Директива register_globals позволяет транслировать GET/POST-параметры, cookies и переменные окружения в глобальные значения. Пусть скрипту передается GET-параметр id следующим образом: index.php?id=17. В этом случае вывести значение параметра можно при помощи скрипта, представленного в листинге 2.13.

Листинг 2.13. Вывод GET-параметра в окно браузера

```
<?php
  echo $_GET['id'];
?>
```

Если директива register_globals принимает значение On, вывести значение GET-параметра можно при помощи глобальной переменной $id (листинг 2.14).

Листинг 2.14. Вывод GET-параметра id при включенной директиве register_globals

```
<?php
  echo $id;
?>
```

Несмотря на все удобство такой трансляции глобальных переменных, PHP-сообщество было вынуждено отказаться от такого режима работы из-за его потенциальной опасности. В листинге 2.15 приводится пример системы аутентификации, которая после проверки логина и пароля устанавливает переменной `$access` значение 1, если доступ разрешен, и значение 0, если аутентификация закончилась неудачей.

Листинг 2.15. Гипотетическая система аутентификации

```
<?php
  if ($_POST['name'] == "name" && $_POST['pass'] == "pass")
  {
    $access = 1;
  }
  if ($access)
  {
    // Доступ к закрытой части документа
  }
?>
```

Если скрипту из листинга 2.15 передать GET- или POST-параметр `access` со значением, отличным от нуля, то аутентификация будет автоматически пройдена, независимо от того, какие логин и пароль были переданы для проверки. Разумеется, брешь в безопасности можно легко устранить, явно инициализировав переменную `$access` вначале скрипта, однако в объемных проектах, состоящих из тысячи файлов, проблема может быть не столь очевидной. В любом случае, начиная с версии PHP 6, директива `register_globals` исключена из файла php.ini.

Еще одной неоднозначной директивой является `magic_quotes_gpc`, позволяющая включить режим "магических кавычек". Директива вводилась для того, чтобы автоматически экранировать кавычки, предотвращая нарушение синтаксиса SQL-запросов и SQL-инъекции при помещении данных, переданных со стороны клиента в базу данных. Однако де-факто директива породила очередную проблему: некоторые виды СУБД (например Sysbase) требуют экранировать одиночную кавычку `'` не обратным слэшем `\'`, а повторением кавычки `''`. Директиву можно включить на сервере или отключить в зависимости от используемой СУБД и предпочтений системного администратора. В результате от переносимого PHP-приложения требуется обработка внешних параметров в зависимости от того, включен режим "магических кавычек" или нет. Выяснить это помогает специальная функция `get_magic_quotes_gpc()`, которая возвращает `TRUE`, если режим включен, и `FALSE` в противном случае. В листинге 2.16 представлен типичный код обработки внешних параметров.

Замечание

Функция `mysql_escape_string()` из состава библиотеки mysql осуществляет экранирование кавычек.

Листинг 2.16. Обработка параметров в зависимости от состояния директивы `magic_quotes_gpc`

```
<?php
  ...
  if (!get_magic_quotes_gpc())
  {
    // Режим "магических кавычек" отключен
    $_POST['name'] = mysql_escape_string($_POST['name']);
    $_POST['pass'] = mysql_escape_string($_POST['pass']);
  }
  ...
?>
```

Как видно из листинга 2.16, если режим "магических кавычек" включен, блок `if` не выполняется, если отключен — выполняется. Начиная с PHP 5.3, директива `magic_quotes_gpc` исключается из php.ini, а функция `get_magic_quotes_gpc()` всегда возвращает `FALSE`.

Определенный интерес представляют директивы `auto_prepend_file` и `auto_append_file`, позволяющие подключить соответственно в начало и конец PHP-файла любой другой PHP-документ.

Пусть имеются два PHP-файла: php_start.php (листинг 2.17) и php_end.php (листинг 2.18), которые соответственно выводят фразы "Start PHP" и "End PHP".

> ***Замечание***
>
> В скрипты, подключаемые директивами `auto_prepend_file` и `auto_append_file`, обычно помещают более практичные вещи, такие как библиотеки общих файлов, системы подсчета времени генерации страницы, счетчики посещения и т. п.

Листинг 2.17. Файл php_start.php

```
<?php
  echo "Start PHP<br />";
?>
```

Листинг 2.18. Файл php_end.php

```
<?php
  echo "End PHP<br />";
?>
```

Назначим директиве `auto_prepend_file` скрипт php_start.php, а директиве `auto_append_file` — скрипт php_end.php:

```
auto_prepend_file = D:/php_start.php
auto_append_file = D:/php_end.php
```

Теперь содержимое этих файлов будет подключаться в начале и конце любого скрипта так, как если бы они содержали соответствующие `include`-инструкции (листинг 2.19).

Листинг 2.19. Скрипт index.php

```
<?php
  echo "Contents<br />";
?>
```

Результат выполнения скрипта из листинга 2.19 представлен на рис. 2.15.

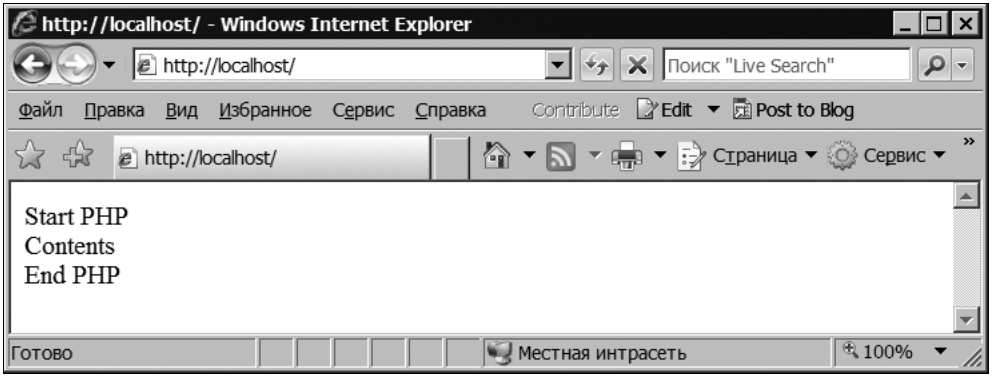


Рис. 2.15. Результат выполнения скрипта из листинга 2.19

Скрипт php_end.php подключается лишь в том случае, если обработчик PHP штатно доходит до конца файла. Если же его работа прерывается, например, при помощи `exit()`, файл php_end.php не подключается.

2.1.10. Загрузка файлов

Директивы управления загрузкой файлов позволяют настроить параметры, влияющие на обработку файлов, загруженных через поле `file` Web-формы (табл. 2.12).

***Таблица 2.12.** Директивы управления загрузкой файлов*

Директива	Описание
`file_uploads`	Включает (`On`) или отключает (`Off`) загрузку файлов по протоколу HTTP (через поле `file`)
`upload_tmp_dir`	Задает путь к каталогу, в который помещаются временные файлы
`upload_max_filesize`	Задает максимальный размер загружаемого на сервер файла

2.1.11. Сетевой доступ

Директивы управления сетевым доступом позволяют настроить параметры, которые влияют на возможности функций, поддерживающих удаленное обращение по сети (табл. 2.13).

Таблица 2.13. *Директивы сетевого доступа*

Директива	Описание
allow_url_fopen	Включает (On) или отключает (Off) возможность обращения файловых функций по сетевому пути (начинающемуся с http://)
allow_url_include	Включает (On) или отключает (Off) возможность обращения инструкций include и require по сетевому пути (начинающемуся с http://)
from	Определяет логин для анонимного FTP-доступа. Традиционно для такого доступа используется e-mail
user_agent	Задает пользовательский агент USER_AGENT, который отправляет PHP при обращении к URL при помощи любой функции, поддерживающей сетевой доступ
default_socket_timeout	Задает время ожидания ответа сервера

2.1.12. Подключение расширений

Помимо ядра языка PHP поддерживает подключение библиотек расширений. Некоторые расширения жестко компилируются вместе с ядром в единый модуль, некоторые подключаются отдельно в виде внешних библиотек. В табл. 2.14 представлены директивы данной группы.

Таблица 2.14. *Директивы управления расширениями*

Директива	Описание
extension_dir	Задает путь к каталогу с библиотеками расширений
extension	Задает название библиотеки расширения, для каждой из библиотек необходима отдельная директива extension

По умолчанию, начиная с PHP 5, все внешние расширения отключены. Это означает, что если имеется потребность работать с базой данных MySQL и динамической графикой (GDLib), следует явно подключить эти расширения в конфигурационном файле php.ini.

За подключение расширений несет ответственность директива extension. Для подключения расширения необходимо снять комментарий (символ #) напротив его. Так для подключения расширения для работы с MySQL необходимо снять комментарий напротив директивы:

```
extension=php_mysql.dll
```

Замечание

Ряд расширений (php_mcrypt.dll) требует дополнительных динамических библиотек, которые необходимо будет поместить в каталог C:\Windows\system32. Часть библиотек расположена в C:\php, часть придется загрузить из Интернета.

Помимо этого необходимо настроить директиву, указав путь к библиотекам расширения:

```
extension_dir = "C:\Program Files\PHP\ext"
```

2.1.13. Настройка сессии

В секции `[Session]` конфигурационного файла php.ini сосредоточены директивы, управляющие поведением сессии. В табл. 2.15 приводится список наиболее часто используемых директив.

Таблица 2.15. *Директивы управления сессией*

Директива	Описание
`session.save_handler`	Задает обработчик механизма сессий, по умолчанию директива принимает значение `files` (сохранение данных сессии в файле в стерилизованном виде). Данная директива предназначена для смены механизма сессии (если он реализован в виде альтернативного расширения)
`session.save_path`	Задает путь к каталогу, в который сохраняются файлы сессии
`session.use_cookies`	Включает (`On`) или отключает (`Off`) использование cookies для хранения *уникального идентификатора сессии* SID. Если директива отключена, SID передается через GET-параметр
`session.cookie_secure`	Если директива включена (`On`), cookies SID могут передаваться только через защищенный канал (https://, протокол HTTP поверх SSL-протокола)
`session.use_only_cookies`	Если директива включена (`On`), PHP использует только cookies для передачи SID, передача их через GET-параметры запрещается
`session.name`	Задает имя сессии по умолчанию, если оно не задается явно функцией `session_name()`
`session.auto_start`	Включает (`On`) или отключает (`Off`) автоматический старт сессии в каждом PHP-файле. В отключенном состоянии сессию необходимо инициировать самостоятельно при помощи функции `session_start()`
`session.cookie_lifetime`	Задает время жизни cookies SID. Если директива принимает значение 0, используются сессионные cookies, которые сохраняются до момента выключения браузера
`session.cookie_path`	Задает область действия cookie SID на сайте, Значение по умолчанию/, означающее, что cookie действует от корня и охватывает весь сайт
`session.cookie_domain`	Задает домен действия cookies
`session.serialize_handler`	Задает обработчик сериализации сессии. По умолчанию принимает значение `php`, означающее, что для сериализации используется стандартная функция `serialize()`

Таблица 2.15 *(окончание)*

Директива	Описание
session.gc_probability	Задает вероятность старта процесса "сборки мусора" при очередной инициализации сессии. Вероятность определяется отношением session.gc_probability/session.gc_divisor. По умолчанию принимает значение 1
session.gc_divisor	Задает вероятность старта процесса "сборки мусора" при очередной инициализации сессии. Вероятность определяется отношением session.gc_probability/session.gc_divisor. По умолчанию принимает значение 100
session.gc_maxlifetime	Задает время, после которого сохраненные данные рассматриваются как устаревшие и подлежащие уничтожению "сборщиком мусора". По умолчанию принимает значение 1440

2.1.14. Настройка даты и времени

Директивы группы настройки даты и времени располагаются в секции [Date] (табл. 2.16).

ЗАМЕЧАНИЕ

Параметры из табл. 2.16 используются функциями date_sunrise(), date_sunset() и date_sun_info().

Таблица 2.16. *Директивы управления датой и временем*

Директива	Описание
date.timezone	Задает часовой пояс по умолчанию
date.default_latitude	Задает широту по умолчанию
date.default_longitude	Задает долготу по умолчанию
date.sunrise_zenith	Задает зенит (90° для точки горизонта) по умолчанию для функции date_sunrise()
date.sunset_zenith	Задает зенит (90° для точки горизонта) по умолчанию для функции date_sunset()

2.1.15. Изменение настроек php.ini средствами Apache

В предыдущем разделе были рассмотрены наиболее часто редактируемые директивы конфигурационного файла php.ini. Сложность заключается в том, что конфигурационный файл php.ini присутствует в единственном экземпляре, а сервер может поддерживать сотни сайтов, для каждого из которых могут требоваться различные настройки. Одним из решений данной проблемы является настройка директив конфигурационного файла php.ini при помощи конфигурационных файлов httpd.conf и .htaccess.

Для того чтобы иметь возможность изменять настройки файла php.ini из конфигурационных файлов httpd.conf и .htaccess, необходимо, чтобы директива `AllowOverride` контейнера `<Directory />` имела значение `Options` или `All` (листинг 2.20).

Листинг 2.20. Настройка директивы `AllowOverride` в httpd.conf

```
<Directory />
    Options All
    AllowOverride All
    Order deny,allow
</Directory>
```

Конфигурационный файл http.conf осуществляет глобальную настройку Web-сервера Apache, в то время как настройки конфигурационного файла .htaccess действуют на текущий каталог (а также вложенные в него подкаталоги).

Установка настроек PHP из конфигурационных файлов httpd.conf и .htaccess осуществляется четырьмя директивами:

- `php_admin_value` — устанавливает строковый параметр php.ini;
- `php_admin_flag` — устанавливает логические параметры php.ini (т. е. те, которые принимают значения `On` или `Off`);
- `php_value` — устанавливает строковый параметр php.ini;
- `php_flag` — устанавливает логические параметры php.ini (т. е. те, которые принимают значения `On` или `Off`);

Директивы `php_admin_value` и `php_admin_flag` в отличие от `php_value` и `php_flag` допускаются к использованию только в конфигурационном файле httpd.conf. Параметр, установленный при помощи данных директив, не может быть изменен впоследствии при помощи конфигурационного файла .htaccess. Директивы `php_value` и `php_flag` могут быть использованы везде, а установленные ими значения переназначены.

В листинге 2.21 приводится пример установки директиве `open_basedir` значения "/www". Таким образом, область действия PHP ограничивается каталогом /www, и никто, кроме администратора сервера, не может изменить эту настройку.

Листинг 2.21. Ограничение действия PHP каталогом /www

```
<Directory /www>
  php_admin_value open_basedir /www
</Directory>
```

Как правило, директиву `open_basedir` применяют более гибко, поместив ее в виртуальном хосте сайта и передав в качестве значения путь к каталогу, где расположен виртуальный хост, можно добиться поведения PHP, при котором скрипты одного сайта не имеют доступа к файлам соседнего виртуального хоста (листинг 2.22).

Листинг 2.22. Разграничение виртуальных хостов

```
# Первый виртуальный хост
<VirtualHost 127.0.0.1:80>
    ServerAdmin suport@domain1.ru
    DocumentRoot "/www/domain1/"
    ServerName domain1.ru
    php_admin_value open_basedir "/www/domain1/"
    ErrorLog "/www/domain1/logs/domain1-error.log"
    CustomLog "/www/domain1/logs/domain1-access.log" common
</VirtualHost>
# Второй виртуальный хост
<VirtualHost 127.0.0.1:80>
    ServerAdmin suport@domain2.ru
    DocumentRoot "/www/domain2/"
    ServerName domain2.ru
    php_admin_value open_basedir "/www/domain2/"
    ErrorLog "/www/domain1/logs/domain2-error.log"
    CustomLog "/www/domain1/logs/domain2-access.log" common
</VirtualHost>
```

Для изменения на уровне .htaccess удобнее воспользоваться директивами `php_value` и `php_flag`. Удобство использования конфигурационного файла .htaccess заключается в том, что для его применения не требуется редактирование файла httpd.conf, к которому имеет доступ только администратор хостинга. Использование .htaccess, как правило, доступно любому разработчику, имеющему FTP-доступ к сайту.

В листинге 2.23 приводится пример включения директивы `register_long_arrays`, разрешающей использование длинных вариантов суперглобальных массивов, а также директивы `auto_prepend_file`, подключающей к каждому PHP-файлу библиотеку include.utils.php.

> ***Замечание***
>
> При использовании директивы `auto_prepend_file` следует указывать полный путь к включаемому файлу от корня диска.

Листинг 2.23. Конфигурационный файл .htaccess

```
php_flag register_long_arrays on
php_value auto_prepend_file /www/domain/include/include.utils.php
```

2.1.16. Функции управления интерпретатором *PHP*

Несмотря на то, что конфигурационные файлы Apache позволяют достаточно гибко изменять параметры php.ini, они лишены динамичности PHP-кода и не могут использовать условные выражения. Поэтому PHP предоставляет разработчикам ши-

рокий спектр функций, позволяющих влиять на параметры PHP непосредственно из скриптов (табл. 2.17).

Таблица 2.17. *Функции управления интерпретатором PHP*

Директива	Описание
`ini_set($varname, $newvalue)`	Устанавливает директиве `$varname` новое значение `$newvalue`. В случае успеха возвращает старое значение директивы. Если установить новое значение не удается, возвращается `FALSE`
`ini_get($varname)`	Возвращает текущее значение директивы `$varname`
`ini_get_all($extension)`	Возвращает ассоциативный массив с директивами расширения `$extension`
`ini_restore($varname)`	Восстанавливает значение директивы `$varname`, присваивая ей значение, установленное в конфигурационном файле php.ini

В листинге 2.24 демонстрируется типичное использование функций управления интерпретатором PHP.

Листинг 2.24. Использование функций управления интерпретатором PHP

```
<?php
  // Читаем содержимое директивы post_max_size
  echo ini_get("post_max_size");
  // Устанавливаем новое значение post_max_size
  ini_set("post_max_size", "32M");
  // Восстанавливаем первоначальное значение
  ini_restore("post_max_size");
?>
```

Если необходимо запретить изменение директив из скрипта, следует либо воспользоваться безопасным режимом *(см. разд. 2.1.3)*, либо определить окончательное значение директивы при помощи директивы `php_admin_value` на уровне конфигурационного файла httpd.conf *(см. разд. 2.1.15)*.

Помимо функций, позволяющих явно управлять значениями директив конфигурационного файла, PHP предоставляет множество справочных и специализированных функций по конкретным директивам (табл. 2.18).

Таблица 2.18. *Справочные функции*

Директива	Значение
`memory_get_usage()`	Возвращает количество байтов, которые занимает текущий скрипт
`php_sapi_name()`	Возвращает тип подключения PHP к Web-серверу. Если PHP подключен как внешнее CGI-приложение, возвращается строка `"cgi"`, если как модуль — строка `"apache"`

Таблица 2.18 *(окончание)*

Директива	Значение
`php_uname([$mode])`	Возвращает информацию об операционной системе, в которой выполняется PHP. Необязательный параметр `$mode` принимает значения, совпадающие с параметрами утилиты `uname`
`phpinfo()`	Возвращает подробную информацию о PHP, его расширениях, установленных переменных окружения в виде PHP-страницы
`phpversion([$extension])`	Возвращает версию PHP. Можно узнать версию отдельного расширения, если передать его имя при помощи необязательного параметра `$extension`

Используя функцию `memory_get_usage()`, можно отслеживать количество занятой скриптом оперативной памяти, чтобы не превысить предел, заданный директивой `memory_limit`, и не привести к возникновению критической ошибки.

Рис. 2.16. Результат работы функции `phpinfo()`

Функция `php_uname()` возвращает информацию об операционной системе, в которой выполняется PHP-скрипт. При помощи единственного необязательного параметра можно задать тип возвращаемой информации:

- `'s'` — название операционной системы;
- `'n'` — имя хоста;
- `'r'` — версия PHP;
- `'v'` — версия операционной системы;
- `'m'` — архитектура машины, например, i386;
- `'a'` — включает всю вышеперечисленную информацию.

В листинге 2.25 приводится пример использования функции `phpinfo()`, позволяющей получать подробнейшую информацию о текущей версии PHP в удобной для анализа форме.

Листинг 2.25. Использование функции `phpinfo()`

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

Результат работы функции `phpinfo()` представлен на рис. 2.16.

2.1.17. PHP как консольный интерпретатор

PHP можно применять не только для построения динамических Web-страниц, но также и для выполнения различных системных скриптов, например, для ежедневного создания и упаковки в массив резервных копий данных, для автоматической обработки текстовых файлов в каталоге и т. п.

В UNIX-подобных операционных системах для этого достаточно выставить права доступа на файл таким образом, чтобы он стал исполняемым, и добавить в начало скрипта путь к PHP-интерпретатору после последовательности `#!` (листинг 2.26).

ЗАМЕЧАНИЕ

Символ `#` в PHP (и вообще в shell-скриптах) является комментарием, но последовательность `#!` (которая в английском варианте называется bang line, hash-bang или she-bang) имеет специальное значение — она указывает путь к интерпретатору скрипта. Дело в том, что в UNIX-подобных операционных системах скрипты могут создаваться на нескольких языках: это и Perl, и язык оболочек, реже PHP или Python. Когда скрипт выполняется Web-сервером, он ориентируется на расширение файла. UNIX-подобные операционные системы на расширение файла, как правило, не ориентируются — его зачастую у файла просто нет. Система читает первую строку и ищет его обработчик. Вместо `#!/usr/bin/php` можно написать `#!/usr/bin/sh` или `#!/usr/bin/perl`. Кроме того, интерпретаторы не обязательно могут находиться по пути /usr/bin — bang line позволяет уточнить путь.

Листинг 2.26. Путь к PHP-интерпретатору в начале файла

```
#!/usr/bin/php
<?php
  // PHP-код
?>
```

В Windows при помощи последовательности `#!` управлять обработчиком нельзя, операционная система ориентируется на расширение файла. Процедура запуска скрипта требует загрузки браузера и набора сетевого адреса. Однако этого можно избежать, назначив в качестве обработчика расширению php консольный интерпретатор PHP. Для этого необходимо выделить любой PHP-файл и в контекстном меню выбрать пункт **Свойства**. Появится окно настройки свойств (рис. 2.17).

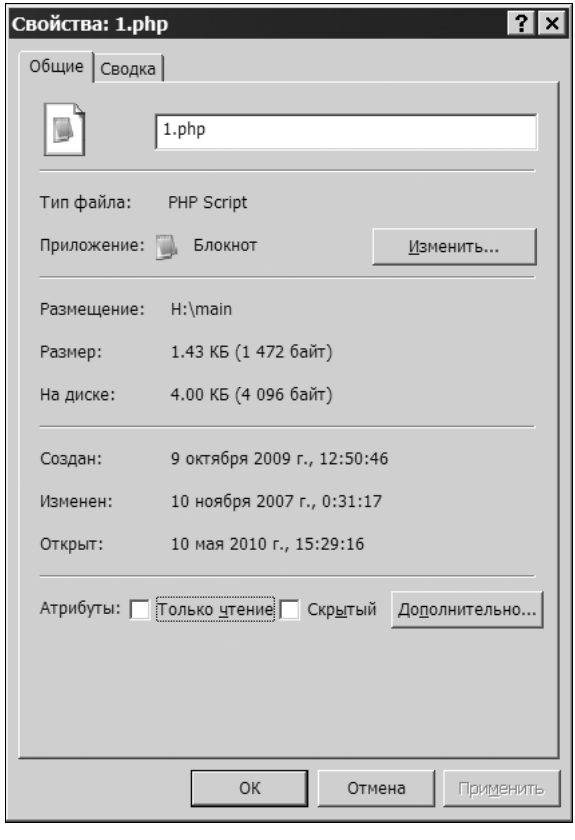


Рис. 2.17. Диалоговое окно **Свойства**

Далее нажмите кнопку **Изменить**. В открывшемся окне нажмите кнопку **Найти**. В открывшемся диалоговом окне выберите файл C:\PHP\php.exe (рис. 2.18).

Замечание

В корневом каталоге PHP находятся три исполняемые модуля: php.exe, php-cgi.exe и php-win.exe. php-cgi.exe предназначен для совместной работы с Web-сервером, именно он об-

рабатывает запросы к PHP-скриптам, если PHP установлен не модулем. php.exe предназначен для консольной обработки скриптов, при его запуске появляется черное окно консоли, в котором отображается весь внешний вывод скрипта. php-win.exe позволяет запускать PHP-скрипты без открытия окна консоли в качестве процесса со скрытым окном. Если вы не хотите, чтобы при запуске PHP-скриптов открывались окна, можно выбрать именно этот обработчик.

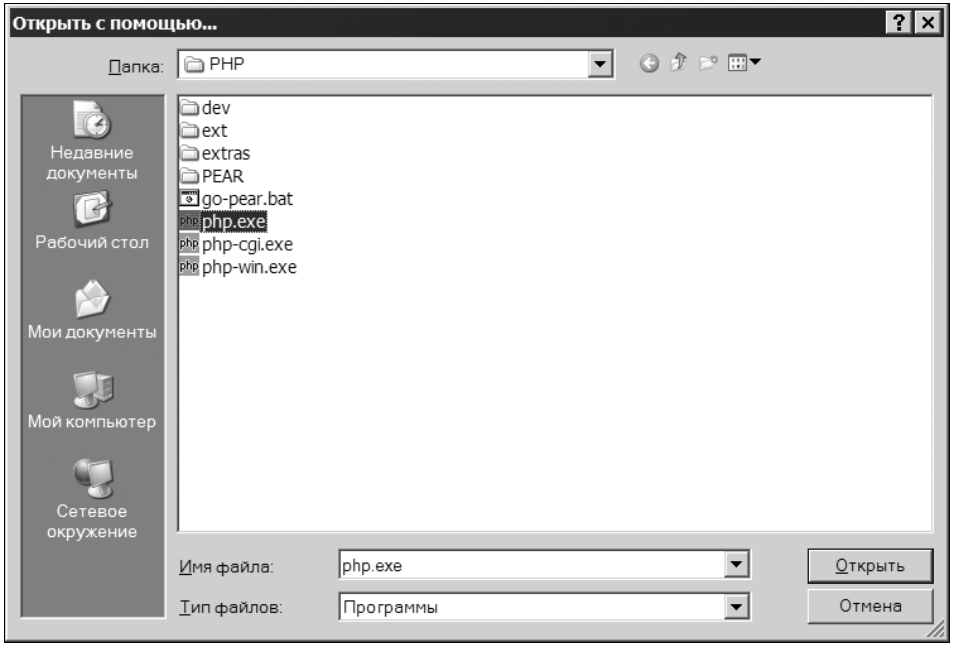


Рис. 2.18. Диалоговое окно открытия файлов

В результате этого в окне выбора обработчика PHP-файлов появится обработчик CLI, как это показано на рис. 2.19. Теперь PHP-скрипты можно запускать нажатием клавиши <Enter> или двойным щелчком мыши — они будут вести себя как обычные программы.

Теперь для того чтобы проверить работоспособность только что созданной схемы, сформируем простейший скрипт index.php, размещающий в корне диска C:\ файл hello, в котором будет помещена фраза `"Hello world!"` (листинг 2.27).

Листинг 2.27. Файл index.php

```
<?php
  // Открываем файл
  $fd = fopen("C:/hello","w");
  // Записываем фразу
  fwrite($fd, "Hello world!");
  // Закрываем файл
  fclose($fd);
?>
```

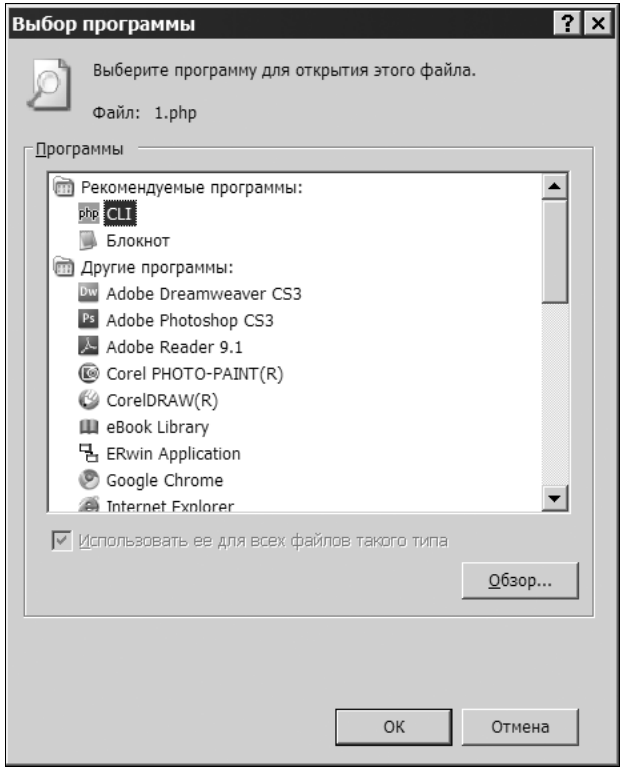


Рис. 2.19. Назначение в качестве обработчика PHP-файлов консольного интерпретатора PHP

Если все было сделано правильно, то двойной щелчок мыши или нажатие клавиши <Enter> по файлу index.php приведет к кратковременному открытию окна консоли и появлению на диске C:\ файла hello.

2.1.18. Запуск скриптов в назначенное время

В состав операционной системы UNIX обязательно входит демон cron, который позволяет выполнять скрипты в строго назначенное время. Большинство хост-провайдеров предоставляют доступ к этому сервису для запуска PHP-скриптов. В Windows XP также имеется встроенный планировщик заданий, однако большинство серверов работают под управлением UNIX-подобных операционных систем. Поэтому даже если для разработки Web-сайтов используется операционная система Windows, имеет смысл установить классический планировщик заданий UNIX — cron.

Классическую реализацию cron для Windows можно загрузить по адресу **http://www.nncron.ru/download.shtml**. На странице представлено две версии cron — nnCron, условно бесплатная программа с Windows-интерфейсом, и nnCron LITE — бесплатная программа, с классическим интерфейсом через конфигурационный файл crontab. Рекомендуется использовать именно nnCron LITE, т. к. знание синтаксиса конфигурационного файла cron.tab позволит без труда работать с UNIX-

версией cron на сервере хост-провайдера. На странице **http://www.nncron.ru/download.shtml** можно также обнаружить русскую документацию по синтаксису cron.tab и плагины к nnCron.

Для того чтобы скрипт можно было запускать при помощи cron, необходимо назначить ему права доступа на исполнение. В UNIX это осуществляется при помощи команды `chmod`.

Права доступа в UNIX выставляются для трех категорий пользователей:

- владельца файла;
- группы владельца;
- остальных пользователей.

В рамках каждой из этих трех групп существуют три вида разрешений:

- `r` — право чтения данного файла;
- `w` — право записи/изменения данного файла;
- `x` — право выполнения данного файла, если он является сценарием или программой (для каталогов право просмотра).

Таким образом, права доступа могут выглядеть так: `rwxr--r--`. UNIX-права доступа можно задавать восьмеричным числом. При этом праву чтения соответствует цифра 4, праву записи — 2, а исполнению — 1. Общие права для групп задаются суммой этих чисел, так 6 (4 + 2) обеспечивает возможность чтения и записи, а 7 (4 + 2 + 1) — предоставляет полный доступ к файлу или каталогу. Тогда для каталога восьмеричное число 0755 означает, что владелец каталога имеет полный доступ (`rwx`), а все остальные имеют право читать файлы в нем и просматривать содержимое каталога (`r-x`). Чтобы назначить файлу index.php права доступа на выполнение `rwxr-xr-x` (0755), необходимо выполнить команду:

```
chmod 755 index.php
```

Если для этих целей используется PHP, то код, выполняющий данную команду, будет выглядеть так, как это представлено в листинге 2.28.

Листинг 2.28. Смена прав доступа

```
<?php
  chmod("index.php", 0755);
?>
```

Для того чтобы можно было выполнять PHP-файлы в Windows, необходимо назначить расширению php в качестве обработчика PHP-интерпретатор, т. к. это описывается в *разд. 2.1.17*.

Назначение заданий осуществляется в файле cron.tab.

Замечание

Если вы воспользовались nnCron для Windows, файл cron.tab можно обнаружить в каталоге C:\Program Files\cron.tab, в UNIX файл cron.tab находится, как правило, в каталоге \etc. (Для

более детальной информации по cron.tab в UNIX следует обратиться к документации конкретной системы.)

Каждая строка файла cron.tab соответствуют одному заданию. Помимо заданий в cron.tab-файле допускаются комментарии, начинающиеся с символа диеза `#`. Формат строки задания выглядит следующим образом:

минуты часы день_месяца месяц день_недели команда

Параметры могут принимать следующие значения

- *минуты* — 0–59;
- *часы* — 0–23;
- *день_месяца* — 1–31;
- *месяц* — 1–12;
- *день_недели* — 0–7 (0 и 7 означают воскресенье);
- *команда* — команда, которая должна быть выполнена, например, `d:/mail/reserve.php`.

Символ `*` означает диапазон с первого до последнего, например, следующее задание

```
0 23 * * * d:/main/reserve.php
```

означает запуск скрипта d:/main/reserve.php каждый день в 23:00. Допускается указание нескольких значений каждого из параметра через запятую. Так задание

```
0 0,8,16 * * * d:/main/cleanup.php
```

означает запуск скрипта d:/main/cleanup.php каждый день в 0:00, 8:00 и 16:00.

ЗАМЕЧАНИЕ

Пробелы служат разделителями между параметрами, поэтому недопустимы пробелы после запятых, т. е. задание `0 0, 8, 16 * * * d:/main/cleanup.php` не будет являться корректным.

Задание

```
0,30 18-23 * * * /home/root/check.php
```

будет запускать скрипт /home/root/check.php каждые полчаса между 18:00 и 23:00. Задание

```
0/10 * * * * /home/root/log.php
```

будет запускать скрипт /home/root/log.php каждые 10 минут. Именно такой режим лучше всего подходит для проверки наличия ссылки на удаленной странице сайта.

ЗАМЕЧАНИЕ

Хост-провайдеры не приветствуют частые запуски скриптов. Перед тем как ставить задания, следует проконсультироваться в службе технической поддержки, какие ограничения существуют у данного хост-провайдера на частоту запуска скриптов и количества заданий.

2.2. Apache

2.2.1. Конфигурационный файл .htaccess

Конфигурационный файл Apache .htaccess управляет поведением Web-сервера для текущего каталога и всех вложенных каталогов.

Не все директивы, которые применяются в конфигурационном файле httpd.conf, допускается использовать в конфигурационном файле .htaccess. К таким директивам, например, относится директива `AccessFileName`, которая определяет имя файла .htaccess (листинг 2.29).

Листинг 2.29. Использование директивы `AccessFileName` в httpd.conf

```
AccessFileName .htaccess
```

Вместо имени .htaccess можно использовать любое другое имя, однако следует иметь в виду, что потребуется редактирование контейнера `Files`, который запрещает доступ к файлам, начинающимся с префикса .ht (листинг 2.30), в противном случае пользователи получат возможность просмотра содержимого конфигурационных файлов.

Листинг 2.30. Запрет доступа к файлам, начинающимся с префикса .ht

```
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
    Satisfy All
</Files>
```

Однако не только приведенные директивы требуются для успешного функционирования конфигурационных файлов .htaccess. Помимо этого необходимо настроить директиву `AllowOverride`, определяющую группы директив, использование которых разрешено в конфигурационном файле .htaccess. Как правило, директива прописывается в контейнере `<Directory />` и определяет настройки для всего сервера.

Директива `AllowOverride` может принимать одно из следующих значений или их комбинацию: `All`, `None`, `AuthConfig`, `FileInfo`, `Indexes`, `Limit`, `Options`. Если указывается несколько значений, то они должны быть разделены пробелом (листинг 2.31).

Листинг 2.31. Использование комбинации значений

```
<Directory />
    Options FollowSymLinks Indexes MultiViews
    AllowOverride AuthConfig FileInfo Indexes Limit Options
</Directory>
```

Значение `All` включает все директивы, доступные для использования в конфигурационном файле .htaccess, а значение `None`, соответственно, отключает.

Каждое из значений директивы `AllowOverride` разрешает использование в конфигурационном файле .htaccess определенной группой директив, которые описываются в табл. 2.19.

Таблица 2.19. *Значения директивы* `AllowOverride`

Значение	Описание
`AuthConfig`	Директива разрешает использование в конфигурационном файле .htaccess директив аутентификации и управления доступом (таких как `AuthDBMGroupFile`, `AuthDBMUserFile`, `AuthGroupFile`, `AuthName`, `AuthType`, `AuthUserFile` и `Require`)
`FileInfo`	Директива разрешает использование в конфигурационном файле .htaccess директив, управляющих типами документов (`AddEncoding`, `AddLanguage`, `AddType`, `DefaultType`, `ErrorDocument` и `LanguagePriority`)
`Indexes`	Директива разрешает использование в конфигурационном файле .htaccess директив, управляющих индексами каталогов (`AddDescription`, `AddIcon`, `AddiconByEncoding`, `AddIconByType`, `DefaultIcon`, `DirectoryIndex`, `FancyIndexing`, `HeaderName`, `IndexIgnore`, `IndexOptions` и `RenameName`)
`Limit`	Директива разрешает использование в конфигурационном файле .htaccess директив, управляющих доступом к узлам (`Allow`, `Deny` и `Order`)
`Options`	Директива разрешает использование в конфигурационном файле .htaccess директив, управляющих свойствами каталогов (`Option` и `XBitHack`)

2.2.2. Установка кодировки по умолчанию

По умолчанию, если документ не содержит `META`-тега с указанием кодировки документа, Apache считает, что используется западноевропейская кодировка ISO-8859-1. В связи с этим клиенту отправляется HTTP-заголовок:

```
Content-Type: text/html; charset=ISO-8859-1
```

Это может приводить к искажению содержимого документа, если он набран в другой кодировке, например, cp1251 (рис. 2.20).

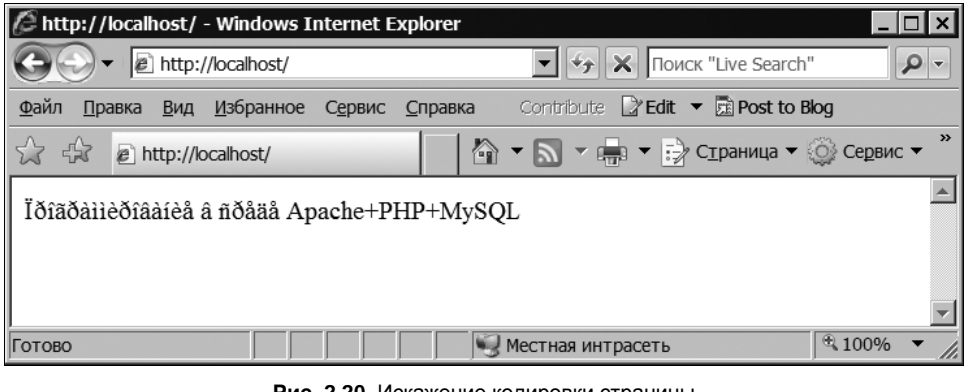


Рис. 2.20. Искажение кодировки страницы

Для того чтобы избежать искажения кодировки, как показано на рис. 2.20, можно создать в каталоге с такими документами конфигурационный файл .htaccess с директивой `AddDefaultCharset`, которая установит нужную кодировку (листинг 2.32).

Листинг 2.32. Управление кодировкой файлов в текущем каталоге

```
AddDefaultCharset Windows-1251
```

Результатом этого будет корректное отображение файлов с кодировкой ср1251 (Windows-1251), даже если документы не определяют собственную кодировку при помощи `META`-тега (рис. 2.21).

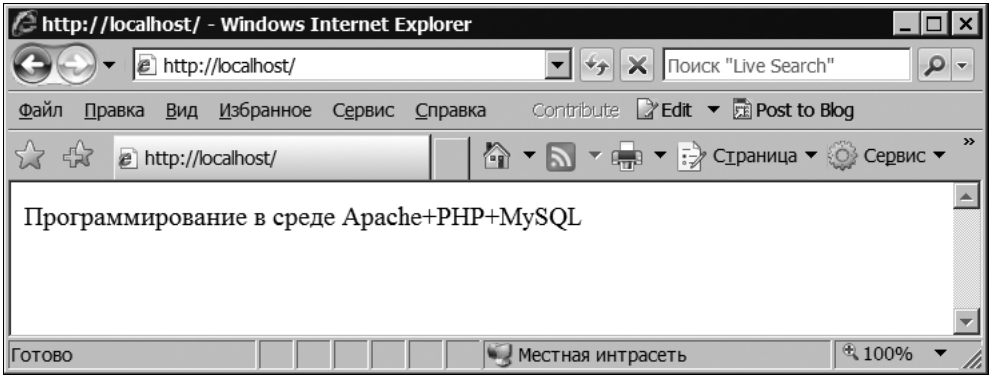


Рис. 2.21. Корректное отображение кодировки страницы

Директива `AddDefaultCharset` может принимать значения из табл. 2.20.

Таблица 2.20. *Значения, которые может принимать директива* `AddDefaultCharset`

Значение	Описание
`ISO-8859-1`	Западноевропейский
`ISO-8859-15`	Западноевропейский с поддержкой символа Евро
`Windows-1252`	Западноевропейский
`CP850`	Западноевропейский
`CP866`	Кириллица (DOS)
`Windows-1251`	Кириллица (Windows)
`KOI8-R`	Кириллица (UNIX)
`UTF-8`	8-битовая кодировка Unicode
`UTF-7`	7-битовая кодировка Unicode
`UCS-2`	16-битовая кодировка Unicode

2.2.3. Список файлов в каталоге

Если пользователь не указывает имя файла в каталоге, например, **http://localhost/**, вместо **http://localhost/index.php**, то Web-сервер Apache самостоятельно ищет индексный файл из списка в директиве `DirectoryIndex` (листинг 2.33).

Листинг 2.33. Директива `DirectoryIndex`

```
DirectoryIndex index.html index.php
```

Web-сервер Apache последовательно ищет файлы, указанные в директиве `DirectoryIndex`, и если такой файл найден — выводится его содержимое. Что же происходит, когда текущий каталог не содержит индексного файла? По умолчанию возвращается ошибка 403 — доступ к содержимому каталога запрещен (рис. 2.22).

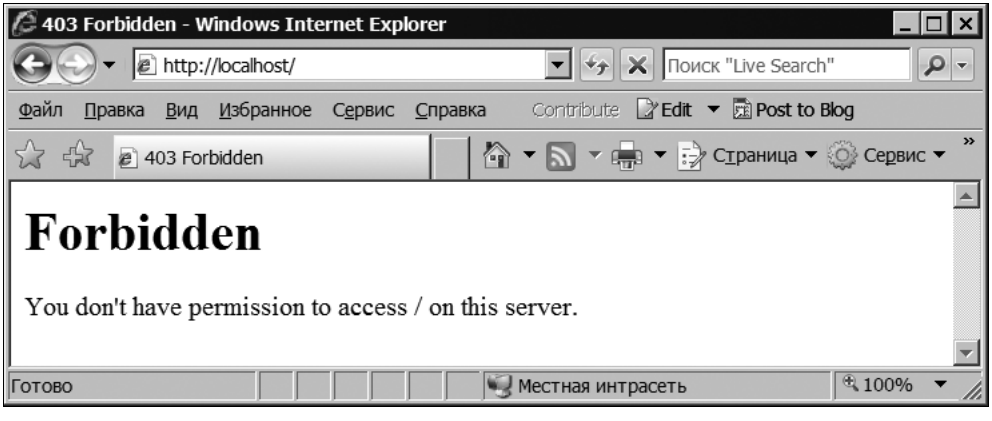


Рис. 2.22. Доступ к каталогу запрещен, если в нем отсутствует индексный файл

Однако можно изменить поведение Web-сервера Apache, если в качестве значения директивы `Options` передать значения директивы `Indexes` и `MultiViews` — в этом случае, если индексный файл отсутствует в текущем каталоге, будет выводиться содержимое данного каталога (рис. 2.23).

Внешний вид страницы, которая отображает содержимое текущего каталога, можно изменять при помощи конфигурационного файла .htaccess.

> ***ЗАМЕЧАНИЕ***
>
> Для применения описанных в данном разделе директив необходимо, чтобы директива `AllowOverride` в конфигурационном файле httpd.conf имела значение `Indexes`.

Директивы `HeaderName` и `ReadmeName` позволяют задать имена файлов, в которых размещается HTML-код шапки страницы и ее завершения соответственно. Создадим два файла, up.html и down.html, содержимое которых представлено в листингах 2.34 и 2.35.

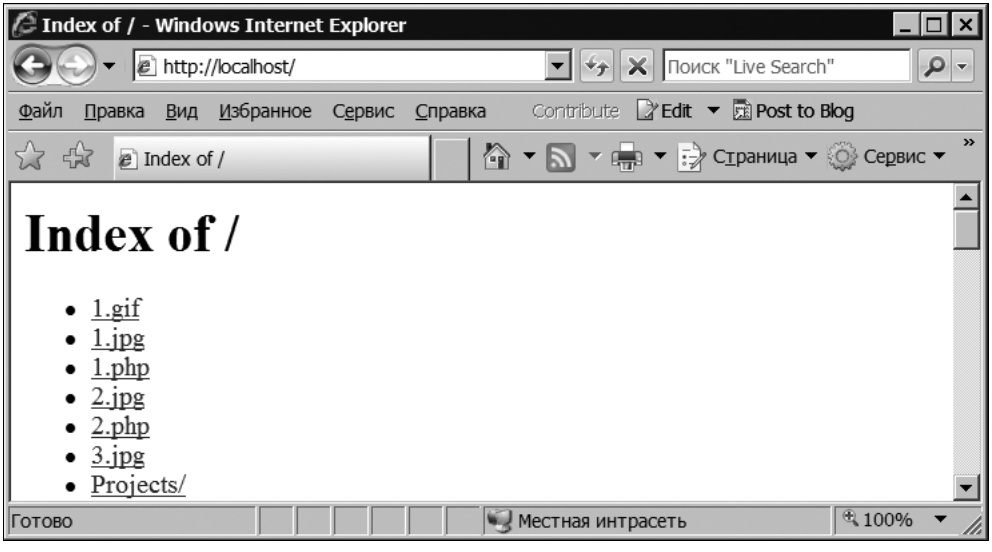


Рис. 2.23. Вывод содержимого текущего каталога

Листинг 2.34. Содержимое файла up.html

```
<h3 style="color: red">Текст шапки страницы</h3>
```

Листинг 2.35. Содержимое файла `down.html`

```
<h3 style="color: red">Текст завершения страницы</h3>
```

Теперь данные файлы можно вывести в начале и конце списка файлов, как это представлено на рис. 2.24.

Для того чтобы добиться этого эффекта, необходимо создать файл .htaccess в каталоге index с содержимым, представленным в листинге 2.36.

Листинг 2.36. Назначения шапки и завершения страницы

```
AddDefaultCharset Windows-1251
HeaderName up.html
ReadmeName down.html
```

Однако, как видно из рис. 2.24, файлы down.html и up.html, выполняющие вспомогательную функцию, также отображаются в списке файлов каталога. Это может быть не совсем удобным. Поэтому для скрытия вспомогательных файлов в списке содержимого каталога предусмотрена специальная директива `IndexIgnore`. Файл .htaccess с содержимым, представленным в листинге 2.37, позволяет скрыть файлы down.html и up.html.

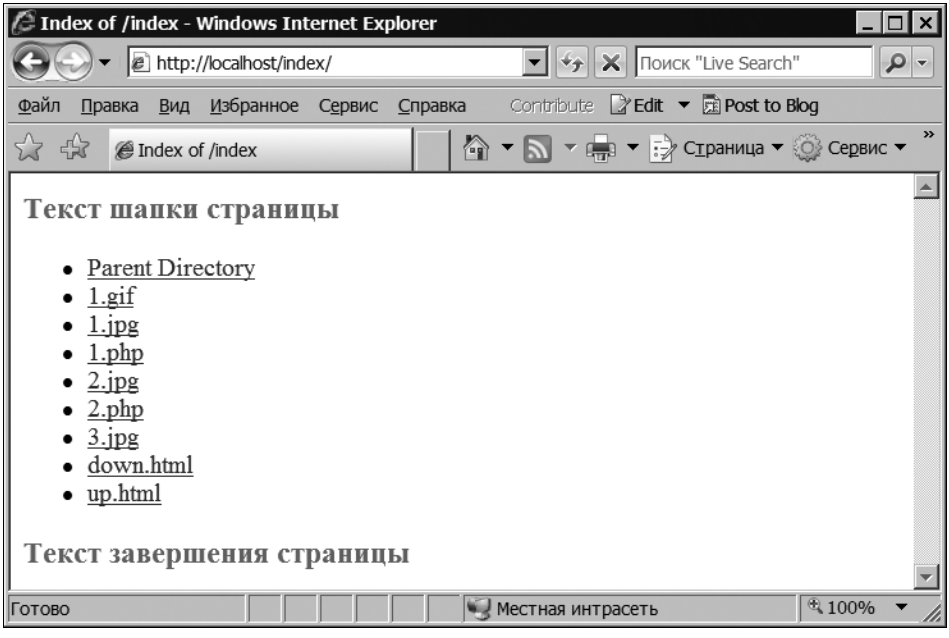


Рис. 2.24. Изменение шапки и завершения страницы
со списком файлов текущего каталога

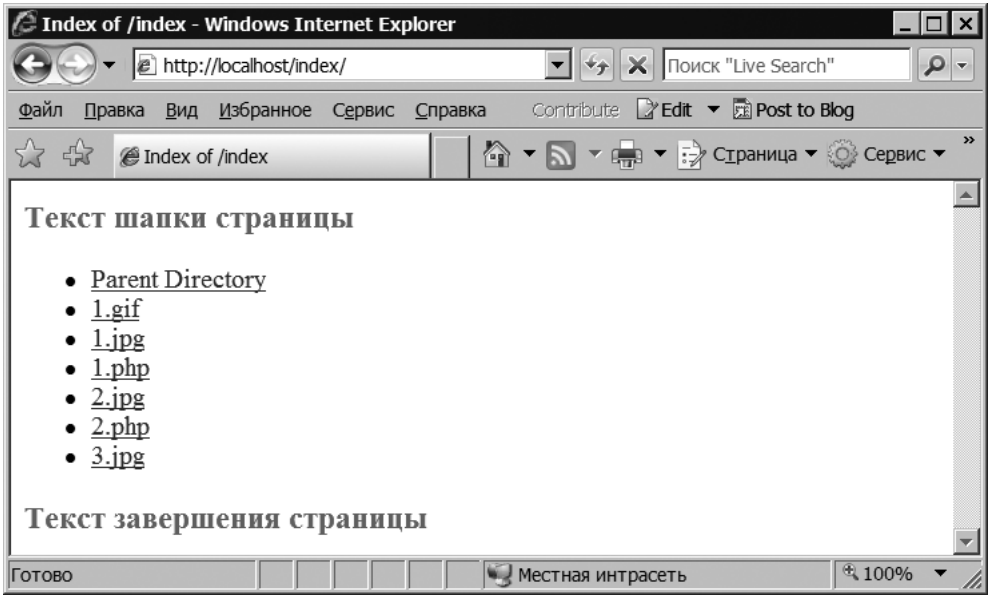


Рис. 2.25. Сокрытие вспомогательных файлов
при помощи конфигурационного файла .htaccess

Листинг 2.37. Сокрытие вспомогательных файлов

```
AddDefaultCharset Windows-1251
HeaderName up.html
ReadmeName down.html
IndexIgnore up.html down.html
```

Результат использования файла .htaccess из листинга 2.37 можно видеть на рис. 2.25.

2.2.4. Выполнение PHP-кода в HTML-файлах

Задача выполнения PHP-кода в HTML-файлах связана с переходом к динамическому представлению, когда в сайт, построенный на статических HTML-страницах, добавляются динамические элементы на PHP. Менять расширения файлов не всегда приемлемо, т. к. ссылки на страницы из поисковых систем и других сайтов в этом случае потеряют актуальность, да и над HTML-кодом самого сайта придется проделать изрядную работу. В этом случае прибегают к назначению в качестве обработчика HTML-файлов интерпретатора PHP. Достаточно добавить в каталог с HTML-файлами конфигурационный файл .htaccess с содержимым, представленным в листинге 2.38, и в HTML-файлах можно будет выполнять PHP-код так, как если бы файлы имели расширение php.

Листинг 2.38. Выполнение PHP-кода в HTML-файлах

```
RemoveHandler .html .htm
AddType application/x-httpd-php .php .htm .html .phtml
```

2.2.5. Страницы ошибок Web-сервера Apache

При обращении клиента к Web-серверу ему возвращается код состояния HTTP-заголовка HTTP/1.1. Коды состояния разделяются на классы, каждый из которых начинается с новой сотни:

- 100–199 — информационные коды состояния. Эти коды состояния сообщают клиенту, что сервер пребывает в процессе обработки запроса. От клиента не требуется реакция на данные коды;
- 200–299 — коды успешного выполнения запроса. Если клиент получает данный код ответа на запрос, это означает, что запрос успешно выполнен, и в теле присланного документа находится запрашиваемый документ, который можно передать пользователю;
- 300–399 — коды переадресации. Данные коды предназначены для того, чтобы дать понять клиенту, что для завершения запроса необходимо выполнить дальнейшие действия (как правило, перейти по новому адресу);
- 400–499 — коды ошибочного запроса. Данные коды отправляются клиенту, если сервер не может обработать его запрос;

- 500–599 — коды ошибок сервера. Данный тип ошибок появляется в том случае, если ошибка возникает по вине сервера (как правило, из-за синтаксической ошибки в файле .htaccess).

Если код HTTP-код является кодом 4xx или 5xx, браузер выводит стандартное сообщение о возникновении ошибки. Так на рис. 2.26 приводится окно браузера Internet Explorer при возникновении ошибки 404 — невозможно найти запрашиваемый документ.

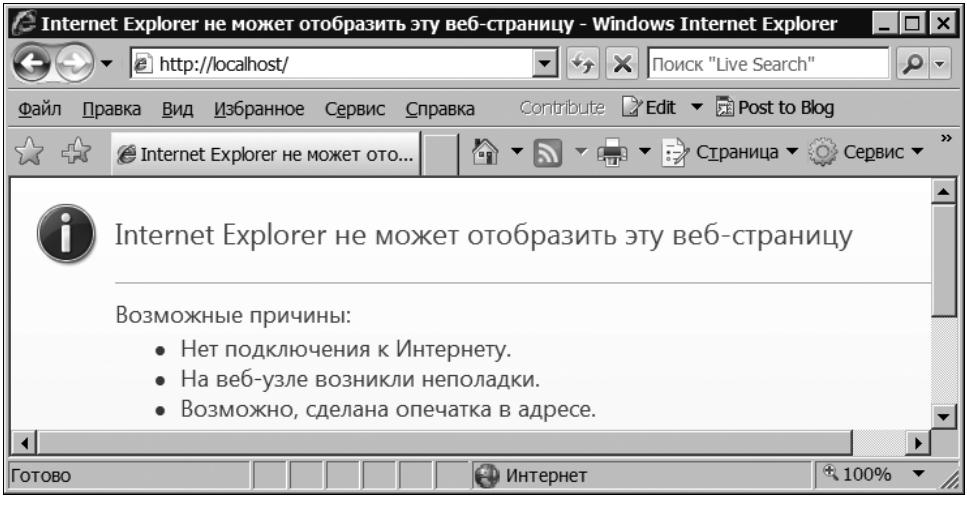


Рис. 2.26. Стандартное окно при возврате сервером HTTP-кода 404

Такое окно выбивается из дизайна сайта и, что самое главное, не предоставляет пользователю ссылок на страницы, откуда он может продолжить работу с сайтом. Для того чтобы предотвратить такое поведение, можно переопределить страницу ошибок для сервера, виртуального хоста или каталога. Для этого используется директива `ErrorDocument`, сразу после которой указывается HTTP-код, а затем действие, которое необходимо совершить. В качестве такого действия может быть вывод сообщения, загрузка файла обработчика ситуации по относительному или сетевому пути (листинг 2.39).

> ***Замечание***
>
> Для использования директивы `ErrorDocument` в конфигурационном файле .htaccess директива `AllowOverride` должна принимать значение `FileInfo`.

Листинг 2.39. Использование директивы `ErrorDocument`

```
ErrorDocument 404 "К сожалению, документ не может быть загружен, %s"
ErrorDocument 403 /errors/403.html
ErrorDocument 401 http://www.mysite.ru/index.php
```

Используя сетевой путь в качестве обработчика, можно даже направить клиента на совершенно другой сайт (это невозможно лишь для HTTP-кода 401).

2.2.6. Переадресация

Для переадресации пользователя предназначена директива `Redirect`, которая при обращении к адресу, указанному в первом аргументе, осуществляет редирект на ресурс, указанный во втором аргументе. Так при использовании конфигурационного файла .htaccess из листинга 2.40 обращение по адресу **http://localhost/index.php** будет приводить к переходу на сайт **http://www.softtime.ru/index.php**.

> ***ЗАМЕЧАНИЕ***
>
> Следует отметить, что переход осуществляется не сервером, а браузером после того, как ему приходит от сервера HTTP-код 3xx.

Листинг 2.40. Использование директивы `Redirect`

```
Redirect / http://www.softtime.ru/
```

По умолчанию директива `Redirect` посылает клиенту HTTP-код состояния 302: запрашиваемый ресурс временно перемещен. Управлять HTTP-кодом можно при помощи ключевых слов из табл. 2.21, размещаемых сразу после директивы `Redirect`.

Таблица 2.21. *Ключевые слова директивы* `Redirect`

Ключевое слово	HTTP-код состояния	Описание
`Permanent`	301	HTTP-код сообщает клиенту, что переадресация является постоянной
`Temp`	302	HTTP-код сообщает клиенту, что переадресация является временной (используется по умолчанию)
`See other`	303	HTTP-код сообщает клиенту, что ресурс был замещен
`Gone`	410	HTTP-код сообщает клиенту, что ресурс был удален

Так для того чтобы сообщить клиенту, что переадресация является постоянной, а не временной, достаточно добавить ключевое слово `Permanent` (листинг 2.41).

Листинг 2.41. Использование ключевого слова `Permanent`

```
Redirect Permanent / http://www.softtime.org/
```

Вместо ключевого слова также допускается использование непосредственно HTTP-кода (листинг 2.42).

Листинг 2.42. Использование HTTP-кодов переадресации (3*xx*)

```
Redirect 301 / http://www.softtime.org/
```

Директивы в листингах 2.41 и 2.42 являются эквивалентными. Так как HTTP-коды переадресации 301 и 302 используются часто, синтаксис конфигурационного файла

.htaccess допускает указание директив `RedirectTemp` и `RedirectPermanent`, которые эквивалентны директиве `Redirect` с ключевыми словами `Temp` и `Permanent`.

2.2.7. Запрет на доступ к ресурсу

Еще одной распространенной задачей является запрет загрузки файлов через браузер, ограничение доступа к служебным текстовым файлам, а также ограничение доступа к файлам, для которых необходимо собирать статистику загрузки, или распространяемым на коммерческой основе.

Для управления доступом и запретом обращения к тем или иным ресурсам служат директивы `allow` и `deny` соответственно. Конфигурационные файлы .htaccess и httpd.conf могут содержать несколько директив `deny` и `allow`, порядок их выполнения определяется директивой `order`:

```
order deny,allow
```

Сначала выполняются все запрещающие (`deny`) директивы и лишь затем директивы, разрешающие доступ (`allow`). Изменение порядка следования директив приводит к смене порядка их выполнения:

```
order allow,deny
```

В листинге 2.43 приводится пример конфигурационного файла .htaccess, который разрешает доступ к ресурсу только пользователям с IP-адресом 127.0.0.1.

Листинг 2.43. Разрешение доступа с IP-адреса 127.0.0.1

```
order deny,allow
deny from all
allow from 127.0.0.1
```

Сначала при помощи директивы `order` задается порядок применения директив `deny` и `allow`. Затем при помощи директивы `deny from all` запрещается обращение со всех IP-адресов. После чего для единственного IP-адреса 127.0.0.1 открывается доступ при помощи директивы `allow from 127.0.0.1`. В работоспособности данного механизма можно легко удостовериться, если испытания происходят на локальной машине: достаточно исправить IP-адрес в конфигурационном файле .htaccess с 127.0.0.1 на 127.0.0.2. Результатом этого будет возврат сервером HTTP-кода состояния 403 — доступ запрещен (рис. 2.27).

Если требуется, наоборот, запретить доступ с определенного IP-адреса, можно воспользоваться директивой, представленной в листинге 2.44.

Листинг 2.44. Запрет на доступ с IP-адреса 127.0.0.1

```
deny from 127.0.0.1
```

Вместо целых IP-адресов можно использовать частичные IP-адреса, указывая лишь начальный фрагмент IP-адреса (листинг 2.45).

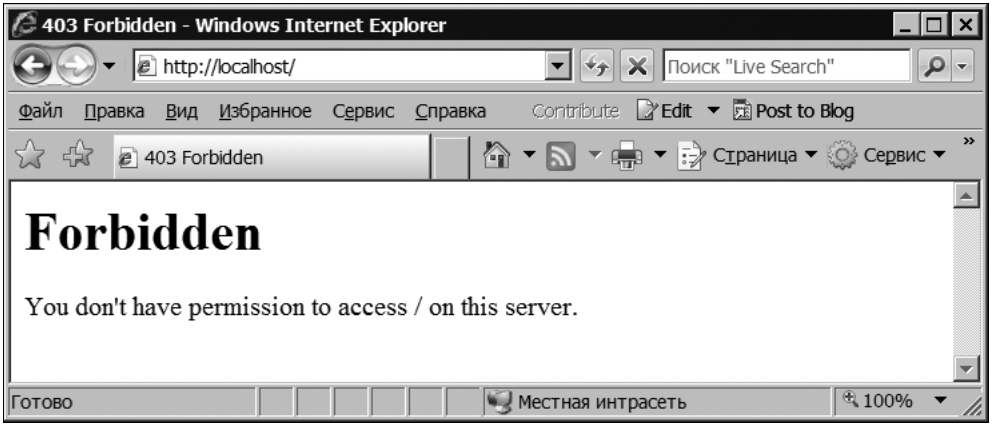


Рис. 2.27. Доступ запрещен

Листинг 2.45. Запрет на доступ с адресов из диапазона 127.0.0.1–127.0.0.255

```
deny from 127.0.0
```

Допускается также указывать маску подсети, если сеть не занимает весь сегмент. Директивы в листингах 2.45 и 2.46 эквивалентны.

Листинг 2.46. Использование маски подсети для запрета доступа

```
deny from 127.0.0.1/255.255.255.0
```

Кроме того, маску подсети можно задавать в CIDR-нотации (листинг 2.47).

Листинг 2.47. Использование CIDR-нотации для маски подсети

```
deny from 127.0.0.1/24
```

Не составит труда также полностью закрыть доступ к каталогам, в которых хранится ценная информация, при помощи директивы, представленной в листинге 2.48.

Листинг 2.48. Полный запрет доступа к каталогу

```
deny from all
```

Важно отметить, что доступ к файлам с виртуального хоста, например, из PHP-кода, по-прежнему возможен, т. к. файлы .htaccess действуют только на сетевой доступ к информации в каталоге.

2.2.8. Запрет загрузки файлов

В предыдущем разделе рассматривалось управлением доступа к каталогам. Однако чаще возникает задача запрета доступа к файлам с определенным расширением, например, расширением txt. Для этого прибегают к контейнеру `<Files>` (листинг 2.49).

Листинг 2.49. Использование контейнера `<Files>`

```
<Files "*.txt">
  deny from all
</Files>
```

В контейнере `<Files>` можно использовать также и имена файлов. Например, для запрета доступа из браузера к файлу config.php можно воспользоваться конфигурационным файлом .htaccess, содержимое которого представлено в листинге 2.50.

Листинг 2.50. Запрет доступа к файлу config.php

```
<Files "config.php">
  deny from all
</Files>
```

Запрет на доступ можно осуществлять также с применением регулярных выражений. Содержимое в листинге 2.51 эквивалентно по результату содержимому из листинга 2.50.

Листинг 2.51. Запрет на непосредственный доступ к файлам

```
<Files ~"*.php$">
  deny from all
</Files>
```

2.2.9. Защита сайта паролем

Распространенной задачей является защита внутренних областей сайта паролем. Для этого, помимо конфигурационного файла .htaccess, потребуется файл .htpasswd для хранения паролей. Создание файла .htpasswd осуществляется при помощи утилиты `htpasswd`, которая хранится в каталоге bin Web-сервера Apache.

Замечание

В качестве имени файла .htpasswd можно выбрать любое другое имя. Однако следует иметь в виду, что файлы, начинающиеся с последовательности, отличной от .ht, будут доступны для просмотра через браузер, и об их сокрытии придется позаботиться самостоятельно.

Для создания нового файла утилите необходимо передать параметр -c, помимо этого потребуется зашифровать пароль методом MD5 при помощи параметра -m. Параметры можно группировать, т. е. использовать -cm (листинг 2.52).

Листинг 2.52. Создание файла .htpasswd

```
htpasswd -cm .htpasswd user
```

Далее утилита запрашивает пароль для пользователя user и создает новый файл .htpasswd. Для того чтобы пароль password можно было задать непосредственно в командной строке, в параметры следует добавить параметр -b (листинг 2.53).

Листинг 2.53. Альтернативный способ создания файла .htpasswd

```
htpasswd -cmb .htpasswd user password
```

В результате выполнения команды из листинга 2.53 будет создан файл со следующим содержимым:

```
user:$apr1$8U2.....$Jehmbz8MGSSzxKQZxHUEs/
```

Каждая строка в файле соответствует одной учетной записи. Если в дальнейшем в уже существующий файл .htpasswd потребуется добавить еще одного пользователя, то параметр -c указывать не следует, т. к. существующие пользователи в файле будут затерты.

Теперь, после того как файл с учетными записями создан, следует создать файл .htaccess с содержимым, представленным в листинге 2.54.

> ***Замечание***
>
> Путь к файлу .htpasswd в конфигурационном файле .htaccess должен быть абсолютным. Относительные пути не допускаются. Если путь содержит пробелы, его необходимо заключать в кавычки.

Листинг 2.54. Защита паролем при помощи файла .htaccess

```
AuthType Basic
AuthName "Password Access"
AuthUserFile /путь/к/файлу/.htpasswd
require valid-user
```

Теперь при доступе к странице, защищенной при помощи конфигурационных файлов .htaccess и .htpasswd, будет выводиться окно с требованием ввести имя пользователя и пароль (рис. 2.28).

Следует отметить, что предоставлять доступ по паролю можно не только к каталогу в целом, но и к файлам с определенным расширением или именем. Для этого необходимо воспользоваться контейнером <Files> (листинг 2.55).

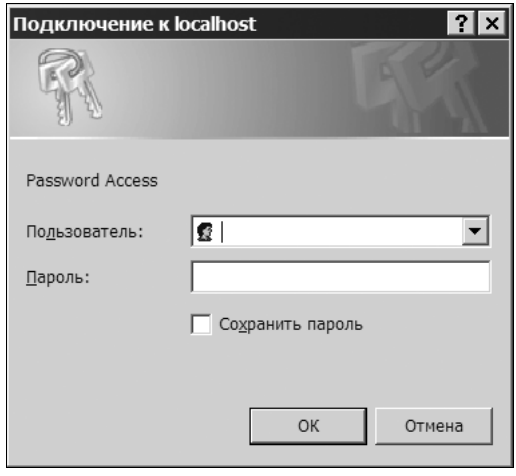


Рис. 2.28. Окно с требованием ввести имя пользователя и пароль

Листинг 2.55. Защита паролем архивных файлов

```
<Files "*.zip">
  AuthType Basic
  AuthName "Password Access"
  AuthUserFile /путь/к/файлу/.htpasswd
  require valid-user
</Files>
```

2.2.10. Преобразование URL-адресов

Одной из наиболее востребованных возможностей Web-сервера Apache является преобразование адресов. Адрес страницы является визитной карточкой: на нее обращают внимание и роботы поисковых систем, и обычные пользователи. CGI-формат (**http://site.dev?category=news&year=2011&month=12&day=27&page=1**) может производить удручающее впечатление даже на профессионала, в то время как прозрачный путь, оформленный в виде подкаталогов, может модифицировать даже неискушенный пользователь (**http://site.dev/news/2011/12/27/**). Для получения удобочитаемых адресов вовсе не обязательно создавать сложную систему вложенных папок, добиться преобразования GET-параметров можно при помощи модуля mod_rewrite.

2.2.10.1. Настройка модуля *mod_rewrite*

Сервер Apache является модульным приложением. Это означает, что он допускает динамическое подключение модулей, расширяющих его функциональность. Одним из таких модулей является модуль `mod_rewrite`, который позволяет задать псевдонимы для URL-адресов. Для его подключения в конфигурационном файле httpd.conf следует снять комментарий (`#`) напротив строки:

```
LoadModule rewrite_module modules/mod_rewrite.so
```

А директива `Options` контейнера `<Directory />` должна принимать значения:

```
Options Indexes FollowSymLinks MultiViews
```

После этого можно составлять правила в конфигурационном файле .htaccess для модификации URL.

2.2.10.2. Преобразование GET-параметра в подкаталог

Рассмотрим простейший случай, пусть имеется URL вида **http://localhost/index.php?id=1**, который необходимо преобразовать в **http://localhost/1/**. Для осуществления этой задачи необходимо в конфигурационный файл .htaccess поместить директивы, представленные в листинге 2.56.

> ***ЗАМЕЧАНИЕ***
>
> Существует мнение, что роботы поисковых систем плохо индексируют динамические страницы с параметрами **http://localhost/index.php?id=1**, считая их за одну страницу **http://localhost/index.php**, и для более успешной индексации сайта следует преобразовывать их адреса следующим образом: **http://localhost/1/**. Это не так. Современные поисковые роботы успешно обрабатывают страницы с параметрами, и основное назначение модуля `mod_rewrite` — обеспечивать более "красивые" и понятные адреса страниц.

Листинг 2.56. Преобразование адреса

```
RewriteEngine On
RewriteBase /
RewriteRule ^([0-9]+) /index.php?id=$1
```

Теперь при обращении к адресу **http://localhost/1/** или **http://localhost/54/** в окно браузера будет загружаться содержимое страниц **http://localhost/index.php?id=1** и **http://localhost/index.php?id=54** соответственно. Очень часто такой прием используется для сокрытия истинного расположения файлов на сервере. Убедиться в работоспособности конфигурационного файла можно, создав в корне виртуального хоста локального сервера файл index.php следующего содержания (листинг 2.57).

Листинг 2.57. Проверочный файл index.php

```
<?php
  echo $_GET['id'];
?>
```

Рассмотрим подробнее директивы, представленные в листинге 2.57. Первая директива `RewriteEngine` включает (`On`) механизм преобразования адресов, т. к. по умолчанию он отключен (это связано с тем, что работа модуля `mod_rewrite` создает дополнительную нагрузку на Web-сервер). Поэтому если имеется возможность избежать использования модуля `mod_rewrite`, ею следует воспользоваться. Однако если преобразование адресов необходимо, директиву `RewriteEngine On` следует помещать в начало любого файла, содержащего директивы `mod_rewrite`, в противном

случае они будут игнорироваться (или клиенту будет возвращаться ошибка серии 5xx — неправильная конфигурация сервера).

Директива `RewriteBase` позволяет указать базовый путь, начиная с которого будет осуществляться поиск соответствия регулярному выражению. В листинге 2.57 в качестве базового URL указывается корень виртуального хоста.

Последняя директива `RewriteRule` задает правило преобразования адреса. Первый аргумент директивы определяет регулярное выражение, соответствие которому ищется в каждом адресе, поступающем Web-серверу от клиента. Второй аргумент определяет адрес, который необходимо загрузить.

При помощи последовательности `$1` можно подставить фрагмент регулярного выражения в круглых скобках. Именно благодаря этому, какое бы число ни передавалось в адресе **http://localhost/nnn/**, оно будет аргументом параметра `id`: **http://localhost/index.php?id=nnn**. При помощи последовательности `$2` можно задать соответствие вторым круглым скобкам в регулярном выражении, при помощи `$3` — третьим и т. д.

Регулярные выражения позволяют задать шаблоны, при помощи метасимволов, имеющих специальное значение (табл. 2.22).

ЗАМЕЧАНИЕ

Регулярные выражения являются мини-языком, и их изучение требует немало усилий. Кроме того, практически невозможно изучить регулярные выражения без практического применения. На сегодняшний день наиболее полно регулярные выражения рассматриваются в книге Дж. Фридла "Регулярные выражения". Если составление регулярных выражений или использование тех или иных конструкций вызывает у вас сложности, вы всегда можете обратиться за помощью на наш форум по регулярным выражениям **http://www.softtime.ru/forum/index.php?id_forum=6**.

Таблица 2.22. *Метасимволы регулярных выражений*

Метасимвол	Описание
`.`	Соответствует любому одному символу
`[0-9]`	Соответствует одной цифре от 0 до 9
`[A-Z]`	Соответствует одному любому символу английского алфавита
`[0-9A-Z]`	Соответствует одному любому символу, являющемуся либо цифрой, либо буквой английского алфавита
`^`	Соответствует началу строки
`$`	Соответствует концу строки
`\b`	Соответствует границе слова
`\w`	Соответствует алфавитно-цифровому символу
`\s`	Соответствует пробельному символу (пробел, перевод строки, символ табуляции и т. п.)
`\t`	Символ табуляции

Таблица 2.22 *(окончание)*

Метасимвол	Описание
\n	Символ перевода строки
\r	Символ возврата каретки
\077	Восьмеричный символ
\x9f	Шестнадцатеричный символ
*	Соответствует 0 или большему числу предыдущих символов
+	Соответствует 1 или большему числу предыдущих символов
?	Соответствует 1 или 0 вхождениям предыдущих символов
{*n*}	Соответствует точно *n* вхождениям предыдущих символов
{*n*,}	Соответствует *n* или более вхождениям предыдущих символов
{*n*,*m*}	Соответствует от *n* до *m* вхождениям предыдущих символов

Символ ^ позволяет явно указать начало строки, в противном случае соответствие может быть найдено в середине строки. Например, регулярному выражению из листинга 2.56 не будет соответствовать адрес **http://localhost/abc/14/**, однако если убрать символ привязки к началу строки ^ (листинг 2.58), то регулярное выражение будет соответствовать этому адресу и передаст в качестве параметра id число 14.

Листинг 2.58. Отсутствие привязки к началу слова

```
RewriteEngine On
RewriteBase /
RewriteRule ([0-9]+) /index.php?id=$1
```

Более того, регулярному выражению из листинга 2.59 будет соответствовать любой адрес, в состав которого входят цифры, например, при запросе **http://localhost/abc/256/14/** браузер загрузит страницу **http://localhost/index.php?id=256**. То есть в качестве параметра id будет передаваться любая первая цифра от начала адреса.

Символ $, напротив, позволяет осуществить привязку к концу строки. Например, для адреса **http://localhost/abc/256/14** директивы из листинга 2.59 приведут к загрузке содержимого страницы **http://localhost/index.php?id=14**. То есть в качестве параметра id будет любая первая цифра от конца адреса.

Листинг 2.59. Привязка к концу строки

```
RewriteEngine On
RewriteBase /
RewriteRule ([0-9]+)/$ /index.php?id=$1
```

Для задания класса символов используются квадратные скобки. Они ограничивают поиск теми символами, которые в них заключены:

```
[abc]
```

Этому регулярному выражению соответствует подстрока, содержащая один символ: либо `a`, либо `b`, либо `c`.

Так для создания регулярного выражения, соответствующего всем буквам английского алфавита, можно, конечно, перечислить все эти буквы в квадратных скобках. Это допустимо, но утомительно и неэлегантно. Более коротко такое регулярное выражение можно записать следующим образом:

```
[a-z]
```

Это выражение соответствует всем буквам английского алфавита, поскольку любые два символа, разделяемые дефисом, задают соответствие диапазону символов, находящихся между ними.

Точно таким же образом задаются регулярные выражения, соответствующие символу числа:

```
[0-9]
```

или

```
[0123456789]
```

Оба этих выражения эквивалентны и соответствуют любой цифре.

Квадратные скобки позволяют определить класс символов, но в регулярном выражении они соответствуют строго одному символу. Чтобы ввести обозначение для нескольких символов, используют квантификаторы `?`, `+` и `*`, которые следуют сразу за символом и изменяют количество вхождений этого символа в строку:

- `?` — символ либо входит в строку один раз, либо вообще в нее не входит;
- `*` — любое количество вхождений символа в строку, в том числе и 0;
- `+` — одно или более число вхождений символа в строку.

Помимо круглых и квадратных скобок в регулярных выражениях также применяются фигурные скобки. Они предназначены для указания числа или диапазона чисел повторения элемента:

- `xy{2}` соответствует строке, в которой за `x` следует два символа `yy`;
- `xy{2,}` соответствует строке, в которой за `x` следует не менее двух `y` (может быть и больше);
- `xy{2,6}` соответствует строке, в которой за `x` следует от двух до шести `y`.

Для указания количества вхождений не одного символа, а их последовательности, применяются круглые скобки:

- `x(yz){2,6}` соответствует строке, в которой за `x` следует от двух до шести последовательностей `yz`;

- `x(yz)*` соответствует строке, в которой за `x` следует ноль и более последовательностей `yz`.

Помимо рассмотренных метасимволов регулярные выражения допускают использование еще одного символа — вертикальной черты `|` для задания альтернативных масок.

```
abc|абв
```

Этому регулярному выражению соответствует любая строка, содержащая подстроки `abc` или `абв`. Вертикальную черту удобно применять при проверке расширений и имен файлов, зон доменных имен и т. д. К примеру, следующее регулярное выражение проверяет, содержатся ли в строке подстроки "`ru`", "`com`" или "`net`":

```
ru|com|net
```

В конфигурационном файле .htaccess может быть несколько директив `RewriteRule`, определяющих различные преобразования адреса.

2.2.10.3. Преобразование адреса к виду 2006/07/20

Рассмотрим регулярное выражение преобразования адреса **http://localhost/2006/07/20/** в **http://localhost/index.php?year=2006&month=07&day=20**. Такое преобразование часто требуется в новостных блоках, блогах и любых блоках, связанных с календарной последовательностью сообщений. Для решения задачи можно воспользоваться конфигурационным файлом .htaccess следующего содержания (листинг 2.60).

> ***Замечание***
>
> Несмотря на то, что в листинге 2.60 директива `RewriteRule` не поместилась в одной строке, в реальных файлах .htaccess такого быть не должно. Каждой директиве отводится лишь одна строка.

Листинг 2.60. Преобразование даты YYYY/MM/DD в GET-параметры

```
RewriteEngine On
RewriteBase /
RewriteRule ^([0-9]{4})/([0-9]{1,2})/([0-9]{1,2})/$
/index.php?year=$1&month=$2&day=$3
```

Для проверки работоспособности правила из листинга 2.60 можно использовать файл index.php следующего содержания (листинг 2.61).

Листинг 2.61. Проверочный файл index.php

```
<?php
  echo "Год — ".$_GET['year']."<br>";
  echo "Месяц — ".$_GET['month']."<br>";
  echo "День — ".$_GET['day']."<br>";
?>
```

Как видно из листинга 2.61, в регулярном выражении, передаваемом в качестве первого аргумента директивы `RewriteRule`, три круглых скобки соответствуют году, месяцу и дню и обозначаются во втором аргументе соответственно `$1`, `$2` и `$3`. Кроме этого, в регулярном выражении используются фигурные скобки, так фрагмент `[0-9]{4}` соответствует четырем цифрам и обозначает год, т. е. правило не сработает, если вместо адреса **http://localhost/2006/07/20/** будет задан адрес **http://localhost/06/07/20/**. Фрагмент `[0-9]{1,2}` соответствует одной или нескольким цифрам, т. е. адреса **http://localhost/2006/07/20/** и **http://localhost/2006/7/20/** считаются допустимыми.

2.2.10.4. Экранирование мета-символов

В табл. 2.22 были приведены метасимволы, которые имеют специальное значение. Как же использовать эти символы, если в адрес входят символы точки или знака вопроса? Для этого символы подвергаются экранированию. Пусть имеется адрес **http://localhost/index.html/**, которому необходимо поставить в соответствие адрес **http://localhost/index.php**. В этом случае набор директив преобразования может выглядеть так, как это представлено в листинге 2.62.

Листинг 2.62. Использование исходных значений метасимволов

```
RewriteEngine On
RewriteBase /
RewriteRule ^index\.html /index.php
```

2.2.10.5. Флаги директивы *RewriteRule*

Директива `RewriteRule` может также принимать в качестве третьего аргумента специальные флаги, влияющие на правила преобразования. Флаги всегда помещаются в квадратные скобки, например, `[R]`. Если требуется указать более одного флага, они разделяются запятыми, например, `[L,R]`. Например, если заменить содержимое файла .htaccess из листинга 2.62 на содержимое листинга 2.63, то вместо загрузки содержимого index.php при обращении к странице index.html будет осуществляться редирект на страницу index.php.

Листинг 2.63. Использование флага [R]

```
RewriteEngine On
RewriteBase /
RewriteRule ^index\.html /index.php [R]
```

Все флаги директивы `RewriteRule` приводятся в табл. 2.23. Каждый флаг имеет две формы написания: краткую, например, `[R]`, и полную, например, `[redirect]`.

Таблица 2.23. *Флаги директивы* `RewriteRule`

Флаг	Описание
`C | chain`	Флаг сообщает, что текущее правило связано со следующим. Это означает, что следующее за текущим правило выполняется только тогда, когда было найдено соответствие первому правилу, иначе все последующие правила, снабженные флагом `C`, игнорируются
`E=`*`var`*`:`*`value`* `| env=`*`var`*`:`*`value`*	С помощью данного флага можно определить дополнительный `GET`-параметр с именем *`var`* и значением *`value`*
`F | forbidden`	Если найдено соответствие правилу с этим флагом, браузеру возвращается код состояния 403 (HTTP Forbidden)
`G | gone`	Если найдено соответствие правилу с этим флагом, браузеру возвращается код состояния 410 (HTTP Gone)
`L | last`	Если найдено соответствие правилу с этим флагом, все последующие правила, определяемые дополнительными директивами `RewriteRule`, игнорируются
`N | next`	При соответствии адресу обработка правил начинается с самой первой директивы `RewriteRule`. Однако поиск осуществляется не в исходном адресе, а уже в преобразованном. Следует быть крайне аккуратным с этим флагом, чтобы не допустить зацикливания
`NC | nocase`	Флаг сообщает, что при поиске соответствия регулярному выражению регистр не должен учитываться
`NS | nosubreq`	Флаг используется для подавления применения текущего правила к внутреннему запросу
`P | proxy`	При использовании данного флага запрос преобразуется в прокси-запрос
`QSA | qsappend`	Флаг позволяет присоединить данные (например, пары *`ключ=значение`*) к строке запроса
`R [=`*`HTTP-код`*`] | redirect`	Выполняет переадресацию на адрес, который ставится в соответствие регулярному выражению вместо того, чтобы загружать содержимое страницы по введенному адресу. Если не задан HTTP-код состояния, по умолчанию используется код 302 (Move Temporarily)
`S=`*`n`* `| skip=`*`n`*	Если найдено соответствие правилу с этим флагом, следующие *`n`* правил пропускаются
`T=`*`MIME-type`* `|`	Флаг устанавливает MIME-тип запроса

2.2.10.6. Несколько правил для переменного количества GET-параметров

Очень часто при разработке блоков с глубоковложенными разделами возникает задача преобразования адресов с различными наборами блоков.

```
http://site.dev/catalog.php
http://site.dev/catalog.php?id_catalog=12
http://site.dev/catalog.php?id_catalog=12&id_position=366
```

Первый адрес отображает индексную страницу со списком каталогов. Второй адрес ведет к странице каталога с идентификационным номером 12. Третий адрес ведет к товарной позиции с идентификационным номером 366 каталога 12. Преобразованные адреса могут выглядеть следующим образом:

```
http://site.dev/catalog/
http://site.dev/catalog/12/
http://site.dev/catalog/12/366/
```

Для каждого набора параметров потребуется отдельное правило, причем каждое из правил должно завершаться флагом `[L]`, предотвращая дальнейшее выполнение правил. В противном случае успешно найденное правило для товарной позиции может быть проигнорировано, т. к. модуль mod_rewrite обнаружит правило для каталога и воспользуется им. В листинге 2.64 приводится возможное решение данной задачи.

Листинг 2.64. Использование нескольких правил `RewriteRule`

```
RewriteEngine On
RewriteBase /
RewriteRule ^catalog/([0-9]+)/([0-9]+)/
/catalog.php?id_catalog=$1&id_position=$2 [L]
RewriteRule ^catalog/([0-9]+)/ /catalog.php?id_catalog=$1 [L]
RewriteRule ^catalog/ /catalog.php?id_catalog=$1 [L]
```

2.2.10.7. Отладка правил *mod_rewrite*

Отлаживать преобразования модуля mod_rewrite может быть утомительным занятием. Задача сильно облегчается, если для виртуального хоста включить запись в журнал всех преобразований. Для этого в контейнер `<VirtualHost>` следует поместить директивы (листинг 2.65), которые указывают путь к журналу и уровень подробности записей в нем (от 1 до 9, где 9 соответствует максимальному уровню подробности).

Листинг 2.65. Подключение журналирования модуля mod_rewrite

```
<VirtualHost 127.0.0.1:80>
  ...
  RewriteLog logs/rewrite.log
  RewriteLogLevel 9
  ...
</VirtualHost>
```

2.2.10.8. Дополнительные правила. Директива *RewriteCond*

Правилу преобразования при помощи директивы `RewriteRule` может предшествовать одно или несколько дополнительных условий, которые задаются при помощи

директивы RewriteCond. Пусть имеется правило, которое преобразует адреса новостного блока **http://site.dev/news/54/** в **http://site.dev/news.php?id=54** (листинг 2.66).

Листинг 2.66. Преобразование адреса

```
RewriteEngine On
RewriteBase /
RewriteRule ^news/([0-9]+)/$ news.php?id=$1 [L]
```

Одной из часто возникающих задач при использовании преобразованных адресов является передача преобразованным адресам дополнительных GET-параметров. Например, статья новости может быть большой по объему, и возникает задача разбиения статьи на несколько страниц. Номер страницы передается в дополнительном GET-параметре page, а адрес приобретает вид **http://site.dev/news/54/?page=2**. В данном случае для передачи нового GET-параметра можно было бы ввести дополнительный уровень вложенности, однако это возможно только в простейшем случае. Множественные правила, например, описанные в *разд. 2.2.10.6*, могут не позволить решить эту задачу корректно, и придется оставить CGI-формат **?page=2**. Решить задачу при помощи лишь директивы RewriteRule здесь не представляется возможным, т. к. эта директива отбрасывает все GET-параметры и работает лишь с участком адреса до знака вопроса. Директива RewriteCond позволяет применять регулярные выражения для переменных окружения, используя формат %{имя_переменной} (листинг 2.67).

Листинг 2.67. Использование GET-параметров в преобразованных адресах

```
RewriteEngine On
RewriteBase /
RewriteCond %{QUERY_STRING} page=([0-9]*)
RewriteRule ^news/([0-9]+)/ /news.php?id=$1&page=%1 [L]
RewriteRule ^news/([0-9]+)/$ news.php?id=$1 [L]
```

Правило RewriteRule, которому предшествует директива RewriteCond, выполняется только тогда, когда срабатывает условие в RewriteCond. Причем директиве RewriteRule может предшествовать несколько условий RewriteCond, в этом случае все они должны быть выполнены.

В листинге 2.67 условие RewriteCond использует переменную окружения QUERY_STRING, в которую Web-сервер Apache помещает содержимое адреса после знака вопроса. Регулярное выражение ищет среди GET-параметров числовой параметр с именем page, и если он обнаружен, передает управление директиве RewriteRule. Причем директива RewriteRule может использовать найденные соответствия в RewriteCond, ссылаясь на них при помощи последовательностей %1, %2, которые соответствуют первым, вторым круглым скобкам и т. д.

В табл. 2.24 приводится список наиболее часто используемых переменных окружения. Большинство этих переменных также доступны PHP-скриптам через суперглобальный массив `$_SERVER`.

Таблица 2.24. *Наиболее часто используемые переменные окружения Web-сервера Apache*

Переменная окружения	Значение
`HTTP_USER_AGENT`	Пользовательский агент (строка, отправляемая клиентом для идентификации) сообщает серверу об используемом браузере, операционной системе и их окружении
`HTTP_REFERER`	Ссылка на предыдущую страницу, с которой пользователь перешел на текущую
`HTTP_COOKIE`	Cookie-переменные, устанавливаемые на стороне клиента, через HTTP-заголовок Set-Cookie
`HTTP_HOST`	Доменное имя сайта, к которому обращается клиент. Сервер может обслуживать сотни сайтов, поэтому клиенты отправляют в HTTP-заголовке Host доменное имя, по которому сервер определяет, какой сайт отображать
`HTTP_ACCEPT`	Предпочтения пользователя в языке и форматах документа
`DOCUMENT_ROOT`	Путь к каталогу, в котором расположены файлы, ответственные за работу сайта
`SERVER_NAME`	Имя сервера, как правило, совпадает с HTTP_HOST
`SERVER_ADDR`	IP-адрес сервера
`REMOTE_ADDR`	IP-адрес клиента
`REQUEST_METHOD`	HTTP-метод, при помощи которого клиент обратился к серверу (как правило, GET, POST или HEAD)
`REQUEST_FILENAME`	Путь к файлу или каталогу, который запросил клиент
`QUERY_STRING`	Содержимое адреса после знака вопроса, где указываются GET-параметры

2.2.10.9. Проверка существования подкаталогов

Создавая адреса при помощи правила преобразования, можно экранировать реально существующие файл и каталоги. Это может произойти умышленно или случайно. В любом случае директива `RewriteCond` позволяет составить условия проверки существования файла или каталога на жестком диске, и в зависимости от результата принять решение, выполнять следующую за ней директиву `RewriteRule` или нет.

В листинге 2.68 две директивы `RewriteCond` проверяют наличие запрошенного файла (`!-f`) или каталога (`!-d`), и если таковые отсутствуют, передает управление директиве `RewriteRule`.

Листинг 2.68. Проверка существования файла

```
RewriteEngine On
RewriteBase /
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule new/(.*) new/index.php?run=$1 [L]
```

Как видно из листинга 2.68, вместо регулярного выражения в директиве `RewriteCond` используются флаги `-f` (файл существует) и `-d` (каталог существует), которые предваряет отрицание `!`.

2.2.10.10. Назначение блокам доменов третьего уровня

Современные сайты и порталы, как правило, состоят из большого количества блоков, для которых могут выделяться отдельные папки в структуре сайта. С другой стороны, подавляющее большинство сайтов располагается в доменах второго уровня. Поэтому часто разработчики связывают домен третьего уровня с конкретной папкой в структуре сайта.

> ***ЗАМЕЧАНИЕ***
>
> Напомним, что домен первого уровня располагается в самом конце адреса, например, .ru, домен второго уровня располагается перед доменом первого уровня, например, softtime.ru.

Например, путь site.dev/news может быть преобразован в news.site.dev, а site.dev/company — в company.site.dev. Одним из возможных путей такого назначения является использование правила `RewriteRule` совместно с условием `RewriteCond`. В листинге 2.69 приводится пример связывания домена третьего уровня news.test.dev с папкой домена второго уровня test.dev/news/.

> ***ЗАМЕЧАНИЕ***
>
> Для того чтобы правила сработали, домен третьего уровня должен быть прописан в DNS-правилах. В противном случае переменная окружения `SERVER_NAME` не будет заполняться, и правила не сработают.

Листинг 2.69. Управление доменами третьего уровня

```
RewriteEngine on
RewriteBase /
RewriteCond %{SERVER_NAME} ^news.test.dev$
RewriteRule index.php news/index.php [L]
```

2.3. MySQL

2.3.1. Работа с утилитой *mysql*

В *разд. 1.16* мы уже познакомились со стандартным MySQL-клиентом — утилитой mysql.

Для соединения с сервером базы данных в параметрах mysql необходимо указать имя пользователя и его пароль. В только что установленной системе существует один пользователь `root`, наделенный правами администратора, а его паролем по умолчанию выступает пустая строка. Поэтому для получения доступа к серверу достаточно набрать команду:

```
mysql –u root
```

Результат выполнения команды представлен на рис. 2.29. После номера версии сервера и информации о справочных ключах выводится приглашение `mysql>`, разрешающее ввод команд.

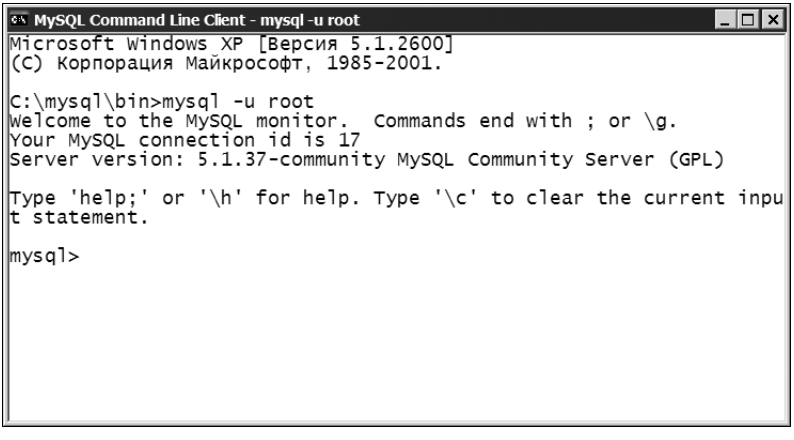


Рис. 2.29. Приглашение `mysql>` консольного клиента mysql

Для демонстрации полноценной авторизации создадим нового пользователя `user` с паролем `hello` при помощи оператора `CREATE USER`, представленного в листинге 2.70. Эту инструкцию следует набрать сразу после приглашения `mysql>`, как это показано на рис. 2.30.

Листинг 2.70. Создание нового пользователя `user`

```
CREATE USER user IDENTIFIED BY 'hello';
```

После этого выйдем из оболочки mysql и попробуем зайти от имени нового пользователя. Выход из оболочки mysql производится при помощи команды `exit` или `quit` (рис. 2.30).

Теперь команда `mysql –u user` не проходит — сервер отказывается предоставить доступ без подтверждения полномочий паролем (рис. 2.31). Для того чтобы соединиться с сервером с правами пользователя `user`, необходимо передать пароль. Для этого используется параметр `-p`, сразу после которого без пробела вводится пароль. Для удобства в других параметрах допускается использование пробела между параметром и его значением.

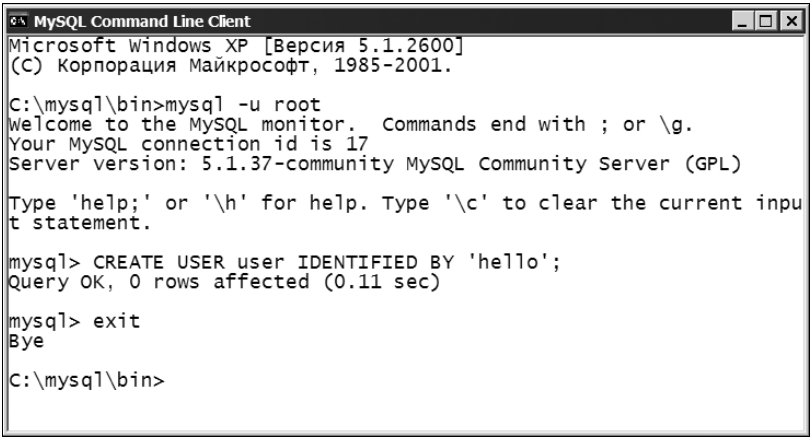


Рис. 2.30. Создание пользователя `user` с паролем `hello` и выход из оболочки

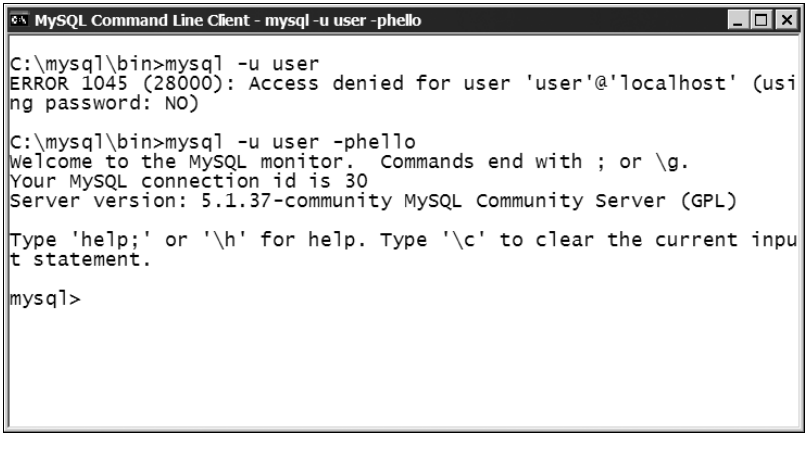


Рис. 2.31. Авторизация с использованием пароля

Например, следующие записи эквивалентны:

```
mysql –uuser –phello
mysql –u user –phello
```

Параметр `–p` является исключением, т. к. все следующие за ним символы воспринимаются как часть пароля.

В некоторых ситуациях требуется скрыть пароль символами звездочки, в этом случае допускается использование параметра `–p` без значения. Утилита mysql запросит пароль при помощи отдельной строки `Enter password:`, в которой вводимые символы будут скрыты символом звездочки (рис. 2.32).

Команды и SQL-инструкции за редким исключением (`exit`, `quit`, `use`) должны заканчиваться точкой с запятой. Например, на рис. 2.33 приведена SQL-инструкция, запрашивающая у сервера MySQL его версию и текущую дату.

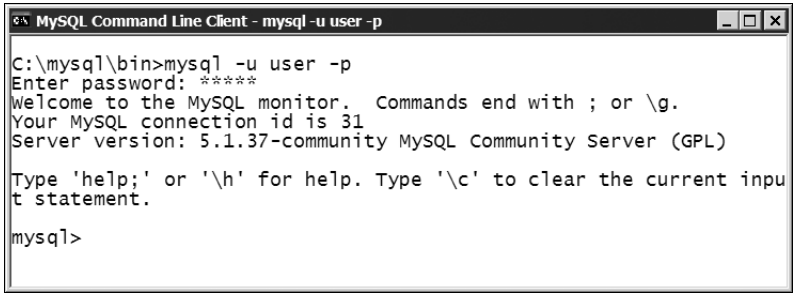


Рис. 2.32. Альтернативный способ авторизации

Замечание

Начиная с MySQL 5.0, изменить символ завершения запроса с точки с запятой на новый символ, например //, можно либо при помощи команды `DELIMITER //`, либо указав разделитель при помощи параметра `--delimiter=name`.

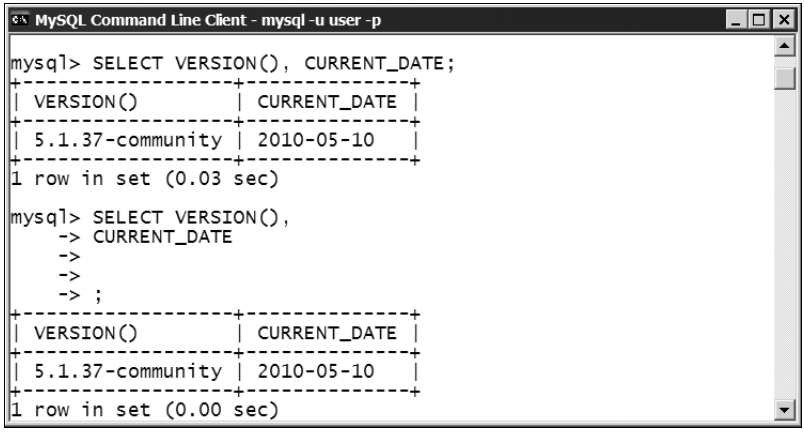


Рис. 2.33. Выполнение команд

После ввода пользователем команды она отправляется на сервер для выполнения, и, если нет ошибок в синтаксисе, на экран выводится результат в виде результирующей таблицы, а на новой строке — приглашение `mysql>`, разрешающее ввод следующей команды.

В первой строке таблицы с результатами содержатся заголовки столбцов, а в следующих строках — ответ сервера на запрос. Обычно заголовками столбцов становятся имена, полученные из таблиц базы. Если же извлекается не столбец таблицы, а значение выражения (как это происходит в приведенном примере), mysql дает столбцу имя запрашиваемого выражения. После этого сообщается количество возвращаемых строк (`1 row in set` — одна строка в результате) и время выполнения запроса.

Для ввода ключевых слов можно использовать любой регистр символов. Так приведенные в листинге 2.71 запросы идентичны.

Листинг 2.71. Имена инструкций и ключевые слова не зависят от регистра символов

```
SELECT VERSION(), CURRENT_DATE;
select version(), current_date;
SeLeCt vErSiOn(), current_DATE;
```

Если команда не помещается на одной строке, возможен переход на другую строку после нажатия клавиши <Enter> — запрос отправляется серверу только после того, как консольный клиент mysql встретит символ точки с запятой. Приглашение командной строки после ввода первой строки этого запроса меняется с `mysql>` на `->` (рис. 2.33). Таким образом, программа mysql показывает, что завершенного выражения она пока что не получила, и ожидает его полного ввода. Точно так же утилита mysql ведет себя, когда ожидает завершения строки, заключенной в двойные (`"`) или одинарные (`'`) кавычки (рис. 2.34).

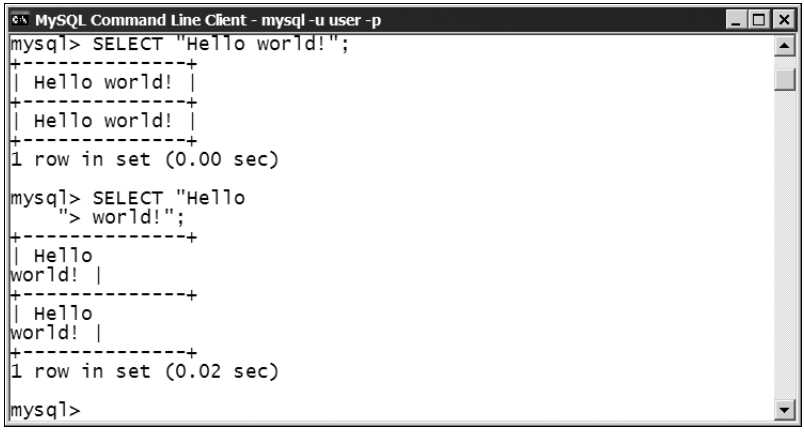


Рис. 2.34. Многострочный ввод текста, заключенного в двойные кавычки

Как видно из рис. 2.34, при переходе на следующую строку приглашение командной строки меняется с `mysql>` на `">`. Если строка заключена в одинарные кавычки, приглашение меняется на `'>`. Для того чтобы отменить текущий запрос, нужно ввести последовательность `\c`.

Уже введенные ранее команды не обязательно вводить снова, для этого достаточно их вызвать клавишами <↑> и <↓> (стрелка вверх и стрелка вниз); очистить строку запроса можно при помощи клавиши <Esc>.

2.3.2. Восстановление утерянного пароля

Если утерян пароль обычного пользователя, авторизовавшись с использованием учетной записи `root`, можно назначить пользователю пароль, как это представлено в листинге 2.72.

Листинг 2.72. Назначение нового пароля

```
SET PASSWORD FOR user = PASSWORD('password')
```

Если утерян пароль root, необходимо запустить сервер с параметром --skip-grant-tables (в конфигурационном файле my.ini следует добавить директиву skip-grant-tables в секции [mysqld]), в этом режиме сервер игнорирует таблицу привилегий и все базы данных (в том числе и системная база данных mysql) доступны без привилегий. В этом случае оператор SET PASSWORD можно выполнять из-под любого пользователя по отношению к любому другому пользователю, в том числе и root. По завершению восстановительных работ сервера следует перезагрузить MySQL без параметра --skip-grant-tables.

Замечание

Восстановить пароль можно также при помощи утилиты mysqladmin, выполнив команду mysqladmin -u root password "password".

2.3.3. Удаленный доступ к *MySQL*

Следует отметить, что учетная запись является составной и принимает форму *'username'@'host'*, где *username* — имя, а *host* — наименование хоста, с которого пользователю *username* разрешено обращаться к серверу MySQL. Так, учетные записи 'root'@'127.0.0.1' и 'wet'@'62.78.56.34' означают, что пользователь с именем root может обращаться с хоста, на котором расположен сервер, а wet — только с хоста с IP-адресом 62.78.56.34. Ни с какого другого хоста обратиться к серверу MySQL с этим именем нельзя, в том числе и с машины, где расположен сам сервер.

Замечание

Если имя пользователя и хост не содержат специальных символов — и %, их необязательно брать в кавычки — root@127.0.0.1.

Замечание

IP-адрес 127.0.0.1 всегда относится к локальному хосту — если сервер и клиент установлены на одном хосте и сеть не используется, то сервер слушает соединения по этому адресу, а клиент отправляет на него SQL-запросы. Во всех операционных системах IP-адрес 127.0.0.1 имеет псевдоним localhost, поэтому вместо IP-адреса можно указывать этот псевдоним, а учетные записи вида 'root'@'127.0.0.1' записывать так: 'root'@'localhost'.

Если к IP-адресу хоста привязано какое-то доменное имя, например 'softtime.ru', то вместо данного хоста можно использовать доменное имя 'root'@'softtime.ru' — такая учетная запись означает, что, зарегистрировавшись с именем root, можно обращаться к серверу с хоста, доменным именем которого является softtime.ru.

Если в дополнение к двум хостам 127.0.0.1 и 62.78.56.34 требуется разрешить пользователю 'wet' обращаться к серверу MySQL с третьего хоста 158.0.55.62, то потребуется создать три учетные записи: 'wet'@'127.0.0.1', 'wet'@'62.78.56.34' и

'wet'@'158.0.55.62'. Число адресов, с которых необходимо обеспечить доступ пользователю, может быть значительным и включать целые диапазоны. Для задания диапазона в имени хоста используется специальный символ %. Так, учетная запись 'wet'@'%' разрешает пользователю 'wet' обращаться к серверу MySQL с любых компьютеров сети. Символ % позволяет задавать диапазоны, поэтому учетная запись 'wet'@'%.softtime.ru' разрешает обращаться к серверу MySQL поддоменам домена softtime.ru. Учетная запись 'wet'@'62.78.56.%' позволяет пользователю 'wet' получить доступ с хостов, обладающих IP-адресами в диапазоне от 62.78.56.0 до 62.78.56.255.

Отсутствие части *host* в учетной записи аналогично использованию %, таким образом, учетные записи 'wet'@'%' и 'wet' эквивалентны.

2.3.4. Управление привилегиями пользователей

Оператор CREATE USER позволяет лишь создавать учетные записи, однако не разрешает изменять привилегии пользователя, т. е. сообщать СУБД MySQL, какой пользователь имеет право только на чтение информации, какой — на чтение и редактирование, а кому предоставлены права изменять структуру базы данных и создавать учетные записи для остальных пользователей.

Для решения этих задач предназначены операторы GRANT и REVOKE: оператор GRANT назначает привилегии пользователю, а REVOKE — удаляет. Если учетная запись, которая появляется в операторе GRANT, не существует, то она автоматически создается. Однако удаление всех привилегий с помощью оператора REVOKE не приводит к автоматическому уничтожению учетной записи — для полного удаления пользователя необходимо воспользоваться оператором DROP USER.

В простейшем случае оператор GRANT выглядит так, как это представлено в листинге 2.73.

Листинг 2.73. Использование оператора GRANT

```
GRANT ALL ON *.* TO 'wet'@'localhost' IDENTIFIED BY 'pass';
```

Запрос в листинге 2.73 создает пользователя с именем wet и паролем pass. Этот пользователь может обращаться к серверу MySQL с локального хоста (localhost) и имеет все права (ALL) для всех баз данных (*.*). Если такой пользователь уже существует, то его привилегии будут изменены на ALL.

> ***ЗАМЕЧАНИЕ***
>
> На данный момент оператор GRANT поддерживает имена хостов, баз данных, таблиц и столбцов размером до 60 символов. Имя пользователя не может быть длиннее 16 символов.

Отобрать права у пользователя 'wet'@'localhost' можно при помощи оператора REVOKE, представленного в листинге 2.74.

Листинг 2.74. Использование оператора REVOKE

```
REVOKE ALL ON *.* FROM 'wet'@'localhost';
```

Вместо ключевого слова ALL, которое обозначает все системные привилегии (кроме GRANT OPTION — передача привилегий другим пользователям), можно использовать любое из ключевых слов, представленных в табл. 2.25.

Таблица 2.25. *Системные привилегии, используемые в операторах GRANT и REVOKE*

Привилегия	Операция, разрешенная привилегией
ALL [PRIVILEGES]	Комбинация всех привилегий за исключением привилегии GRANT OPTION, которая всегда задается отдельно или с предложением WITH GRANT OPTION
ALTER	Позволяет редактировать таблицу при помощи оператора ALTER TABLE
ALTER ROUTINE	Позволяет редактировать или удалять хранимую процедуру
CREATE	Позволяет создавать таблицу при помощи оператора CREATE TABLE
CREATE ROUTINE	Позволяет создавать хранимую процедуру
CREATE TEMPORARY TABLES	Позволяет создавать временные файлы
CREATE USER	Позволяет работать с учетными записями при помощи операторов CREATE USER, DROP USER, RENAME USER и REVOKE ALL PRIVILEGES
CREATE VIEW	Позволяет создавать представление при помощи оператора CREATE VIEW
DELETE	Позволяет удалять записи при помощи оператора DELETE
DROP	Позволяет удалять таблицы при помощи оператора DROP TABLE
EVENT	Позволяет создать события для планировщика заданий
EXECUTE	Позволяет выполнять хранимые процедуры
FILE	Позволяет работать с файлами при помощи операторов SELECT...INTO OUTFILE и LOAD DATA INFILE
INDEX	Позволяет работать с индексами, в частности, использовать операторы CREATE INDEX и DROP INDEX
INSERT	Позволяет добавлять в таблицу новые записи при помощи оператора INSERT
LOCK TABLES	Позволяет осуществлять блокировки таблиц при помощи операторов LOCK TABLES и UNLOCK TABLES. Для вступления в действие этой привилегии должна быть установлена привилегия SELECT
PROCESS	Позволяет использовать оператор SHOW FULL PROCESSLIST
REFERENCES	Зарезервировано, но не реализовано
RELOAD	Позволяет использовать оператор FLUSH для обновления таблиц разрешений, кэша и журналов
REPLICATION CLIENT	Позволяет запрашивать, является ли сервер главным или подчиненным в репликации

Таблица 2.25 *(окончание)*

Привилегия	Операция, разрешенная привилегией
REPLICATION SLAVE	Данная привилегия необходима для подчиненных серверов репликации
SELECT	Позволяет осуществлять выборки таблиц при помощи оператора SELECT
SHOW DATABASES	Позволяет просматривать список всех таблиц на сервере MySQL при помощи оператора SHOW DATABASES
SHOW VIEW	Позволяет использовать оператор SHOW CREATE VIEW
SHUTDOWN	Позволяет завершать работу сервера при помощи mysqladmin shutdown
SUPER	Позволяет выполнять административные функции при помощи операторов CHANGE MASTER, KILL, PURGE MASTER LOGS и SET GLOBAL
TRIGGER	Позволяет создавать и уничтожать триггеры
UPDATE	Позволяет обновлять содержимое таблиц при помощи оператора UPDATE
USAGE	Синоним для статуса "отсутствуют привилегии"
GRANT OPTION	Позволяет управлять привилегиями других пользователей, без данной привилегии невозможно выполнить операторы GRANT и REVOKE

Рассмотрим несколько примеров. Для того чтобы разрешить пользователю editor полный доступ на просмотр, заполнение, редактирование и удаление таблиц, необходимо воспользоваться запросом, представленным в листинге 2.75.

Листинг 2.75. Предоставление привилегий для пользователя editor

```
GRANT SELECT, INSERT, DELETE, UPDATE ON *.* TO editor;
```

Если в качестве пользователя выступает приложение, которое добавляет новые записи или обновляет текущие, то его права можно ограничить лишь привилегиями INSERT и UPDATE (листинг 2.76). Это позволит избежать выполнения операций DELETE и SELECT, которыми может воспользоваться злоумышленник при помощи инъекционного кода.

Листинг 2.76. Предоставление привилегий INSERT и UPDATE

```
GRANT INSERT, UPDATE ON *.* TO program;
```

Для того чтобы назначить все привилегии сразу, следует воспользоваться запросами, представленными в листинге 2.77.

Листинг 2.77. Предоставление всех привилегий

```
GRANT ALL ON *.* TO superuser;
GRANT GRANT OPTION ON *.* TO superuser;
```

Выполнить операции по наделению пользователя всеми правами в одном запросе не получится, т. к. ключевое слово ALL всегда употребляется отдельно и не должно использоваться совместно с другими ключевыми словами из табл. 2.25 (листинг 2.78).

Листинг 2.78. Ошибка при совместном использовании ALL и GRANT OPTION

```
GRANT ALL, GRANT OPTION ON *.* TO superuser;
ERROR 1064: You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near '
GRANT OPTION ON *.* TO superuser' at line 1
```

Ключевое слово ON в операторе GRANT задает уровень привилегий, которые могут быть заданы на одном из четырех уровней, представленных в табл. 2.26.

***Таблица 2.26.** Уровни привилегий*

Ключевое слово ON	Уровень
ON *.*	Глобальный уровень касается всех баз данных и таблиц. Пользователь с полномочиями на глобальном уровне может обращаться ко всем базам данных
ON *	Если текущая база данных не была выбрана при помощи оператора USE, данное предложение эквивалентно ON *.*, если произведен выбор текущей базы данных, то устанавливаемые при помощи оператора GRANT привилегии относятся ко всем таблицам текущей базы данных
ON *db*.*	Уровень базы данных — привилегии распространяются на таблицы базы данных *db*
ON *db*.*tbl*	Уровень таблицы — привилегии распространяются на таблицу *tbl* базы данных *db*
ON *db*.*tbl*	Уровень столбца — привилегии уровня столбца касаются отдельных столбцов в таблице *tbl* базы данных *db*. Список столбцов указывается в скобках через запятую после ключевых слов SELECT, INSERT, UPDATE

Привилегии EXECUTION, FILE, PROCESS, RELOAD, REPLICATION CLIENT, REPLICATION SLAVE, SHOW DATABASES, SHUTDOWN и SUPER могут быть установлены лишь на глобальном уровне. Для таблиц можно установить только следующие типы привилегий: SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, GRANT OPTION, INDEX и ALTER. Это следует учитывать при использовании конструкции GRANT ALL, которая назначает привилегии на текущем уровне. Так запрос уровня базы данных GRANT ALL ON *db*.* не предоставляет никаких глобальных привилегий, таких как FILE или RELOAD.

В листинге 2.79 на примере создания пользователя editor демонстрируется использование привилегий уровня базы данных (test).

Листинг 2.79. Предоставление привилегий уровня базы данных

```
GRANT ALL ON test.* TO editor;
```

Замечание

Следует отметить, что отсутствие конструкции `IDENTIFIED BY` приводит к тому, что в качестве пароля пользователя назначается пустая строка.

Если у пользователя нет привилегий на доступ к базе данных, имя базы данных не отображается в ответ на запрос `SHOW DATABASES`, если только у пользователя нет специальной привилегии `SHOW DATABASES`.

Следующий уровень — это привилегии на уровне таблицы. В листинге 2.80 представлен запрос, который наделяет учетную запись `manager` полными привилегиями для доступа к таблице `user` базы данных `test`.

Листинг 2.80. Предоставление привилегий уровня таблицы

```
GRANT ALL ON test.user TO manager;
```

Если у пользователя нет привилегий на доступ к таблице, то эта таблица не отображается в ответ на запрос списка таблиц базы данных — `SHOW TABLES`.

В именах баз данных и таблиц допускается использование символов `%` и `_`, которые имеют тот же смысл, что и в операторе `LIKE`, т. е. заменяют произвольное число и один символ соответственно. Это означает, что если требуется использовать символ `_` как часть имени базы данных, его необходимо экранировать обратным слэшем, иначе можно нечаянно предоставить доступ к базам данных, соответствующим введенному шаблону. Например, для базы данных `list_user` оператор `GRANT` может выглядеть так, как это представлено в листинге 2.81.

Листинг 2.81. Символ _ в имени базы данных

```
GRANT ALL ON 'list\_user'.* TO 'wet'@'localhost';
```

Особняком в привилегиях стоит привилегия `GRANT OPTION` — способность наделять других пользователей правами на передачу пользовательских прав. Обычно эту привилегию назначают при помощи отдельного запроса, но язык запросов SQL предоставляет специальное предложение `WITH GRANT OPTION`, которое позволяет наделять учетную запись этой привилегией. Предложение `WITH GRANT OPTION` помещается в конце запроса (листинг 2.82).

Листинг 2.82. Использование предложения `WITH GRANT OPTION`

```
GRANT ALL ON test.* TO 'wet'@'localhost' WITH GRANT OPTION;
```

В результате выполнения запроса из листинга 2.82 пользователь `wet` наделяется правами вызова оператора `GRANT ALL` для предоставления привилегий другим пользователям на базу данных `test` и ее таблицы. Ни на какие другие базы данных пользователь `wet` не имеет права выдавать привилегии. Другими словами, пользователь,

обладающий правами GRANT OPTION, может передавать другим пользователям только те права, которые принадлежат ему самому.

Следует крайне осторожно обращаться с привилегией GRANT OPTION, т. к. она распространяется не только на привилегии, которыми пользователь обладает в настоящий момент, но и на все привилегии, которые он может получить в будущем.

Для отмены привилегий учетной записи предназначен оператор REVOKE (листинг 2.83). Его синтаксис похож на синтаксис оператора GRANT с той лишь разницей, что ключевое слово TO заменено на FROM, а предложения IDENTIFIED BY, REQUIRE и WITH GRANT OPTION отсутствуют.

Листинг 2.83. Использование оператора REVOKE

```
REVOKE DELETE, UPDATE ON shop.* FROM 'wet'@'localhost';
```

Следует помнить, что оператор REVOKE отменяет привилегии, но не удаляет учетные записи, для их удаления необходимо воспользоваться оператором DROP USER.

Для просмотра существующих привилегий необходимо выполнить оператор SHOW GRANTS (листинг 2.84). Если в результирующей таблице учетной записи нет, пользователь не обладает никакими правами, и его учетная запись может быть удалена.

Листинг 2.84. Использование оператора SHOW GRANTS

```
SHOW GRANTS;
+-----------------------------------------------------------------------+
| Grants for root@localhost                                             |
+-----------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
```

Как видно из листинга 2.84, только пользователь 'root'@'localhost' обладает привилегиями, все остальные пользователи могут быть удалены при помощи оператора DROP USER.

2.3.5. Ограничение на число соединений с сервером и число запросов

Ключевое слово WITH в операторе GRANT помимо привилегии GRANT OPTION позволяет наложить ограничения на число подключений, запросов, обновлений и параллельных запросов в час. Это осуществляется при помощи опций MAX_CONNECTIONS_PER_HOUR, MAX_QUERIES_PER_HOUR, MAX_UPDATES_PER_HOUR и MAX_USER_CONNECTIONS соответственно. Запрос в листинге 2.85 устанавливает для пользователя wet полный доступ к базе данных shop, но с ограничением — не больше 10 подключений к серверу MySQL в час и не более 1000 запросов, из которых только 200 могут быть операциями обновления UPDATE, а 3 запроса протекать одновременно.

Листинг 2.85. Ограничение учетной записи

```
GRANT ALL ON shop.* TO 'wet'@'localhost' IDENTIFIED BY 'pass'
WITH MAX_CONNECTIONS_PER_HOUR 10
     MAX_QUERIES_PER_HOUR 1000
     MAX_UPDATES_PER_HOUR 200
     MAX_USER_CONNECTIONS 3;
```

Если значение каждой из опции равно 0 (по умолчанию), это означает, что у пользователя нет ограничений по этому параметру.

2.3.6. Перенос каталога данных на другой диск

При установке базы данных устанавливаются в подкаталог C:/Documents and Settings/All Users/Application Data/MySQL/MySQL Server 5.1/Data/. Имеет смысл назначить диск C только для установки программного обеспечения, а все данные сосредоточить на другом диске, например, D. Такой подход позволяет форматировать диск C перед установкой или переустановкой операционной системы, не боясь потерять ценные данные.

Перед манипуляциями с каталогом данных необходимо остановить сервер MySQL *(см. раздел 1.13)*. После этого необходимо переместить данные из подкаталога C:/Documents and Settings/All Users/Application Data/MySQL/MySQL Server 5.1/Data/ в новое место, допустим, D:/mysql. В конфигурационный файл my.ini необходимо внести изменения в секцию `[mysqld]`, указав для директивы `datadir` новый путь (листинг 2.86).

Листинг 2.86. Фрагмент файла my.ini

```
...
[mysqld]
...
datadir="D:/mysql/"
...
```

После завершения переноса каталога данных и редактирования конфигурационного файла my.ini MySQL-сервер необходимо запустить. В старой версии MySQL необходимо заменить системный каталог базы данных mysql новым, т. к. структура базы данных регулярно изменяется от версии к версии.

2.3.7. Перенос баз данных с одного сервера на другой

Рассмотрим перенос баз данных с одного сервера на другой или просто создание резервной копии баз данных. Чаще всего используется одно из двух решений: копирование бинарных файлов баз данных или создание SQL-дампа.

2.3.7.1. Копирование бинарных файлов

Таблицы типа MyISAM (основной тип таблиц в MySQL) можно перемещать с одного сервера на другой независимо от версии сервера и операционной системы, под управлением которой он работает.

> ***Замечание***
> Создание в каталоге данных нового каталога аналогично созданию новой базы данных.

В каталоге данных для каждой базы данных заводится свой подкаталог, каждая таблица представлена тремя файлами, имена которых совпадают с именем таблицы, а расширения имеют следующий смысл:

- frm — определяет структуру таблицы, имена полей, их типы, параметры таблицы и т. п.;
- MYD — содержит данные таблицы (расширение образовано от сокращения MYData);
- MYI — содержит индексную информацию (расширение образовано от сокращения MYIndex).

Однако копирование данных непосредственно из каталога данных работающего сервера может привести к повреждению копий таблиц, причем это может случиться даже в том случае, если MySQL-сервер не обращался к копируемым таблицам в текущий момент. Поэтому перед копированием бинарных данных необходимо либо остановить сервер, либо блокировать таблицы на запись. Блокировку таблиц удобно осуществить при помощи оператора `FLUSH TABLES` (листинг 2.87).

Листинг 2.87. Блокировка таблиц на запись

```
FLUSH TABLES WITH READ LOCK;
```

Для того чтобы блокировка осталась в силе на время копирования, следует оставить клиент включенным и не выходить из него до тех пор, пока копирование каталога данных не будет завершено.

Для того чтобы снять блокировку на запись, следует выполнить запрос `UNLOCK TABLES` (листинг 2.88).

Листинг 2.88. Снятие блокировки

```
UNLOCK TABLES;
```

В дистрибутив MySQL входит скрипт горячего копирования бинарных файлов баз данных `mysqlhotcopy`. Данный скрипт действует по той же схеме, что описана ранее: блокирует таблицы базы данных `base` и копирует их бинарное представление по указанному пути /to/new/path (листинг 2.89).

Листинг 2.89. Использование скрипта `mysqlhotcopy`

```
mysqlhotcopy base /to/new/path
```

2.3.7.2. Создание SQL-дампа

Иногда требуется развернуть базу данных в другой СУБД, в СУБД MySQL более ранней версии, не поддерживающей нововведения поздних версий, или на сервере, где отсутствует доступ к каталогу данных. В этом случае часто прибегают к созданию SQL-дампов. SQL-дамп — это текстовый файл с SQL-инструкциями, выполнение которых воссоздает базу данных.

Основным инструментом для создания SQL-дампов служит утилита mysqldump. Для того чтобы создать резервную копию базы данных, например `base`, необходимо выполнить команду, представленную в листинге 2.90.

Листинг 2.90. Создание дампа базы данных `base`

```
C:\mysql\bin> mysqldump –u root base > base.sql
```

Как видно из листинга 2.91, утилита `mysqldump` принимает имя пользователя при помощи параметра `-u`. Если учетная запись защищена паролем, следует добавить параметр `-p`. После всех параметров указывается имя базы данных `base`, для которой производится создание дампа. Так как вывод данных осуществляется в стандартный поток (за которым по умолчанию закреплен экран монитора), его следует перенаправить в файл (в листинге 2.90 это base.sql). Перенаправление данных осуществляется при помощи оператора >. Если вместо оператора > использовать >>, то данные не будут перезаписывать уже существующий файл, а будут добавлены в конец файла.

При помощи параметра `--databases` (в сокращенной форме `-B`) можно создать дамп сразу нескольких баз данных, указав их через пробел (листинг 2.91).

Листинг 2.91. Создание дампа нескольких баз данных

```
C:\mysql5\bin> mysqldump –u root –B base mysql > base_mysql.sql
```

Так команда, представленная в листинге 2.91, сохраняет дампы баз данных `base` и `mysql` в файл base_mysql.sql.

Если необходимо сохранить дамп всех баз данных MySQL-сервера, следует воспользоваться параметром `--all-databases` или в сокращенной форме `-A` (листинг 2.92).

Листинг 2.92. Создание дампа всех баз данных MySQL-сервера

```
mysqldump –u root --all-databases > all_databases.sql
```

Развернуть SQL-дамп на другом сервере можно при помощи утилиты mysql в пакетном режиме (листинг 2.93).

Листинг 2.93. Развертывание дампа базы данных с использованием mysql

```
mysql –u root test < base.sql
```

В листинге 2.93 данные из дампа base.sql перенаправляются на стандартный вход утилиты mysql, которая размещает таблицы базы данных `base` в базе данных `test`.

Развернуть SQL-дамп можно не только в пакетном режиме, но и в диалоговом. Самый простой способ — это воспользоваться командой `SOURCE`, которая выполняет несколько инструкций, перечисленных в SQL-дампе (листинг 2.94).

Замечание

Следует указывать абсолютный путь к файлу в команде `SOURCE` или помещать SQL-дамп в каталог bin. В последнем случае достаточно использовать лишь имя файла.

Листинг 2.94. Использование команды `SOURCE`

```
mysql> SOURCE base.sql;
```

2.3.8. Настройка phpMyAdmin

Основным клиентом при работе с СУБД MySQL на сайтах многих хост-провайдеров выступает Web-интерфейс phpMyAdmin.

Замечание

Последнюю версию phpMyAdmin вы можете загрузить с официального сайта этого Web-приложения по адресу **http://www.phpmyadmin.net/**.

После распаковки содержимого архива с phpMyAdmin необходимо открыть файл config.inc.php и изменить значение хоста `$cfg['Servers'][$i]['host']`, пользователя базы данных `$cfg['Servers'][$i]['user']`, его пароль `$cfg['Servers'][$i]['password']` и режима идентификации согласно вашему окружению. Здесь "хост" означает сервер, где расположена СУБД MySQL. Так в случае локальной машины можно подразумевать "localhost". Также необходимо исправить значение для `$cfg['PmaAbsoluteUri']` с целью указания точного сетевого адреса phpMyAdmin: если настройка Web-приложения происходит на локальной машине, и в первом пункте архив был распакован в папку phpMyAdmin, то следует изменить значение этой директивы на `http://localhost/phpMyAdmin`.

phpMyAdmin может применяться для администрирования нескольких серверов MySQL, поэтому будьте внимательны: исправления для одного сервера не коснутся другого сервера. Одна из ошибок конфигурирования заключается в изменении упомянутых директив для одного из последующих серверов, в то время как для первого сервера директивы остаются невыполненными.

Замечание

Конфигурационный файл config.inc.php является PHP-скриптом, и его синтаксис должен соответствовать правилам. Поэтому начало следующего сервера отмечается оператором инкремента переменной `$i`. Авторы phpMyAdmin настоятельно рекомендуют не использовать абсолютные значения серверов, как `$cfg['Servers'][0]`, `$cfg['Servers'][1]` и т. д.

Если в качестве сервера базы данных выступает MySQL версии выше 4.1, может потребоваться изменить еще несколько директив конфигурационного файла phpMyAdmin для настройки кодировок по умолчанию. Так если планируется использовать в качестве кодировки по умолчанию cp1251, то следует выставить следующие значения для директив `$cfg['DefaultLang']` и `$cfg['DefaultCharset']`:

```
$cfg['DefaultLang'] = 'ru-win1251';
$cfg['DefaultCharset'] = 'windows-1251';
```

По умолчанию SQL-дампы в phpMyAdmin для MySQL версии выше 4.1 создаются в кодировке UTF8. Для того чтобы иметь возможность выбрать кодировку дампа при его создании, необходимо исправить значение директивы `$cfg['AllowAnywhereRecoding']` на `true`:

```
$cfg['AllowAnywhereRecoding'] = true;
```

ГЛАВА 3

Массивы

Массивы являются самым распространенным и простейшим способом группировки данных. Изначально массив определяется как индексированная совокупность переменных одного типа. Отдельные переменные в массиве принято называть *элементами массива*. Однако PHP расширяет это определение, в массивах PHP можно хранить переменные произвольного типа, вплоть до массивов и объектов. Для того чтобы обратиться к элементу массива, необходимо указать индекс или ключ. В листинге 3.1 создается массив `$arr` из двух элементов, первый содержит строку "Hello", второй — "world".

Листинг 3.1. Обращение к элементам массива

```
<?php
  // Создаем массив $arr из двух элементов
  $arr[] = "Hello";
  $arr[] = "world";
  // Выводим второй элемент массива
  echo $arr[1]; // world
?>
```

Для того чтобы обратиться к отдельному элементу массива, необходимо в квадратных скобках указать *индекс элемента массива*. Так обращение к `$arr[0]` запрашивает первый элемент массива, `$arr[1]` — второй и т. д.

Массивы, к элементам которых следует обращаться через числовой индекс, называют *индексными*. Помимо индексного массива PHP поддерживает работу с *ассоциативными массивами*. К элементам таких массивов можно обращаться через строковый индекс, который принято называть *ключом*. В листинге 3.2 создается ассоциативный массив `$arr` из двух элементов, обратиться к первому из которых можно по ключу `first`, а ко второму — по ключу `second`.

Листинг 3.2. Обращение к элементам ассоциативного массива

```
<?php
  // Создаем массив $arr из двух элементов
  $arr['first']  = "Hello";
  $arr['second'] = "world";
  // Выводим второй элемент массива
  echo $arr[1]; // world
?>
```

В последующих разделах мы подробно ознакомимся с тем, как эффективно организовать работу с массивами.

3.1. Создание массива

Для создания массивов PHP предоставляет множество способов, начиная от специализированных конструкций, заканчивая стандартными функциями. Как будет показано в следующих главах книги, многие функции расширений возвращают результаты своей работы в виде массивов. В последующих разделах будут рассмотрены базовые способы создания массивов, поддерживаемые ядром PHP.

3.1.1. Конструкция *array()*

Существует несколько способов создания массивов. Первый из них заключается в использовании конструкции `array()`, в круглых скобках которой через запятую перечисляются элементы массива. В листинге 3.3 создается массив `$arr`, состоящий из трех элементов, пронумерованных от 0 до 2.

Листинг 3.3. Создание массива при помощи конструкции `array()`

```
<?php
  $arr = array("Hello ", "world", "!");
  echo $arr[0]; // Hello
  echo $arr[1]; // world
  echo $arr[2]; // !
?>
```

В листинге 3.3 каждый элемент массива выводится при помощи отдельной конструкции `echo`. При попытке вывода всего массива `$arr` в окно браузера вместо его содержимого будет выведена строка "Array". Для просмотра структуры и содержимого массива предусмотрена специальная функция `print_r()`. В листинге 3.4 с ее помощью выводится структура массива `$arr`. Для удобства восприятия вызов функции `print_r()` обрамляется HTML-тегами `<pre>` и `</pre>`, которые сохраняют структуру переносов и отступов при отображении результата в браузере.

Листинг 3.4. Вывод структуры массива при помощи функции `print_r()`

```
<?php
  $arr = array("Hello ", "world", "!");
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 3.4 будет следующий дамп массива `$arr`:

```
Array
(
    [0] => Hello
    [1] => world
    [2] => !
)
```

Как видно из листингов 3.3 и 3.4, элементам массива автоматически назначены индексы, начиная с нулевого, как это принято в C-подобных языках программирования. Однако индекс начального элемента, а также порядок следования индексов можно изменять. Для этого в конструкции `array()` перед элементом указывается его индекс при помощи оператора `=>`. В листинге 3.5 первому элементу массива `$arr` назначается индекс 10, последующие элементы автоматически получают значения 11 и 12.

Листинг 3.5. Назначение индекса первому элементу массива

```
<?php
  $arr = array(10 => "Hello ", "world", "!");
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 3.5 будет следующий дамп массива `$arr`:

```
Array
(
    [10] => Hello
    [11] => world
    [12] => !
)
```

Замечание

В массиве `$arr` из листинга 3.5 элементы с индексами 0–9 и 13 просто не существуют, обращение к ним воспринимается как обращение к неинициализированной переменной (т. е.

переменной, имеющей значение NULL). Если в конфигурационном файле php.ini включено отображение замечаний (Notice), при попытке обращения к несуществующему элементу массива будет выдано соответствующее предупреждение.

Назначать индексы можно не только первому элементу, но и всем последующим элементам массива (листинг 3.6).

Листинг 3.6. Назначение индексов всем элементам массива

```
<?php
  $arr = array(10 => "Hello ", 9 => "world", 8 => "!");
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 3.6 будет следующий дамп массива $arr:

```
Array
(
    [10] => Hello
    [9] => world
    [8] => !
)
```

3.1.2. Непосредственное создание элементов

Еще одним способом создания массива является присвоение неинициализированным элементам массива новых значений (листинг 3.7).

Листинг 3.7. Создание массива при помощи квадратных скобок

```
<?php
  $arr[10] = "Hello ";
  $arr[11] = "world";
  $arr[12] = "!";
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Скрипт из листинга 3.7 по результату выполнения эквивалентен листингу 3.5. Допускается и автоматическое назначение индексов элементам, для этого значение индекса просто не указывается в квадратных скобках (листинг 3.8).

Листинг 3.8. Автоматическое назначение индекса

```
<?php
  $arr[] = "Hello ";
  $arr[] = "world";
  $arr[] = "!";
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

3.1.3. Создание массива: приведение типа

Еще один способ создания массива заключается в приведении скалярной переменной (т. е. переменной типа `int`, `float`, `string` или `boolean`) к типу `array`. Результатом этого становится массив, содержащий один элемент с индексом 0 и значением, равным значению переменной (листинг 3.9).

Листинг 3.9. Приведение переменной к массиву

```
<?php
  $var = "Hello world";
  $arr = (array) $var;
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 3.9 будет следующий дамп массива `$arr`:

```
Array
(
    [0] => Hello world
)
```

3.1.4. Использование специализированных функций

Помимо конструкции `array()`, существует еще несколько функций, которые позволяют создавать массивы. Их список приведен в табл. 3.1.

Таблица 3.1. *Функции создания массивов*

Функция	Описание
`array([...])`	Создает массив из значений, переданных конструкции в качестве параметров

Таблица 3.1 *(окончание)*

Функция	Описание
`array_fill ($start_index, $num, $value)`	Возвращает массив, содержащий `$num` элементов, имеющих значение `$value`. Нумерация индексов при этом начинается со значения `$start_index`
`range($low, $high [, $step])`	Создает массив со значениями из интервала от `$low` до `$high` и шагом `$step`
`explode($delimiter, $str [, $limit])`	Функция возвращает массив из строк, каждая из которых соответствует фрагменту исходной строки `$str`, находящемуся между разделителями, определяемым аргументом `$delimiter`. Необязательный параметр `$limit` определяет максимальное количество элементов в массиве

Если элементы массива должны содержать одинаковые значения, для его создания удобно воспользоваться функцией `array_fill()`, которая имеет следующий синтаксис:

```
array_fill($start_index, $num, $value)
```

Функция возвращает массив, содержащий `$num` элементов, имеющих значение `$value`. Нумерация индексов при этом начинается со значения `$start_index` (листинг 3.10).

Листинг 3.10. Создание массива при помощи функции `array_fill()`

```
<?php
  // Создаем массив
  $arr = array_fill(5, 6, "Hello world!");
  // Выводим дамп массива
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 3.10 будут следующие строки:

```
Array
(
    [5] => Hello world!
    [6] => Hello world!
    [7] => Hello world!
    [8] => Hello world!
    [9] => Hello world!
    [10] => Hello world!
)
```

Еще одним удобным способом создания массивов является использование функции `range()`, которая позволяет создавать массивы для интервалов значений и имеет следующий синтаксис:

```
range($low, $high [, $step])
```

Параметр `$low` определяет начало интервала, а `$high` — его окончание. Необязательный параметр `$step` позволяет задать шаг, с которым будут создаваться элементы. В листинге 3.11 демонстрируется создание массива из 6 элементов, которые принимают значения в интервале от 0 до 5.

Листинг 3.11. Использование функции `range()`

```
<?php
  $arr = range(0, 5);
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 3.11 будет следующий дамп массива:

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

По умолчанию, массив заполняется значениями из интервала с шагом, равным единице, однако этот шаг можно изменить при помощи третьего параметра функции `range()` (листинг 3.12).

Замечание

В функции `range()` допускается использование отрицательного шага.

Листинг 3.12. Использование шага в функции `range()`

```
<?php
  $arr = range(0, 1, 0.2);
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 3.12 будет следующий дамп массива:

```
Array
(
    [0] => 0
    [1] => 0.2
```

```
    [2] => 0.4
    [3] => 0.6
    [4] => 0.8
    [5] => 1
)
```

Часто возникает задача получения массива за счет разбиения строки по какому-либо разделителю. В этом случае удобно воспользоваться функцией `explode()`, которая имеет следующий синтаксис:

```
explode($delimiter, $str [, $limit])
```

Функция возвращает массив из строк, каждая из которых соответствует фрагменту исходной строки `$str`, находящемуся между разделителями, определяемыми параметром `$delimiter`.

Необязательный параметр `$limit` определяет максимальное количество элементов в массиве. Оставшаяся (неразделенная) часть будет содержаться в последнем элементе. Пример использования функции `explode()` приведен в листинге 3.13.

> ***ЗАМЕЧАНИЕ***
>
> Для того чтобы преобразовать массив в строку, используется обратная функция `implode()`.

Листинг 3.13. Использование функции `explode()`

```
<?php
  $str = "Имя Фамилия e-mail";
  $arr = explode(" ", $str);
  echo "<pre>";
  print_r ($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 3.13 являются следующие строки:

```
Array
(
  [0] => Имя
  [1] => Фамилия
  [2] => e-mail
)
```

В листинге 3.13 строка `$str` разбивается на отдельные элементы массива `$arr` по пробельному символу. В качестве первого параметра функция `explode()` может использовать не одиночный символ, а целые последовательности, например, запятая и пробел ", " или перевод строки "\r\n". Однако если необходимо разбить строку по сложному критерию, например, по пробелам, переводам строк, причем количество пробельных символов может быть произвольным, вместо функции `explode()` следует использовать регулярные выражения.

3.1.5. Многомерные массивы

В качестве элементов массива могут выступать другие массивы, в этом случае говорят о *многомерных массивах*. Массивы можно создавать, обращаясь к элементам или используя вложенные конструкции `array()`. В листинге 3.14 демонстрируется создание многомерного ассоциативного массива `$ship`.

Листинг 3.14. Создание многомерного массива

```
<?php
  $ship = array(
       "Пассажирские корабли" => array("Лайнер","Яхта","Паром"),
       "Военные корабли" => array("Авианосец","Линкор","Эсминец"),
       "Грузовые корабли" => array("Сормовский","Волго-Дон","Окский")
        );
  echo "<pre>";
  print_r($ship);
  echo "</pre>";
?>
```

В результате такой инициализации будет создан массив следующей структуры:

```
Array
(
  [Пассажирские корабли] => Array
    (
      [0] => Лайнер
      [1] => Яхта
      [2] => Паром
    )
  [Военные корабли] => Array
    (
      [0] => Авианосец
      [1] => Линкор
      [2] => Эсминец
    )
  [Грузовые корабли] => Array
    (
      [0] => Сормовский
      [1] => Волго-Дон
      [2] => Окский
    )
)
```

В этом примере создан двумерный массив с размерами 3×3, т. е. получилось три массива, каждый из которых содержит в себе по три элемента. Следует отметить, что полученный массив является смешанным, т. е. в нем присутствуют как

индексы, так и ключи ассоциативного массива — обращение к элементу `$ship['Пассажирские корабли'][0]` возвратит значение "Лайнер".

3.2. Вывод массива на печать

С одним из способов вывода массива мы уже познакомились, это вывод дампа массива при помощи функции `print_r()` (листинг 3.15).

Листинг 3.15. Вывод дампа массива при помощи функции `print_r()`

```
<?php
  $arr = array(1, 2, 3);
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Это самый быстрый способ, который часто применяется для отладочных целей. Если элементы массива при выводе должны быть оформлены, то можно воспользоваться операторами цикла `for` или `while`. Для ассоциативных массивов предназначен специализированный оператор цикла `foreach`.

Итерационный цикл `for` имеет следующий синтаксис:

```
for(begin; condition; body)
{
  операторы;
}
```

Здесь оператор `begin` (инициализация цикла) — последовательность определений и выражений, разделяемая запятыми. Все выражения, входящие в инициализацию, вычисляются только один раз при входе в цикл. Как правило, здесь устанавливаются начальные значения счетчиков и параметров цикла. Смысл выражения-условия (`condition`) такой же, как и у циклов с пред- и постусловиями — цикл выполняется до тех пор, пока выражение `condition` истинно (`TRUE`), и прекращает свою работу, когда оно ложно (`FALSE`). При отсутствии выражения-условия предполагается, что его значение всегда истинно. Выражения `body` вычисляются в конце каждой итерации после выполнения тела цикла.

В листинге 3.16 демонстрируется обход индексного массива при помощи цикла `for`.

Листинг 3.16. Обход массива в цикле `for`

```
<?php
  $number = array("1", "2", "3");
  for($i=0; $i < count($number); $i++)
```

```
  {
    echo $number[$i];
  }
?>
```

Результатом работы скрипта из листинга 3.16 будет строка: `123`.

Для подсчета количества элементов в массиве используется функция `count()` *(см. разд. 3.3)*, которая принимает в качестве единственного параметра массив и возвращает количество элементов в нем. Так в листинге 3.16 для массива `$number` будет возвращено значение 3.

Обход массива в цикле можно организовать при помощи цикла `foreach`, который специально создан для ассоциативных массивов и имеет следующий синтаксис:

```
foreach($array as [$key =>] $value)
{
    операторы;
}
```

Цикл последовательно обходит элементы массива `$array` с первого до последнего, помещая на каждой итерации цикла ключ в переменную `$key`, а значение в переменную `$value`. Имена этих переменных могут быть любые (листинг 3.17).

> ***ЗАМЕЧАНИЕ***
>
> В качестве первого аргумента конструкции `foreach` обязательно должен выступать массив, иначе конструкция выводит предупреждение.

Листинг 3.17. Обход массива в цикле `foreach`

```
<?php
  $number = array("first" => "1",
                  "second" => "2",
                  "third" => "3");
  foreach($number as $index => $val) echo "$index = $val <br>";
?>
```

Результатом работы скрипта из листинга 3.17 будут следующие строки:

```
first = 1
second = 2
third = 3
```

Переменная `$index` для ключа массива необязательна и может быть опущена (листинг 3.18).

Листинг 3.18. Вариант обхода массива в цикле `foreach`

```
<?php
  $number = array("first" => "1",
                  "second" => "2",
                  "third" => "3");
```

```
  foreach($number as $val)
  {
    echo $val; // выведет 123
  }
?>
```

Обход многомерных массивов проводится с помощью вложенных циклов `foreach`, при этом число вложенных циклов соответствует размерности массива (листинг 3.19).

Листинг 3.19. Обход многомерных массивов в цикле

```
<?php
  $ship = array(
       "Пассажирские корабли" => array("Лайнер","Яхта","Паром"),
       "Военные корабли" => array("Авианосец","Линкор","Эсминец"),
       "Грузовые корабли" => array("Сормовский","Волго-Дон","Окский")
        );
  foreach($ship as $key => $type)
  {
    // вывод значений основных массивов
    echo "<b>$key</b><br>";
    foreach($type as $ship)
    {
      // вывод значений для каждого из массивов
      echo "<li>$ship</li><br>";
    }
  }
?>
```

Результат выполнения скрипта из листинга 3.19 представлен на рис. 3.1.

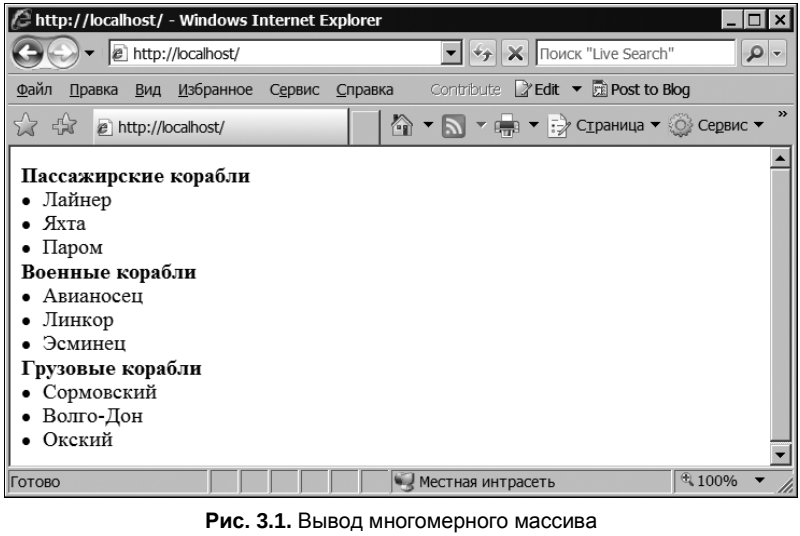


Рис. 3.1. Вывод многомерного массива

3.3. Количество элементов в массиве

Для манипуляций с массивами и их элементами часто необходимо определить количество элементов в массиве. В предыдущем разделе мы уже сталкивались с функцией `count()` для подсчета количества элементов в массиве для обхода их в цикле. Полный список функций, предназначенных для решения этой задачи, представлен в табл. 3.2.

Таблица 3.2. *Количество элементов в массиве*

Функция	Описание
`count ($array [, $mode])`	Возвращает количество элементов массива `$array`. Если `$mode` принимает значение `COUNT_RECURSIVE`, функция рекурсивно обходит многомерный массив, в противном случае подсчитывается количество элементов только на текущем уровне
`sizeof()`	Синоним для функции `count()`
`array_count_values ($input)`	Подсчитывает количество уникальных значений среди элементов массива и возвращает ассоциативный массив, ключи которого — значения массива, а значения — количество их вхождений в массив `$input`

Для подсчета количества элементов в массиве предназначена функция `count()`, которая имеет следующий синтаксис:

```
count ($array [, $mode])
```

Возвращает количество элементов массива `$array`. В листинге 3.20 демонстрируется пример с использованием этой функции.

Листинг 3.20. Использование функции `count()`

```
<?php
  $car = array("жигули",
               "волга",
               "запорожец",
               "ока");
  echo count($car); // 4
?>
```

Если в качестве аргумента функции передается многомерный массив, то по умолчанию она возвращает размерность текущего измерения (листинг 3.21).

Листинг 3.21. Использование `count()` совместно с многомерными массивами

```
<?php
  $arr = array(
              array(1, 2, 3, 4),
              array(5, 6, 7, 8)
             );
```

```
  echo count($arr);    // 2
  echo count($arr[0]); // 4
?>
```

В первом случае возвращается число 2, т. к. в первом измерении только 2 элемента — это массивы `array(1, 2, 3, 4)` и `array(5, 6, 7, 8)`. Во втором случае в качестве аргумента передается уже массив `array(1, 2, 3, 4)`, в котором четыре элемента, о чем и сообщает функция `count()`.

Для того чтобы функция рекурсивно обходила многомерный массив, подсчитывая количество всех элементов (в том числе во вложенных массивах), необязательному параметру `$mode` необходимо передать в качестве значения константу `COUNT_RECURSIVE` (листинг 3.22).

Листинг 3.22. Подсчет элементов многомерного массива

```
<?php
  $arr = array(
              array(1, 2, 3, 4),
              array(5, 6, 7, 8)
             );
  echo count($arr, COUNT_RECURSIVE); // 10
?>
```

Еще одной интересной функцией является `array_count_values()`, которая подсчитывает количество уникальных значений среди элементов массива и имеет следующий синтаксис:

```
array_count_values ($array)
```

В качестве результата возвращает ассоциативный массив, в качестве ключей которого выступают значения массива, а в качестве значения — количество их вхождений в массив `$array` (листинг 3.23).

> ***Замечание***
>
> Массив `$array` может иметь в качестве элементов лишь числа или строки, в противном случае генерируется предупреждение.

Листинг 3.23. Использование функции `array_count_values()`

```
<?php
  $array = array (1, "hello", 1, "world", "hello");
  $new = array_count_values ($array);

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

В результате выполнения скрипта из листинга 3.23 будет выведен следующий дамп массива `$new`:

```
Array
(
    [1] => 2
    [hello] => 2
    [world] => 1
)
```

3.4. Переменная или массив?

PHP является слаботипизированным языком программирования, при создании переменной или массива не обязательно указывать их тип. Причем за время работы программы, переменная может многократно изменять свой тип: становиться переменной или массивом. Особенно остро эта проблема встает, если функция, возвращающая массив (в случае неудачи), выдает логическое значение `FALSE`.

Если необходимо убедиться, является ли текущая переменная массивом, используют функцию `is_array()` (листинг 3.24).

Листинг 3.24. Использование функции `is_array()`

```
<?php
  $arr = array(1, 2, 3);

  if(is_array($arr)) echo "Это массив<br />";
  else echo "Это не массив<br />";
  if(is_array($arr[0])) echo "Это массив<br />";
  else echo "Это не массив<br />";
?>
```

В качестве результата скрипт выводит следующие строки:

```
Это массив
Это не массив
```

3.5. Существует ли элемент массива?

Индексы массивов могут начинаться с чисел, отличных от нуля. В случае ассоциативных массивов ключи и вовсе могут быть произвольными. Проверить, существует тот или иной элемент массива, можно при помощи конструкции `isset()` (листинг 3.25).

Листинг 3.25. Проверка существования элементов массива при помощи `isset()`

```
<?php
  $arr = array(5 => 1, 2, 3);
```

```
  for($i = 0; $i < 10; $i++)
  {
    if(isset($arr[$i])) echo "Элемент \$arr[$i] существует<br>";
    else echo "Элемент \$arr[$i] не существует<br>";
  }
?>
```

В качестве результата скрипт из листинга 3.25 выводит следующие строки:

```
Элемент $arr[0] не существует
Элемент $arr[1] не существует
Элемент $arr[2] не существует
Элемент $arr[3] не существует
Элемент $arr[4] не существует
Элемент $arr[5] существует
Элемент $arr[6] существует
Элемент $arr[7] существует
Элемент $arr[8] не существует
Элемент $arr[9] не существует
```

3.6. Как получить список всех индексов массива?

Перебор индексов массива, как это демонстрировалось в предыдущем разделе, не всегда эффективен и возможен. Для массива со сложной структурой либо прибегают к специальному циклу `foreach` (см. *разд. 3.2*), либо получают список ключей массива при помощи функции `array_keys()`, которая имеет следующий синтаксис:

```
array_keys($arr [, $search_value [, $strict]])
```

Функция возвращает массив ключей для массива `$arr`. Если указан необязательный параметр `$search_value`, возвращаются только те ключи, которые указывают на элементы со значением `$search_value`. Если необязательный параметр `$strict` принимает значение `TRUE`, то при сравнении значения элемента массива с `$search_value` будет использоваться оператор эквивалентности `===`, в противном случае будет использован оператор равенства `==`. В листинге 3.26 приводится пример использования функции.

Листинг 3.26. Использование функции `array_keys()`

```
<?php
  // Исходный массив
  $arr = array(0 => 100, "color" => "red");

  // Получаем массив ключей
  $key = array_keys($arr);
```

```
  // Выводим дамп массива
  echo "<pre>";
  print_r($key);
  echo "</pre>";
?>
```

Результат:

```
Array
(
    [0] => 0
    [1] => color
)
```

В листинге 3.27 демонстрируется использование параметра `$search_value`. Исходный массив содержит три элемента со значением `"blue"`, при помощи функции `array_keys()` извлекаются их ключи.

Листинг 3.27. Извлечение ключей элементов со значением `"blue"`

```
<?php
  // Исходный массив
  $arr = array("blue", "red", "green", "blue", "blue");

  // Получаем массив ключей
  $key = array_keys($arr, "blue");

  // Выводим дамп массива
  echo "<pre>";
  print_r($key);
  echo "</pre>";
?>
```

Результат:

```
Array
(
    [0] => 0
    [1] => 3
    [2] => 4
)
```

3.7. Содержит ли массив заданный элемент?

Конструкция `isset()` выполняет проверку существования элемента с заданным ключом, однако помимо этого зачастую требуется выяснить, входит ли в состав того или иного массива элемент с заданным значением.

Функция in_array() осуществляет поиск значения в массиве и имеет следующий синтаксис:

```
in_array($value, $arr [, $strict])
```

Функция проверяет, существует ли значение $value в массиве $arr, и возвращает TRUE, если значение найдено, и FALSE в противном случае. Если необязательный параметр $strict принимает значение TRUE, то при сравнении значения элемента массива с $search_value будет использоваться оператор эквивалентности === (т. е. сравнению будет подвергаться не только значение, но и тип элемента), в противном случае будет использован оператор равенства (листинг 3.28).

Листинг 3.28. Поиск элемента в массиве

```
<?php
  $number = array(0.57, '21.5', 40.52);
  if (in_array(21.5, $number)) echo "Значение 21.5 найдено";
  else echo "Ничего не найдено";
?>
```

В результате будет выведена фраза:

```
Значение 21.5 найдено
```

Заметим, что найденный элемент массива '21.5' взят в одинарные кавычки, т. е. относится к строковому типу, в отличие от других элементов массива $number. В приведенном варианте использования функции это различие не фиксируется. Для того чтобы функция использовала для сравнения оператор эквивалентности ===, вместо оператора равенства == необходимо третий необязательный параметр $strict установить в значение TRUE (листинг 3.29).

Листинг 3.29. Поиск элемента в массиве с различием по типу

```
<?php
  $number = array(0.57, '21.5', 40.52);
  if (in_array(21.5, $number, true)) echo "Значение 21.5 найдено";
  else echo "Ничего не найдено";
?>
```

В результате будет выведена фраза:

```
Ничего не найдено
```

В этом случае ничего найдено не будет, т. к. тип элемента, передаваемого функции in_array() (float), отличается от типа элемента в массиве (string). При таком вызове функции элемент 21.5 будет найден только, если он так же будет взят в кавычки (т. е. тоже будет строкового типа):

```
if (in_array('21.5', $number, true))
```

3.8. Поиск ключа по значению

По значению ключа или индекса очень просто получить значение искомого элемента: достаточно поместить ключ в квадратные скобки. Однако не менее редко может возникать обратная задача — поиск ключа ассоциативного массива по значению. Для проверки существования ключа в массиве и поиска ключа по значению элемента массива предназначены специализированные функции из табл. 3.3.

Таблица 3.3. *Проверка существования ключей*

Функция	Описание
`array_key_exists ($key, $arr)`	Возвращает `TRUE`, если в массиве `$arr` присутствует ключ `$key`, в противном случае возвращается `FALSE`
`array_search($value, $arr [,$strict])`	Ищет значение `$value` в массиве `$arr` и, если значение найдено, возвращает соответствующий ключ, в противном случае возвращается `FALSE`. Если необязательный параметр `$strict` принимает значение `TRUE`, то при сравнении значения элемента массива с `$value` будет использоваться оператор эквивалентности `===`, в противном случае будет использован оператор равенства `==`

Для поиска заданного ключа в массиве можно воспользоваться функцией `array_key_exists()`, которая имеет следующий синтаксис:

```
array_key_exists ($key, $arr)
```

Функция возвращает `TRUE`, если ключ `$key` найден в массиве `$arr` (листинг 3.30).

Листинг 3.30. Поиск ключей массива

```
<?php
  $arr = arr("first_numb" => 1, "second_numb" => 2);
  if (array_key_exists("first_numb", $arr) echo "OK";
?>
```

Найти ключ массива по значению позволяет функция `array_search()`, которая имеет следующий синтаксис:

```
array_search ($value, $arr [, $strict])
```

Функция ищет значение `$value` в массиве `$arr` и, если значение найдено, возвращает соответствующий ключ, в противном случае возвращается `FALSE`. Если необязательный параметр `$strict` принимает значение `TRUE`, то при сравнении значения элемента массива с `$value` будет использоваться оператор эквивалентности `===`, в противном случае будет использован оператор равенства `==`. Пример использования функции приводится в листинге 3.31.

Листинг 3.31. Использование функции array_search()

```
<?php
  $array = array(0 => 'blue', 1 => 'red', 2 => 'green', 3 => 'red');
  $key = array_search('green', $array); // $key = 2;
  $key = array_search('red', $array);   // $key = 1;
?>
```

3.9. Сумма элементов массива

Функция `array_sum()` возвращает сумму всех числовых элементов массива и имеет следующий синтаксис:

```
array_sum($arr)
```

Тип возвращаемого числа определяется типом значений элементов массива. Пример с использованием функции `array_sum()` демонстрируется в листинге 3.32.

Листинг 3.32. Использование функции array_sum()

```
<?php
  $arr = array(1,2,3,4,5);
  $sum = array_sum($arr);
  echo $sum; // выводит 15
?>
```

3.10. Случайные элементы массива

Для получения случайного значения индексного массива можно использовать функцию `rand()`, которая возвращает случайное число из заданного диапазона. Функция `rand()` имеет следующий синтаксис:

```
rand ([$min, $max]);
```

Необязательные параметры `$min` и `$max` задают соответственно начало и конец интервала, из которого выбирается случайное значение. При помощи функции `rand()` можно извлечь случайный индекс массива, указав в качестве начала интервала 0, а в качестве конца — количество элементов в массиве за вычетом единицы (листинг 3.33).

Листинг 3.33. Получение случайного элемента массива

```
<?php
  // Объявляем массив
  $arr = array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9);

  // Получаем случайный индекс массива
  $index = rand(0, count($arr) – 1);
```

```
  // Выводим случайный элемент массива
  echo $arr[$index];
?>
```

Подход, описанный в листинге 3.25, применим только для индексных массивов, индексы которых начинаются с 0 и не имеют перерывов в нумерации. Для всех остальных массивов используется функция `array_rand()` (табл. 3.4), которая возвращает массив выбранных случайным образом индексов элементов массива `$arr`. Функция имеет следующий синтаксис:

```
array_rand ($arr [, $num_req])
```

Функция выбирает случайное значение из массива `$arr`. Если параметр `$num_req` не указывается, возвращается ключ для случайного элемента массива `$arr`, если в параметре `$num_req` задается число больше 1, возвращается массив со случайным набором ключей массива `$arr` (листинг 3.34).

Таблица 3.4. *Случайные элементы массива*

Функция	Описание
`array_rand ($arr [, $num_req]))`	Выбирает одно или несколько случайных значений из массива. Если параметр `$num_req` не указывается, возвращается ключ для случайного элемента массива `$arr`, если в параметре `$num_req` задается число больше 1, возвращается массив со случайным набором ключей массива `$arr`
`shuffle ($arr))`	Перемешивает элементы массива `$arr` в случайном порядке

Листинг 3.34. Использование функции `array_rand()`

```
<?php
  // Инициируем массив
  $arr = array("синий", "желтый", "зеленый", "красный", "оранжевый");
  $rand_keys = array_rand($arr, 2);

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

Еще одной полезной функцией для получения случайных значений является функция `shuffle()` (табл. 3.4), которая перемешивает элементы массива случайным образом и имеет следующий синтаксис:

```
shuffle ($array)
```

Листинг 3.35 демонстрирует работу функции.

Листинг 3.35. Функция `shuffle()`

```
<?php
  $arr = array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
  shuffle($arr);
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результат работы функции `shuffle()` может выглядеть следующим образом:

```
Array
(
    [0] => 7
    [1] => 9
    [2] => 8
    [3] => 1
    [4] => 3
    [5] => 10
    [6] => 2
    [7] => 4
    [8] => 5
    [9] => 6
)
```

3.11. Слияние массивов

Оператор плюс +, позволяющий складывать значения двух переменных, применим и для массивов (листинг 3.36).

Замечание

Наряду с оператором плюс +, допускается использование оператора +=.

Листинг 3.36. Слияние массивов при помощи оператора +

```
<?php
  $fst = array(1 => "one", 2 => "two");
  $snd = array(3 => "three", 4 => "four");
  $sum = $fst + $snd;
  echo "<pre>";
  print_r($sum);
  echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 3.36 в окно браузера будет выведен следующий дамп нового массива `$sum`:

```
Array
(
    [1] => one
    [2] => two
    [3] => three
    [4] => four
)
```

Если в обоих складываемых массивах присутствуют элементы с одинаковыми индексами, в результирующий массив попадает только один элемент из правого массива (листинг 3.37).

Листинг 3.37. В результирующий массив попадает только один элемент с одинаковыми индексами

```
<?php
  $fst = array("one", "two");
  $snd = array("three", "four", "five");
  $sum = $fst + $snd;
  echo "<pre>";
  print_r($sum);
  echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 3.37 в окно браузера будет выведен следующий дамп нового массива `$sum`:

```
Array
(
    [0] => one
    [1] => two
    [2] => five
)
```

Для того чтобы в результирующий массив попали элементы обоих массивов, вместо оператора + используют специальные функции (табл. 3.5).

***Таблица 3.5.** Функции слияния массивов*

Функция	Описание
`array_merge ($array1 [, $array2 [, ...]])`	Сливает массивы `$array1`, `$array2`, ... в один и возвращает его в качестве значения
`array_merge_recursive ($array1 [,$array2, [, ...]])`	Рекурсивно сливает массивы `$array1`, `$array2`, ... в один и возвращает его в качестве значения

В листинге 3.38 приводится пример использования функции `array_merge()`.

Замечание

Функция `array_merge()` может принимать в качестве параметров более чем два массива.

Листинг 3.38. Использование функции `array_merge()`

```
<?php
  $fst = array("one", "two");
  $snd = array("three", "four", "five");
  $sum = array_merge($fst, $snd);
  echo "<pre>";
  print_r($sum);
  echo "</pre>";
?>
```

В результате выполнения скрипта в окно браузера будет выведен следующий дамп нового массива `$sum`:

```
Array
(
    [0] => one
    [1] => two
    [2] => three
    [3] => four
    [4] => five
)
```

В арсенале PHP имеется функция `array_merge_recursive()`, которая ведет себя аналогично `array_merge()` в отношении индексных массивов и как оператор + в отношении ассоциативных массивов (листинг 3.39).

Листинг 3.39. Использование функции `array_merge_recursive()`

```
<?php
  $ar1 = array("color" => array ("favorite" => "red"), 5);
  $ar2 = array(10, "color" => array ("favorite" => "green", "blue"));
  $sum = array_merge_recursive ($ar1, $ar2);
  echo "<pre>";
  print_r($sum);
  echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 3.39 в окно браузера будет выведен следующий дамп нового массива `$sum`:

```
Array
(
    [color] => Array
```

```
        (
            [favorite] => Array
                (
                    [0] => red
                    [1] => green
                )
            [0] => blue
        )
    [0] => 5
    [1] => 10
)
```

3.12. Преобразование каждого элемента массива

В общем случае для того, чтобы обработать элементы массива, достаточно обойти их в цикле и применить требуемые операции к каждому из элементов. Так как эта задача возникает достаточно часто, в PHP предусмотрена специализированная функция `array_walk()`, которая имеет следующий синтаксис:

```
array_walk ($arr, $funcname [, $userdata])
```

Функция применяет пользовательскую функцию `$funcname` ко всем элементам массива `$arr`. Параметр `$userdata` используется в качестве дополнительного параметра пользовательской функции.

В пользовательскую функцию передаются два или три аргумента: значение текущего элемента, его индекс и аргумент `$userdata`. Последний аргумент является необязательным. В случае если `$funcname` требует более трех аргументов, при каждом ее вызове будет выдаваться предупреждение, и, чтобы они не выдавались, нужно поставить знак `@` перед функцией `array_walk()`.

Пусть необходимо вывести все элементы массива. Для этого можно сначала написать функцию, которая будет их выводить, а затем вызвать ее для каждого из элементов массива (листинг 3.40).

> ***Замечание***
>
> Функция `array_walk()` имеет модификацию `array_walk_recursive()`, которая позволяет применять пользовательскую функцию к многомерным массивам.

Листинг 3.40. Использование функции `array_walk()`

```
<?php
  $name = array ("m"=>"maksim", "i"=>"igor", "s"=>"sergey");
  function print_array ($item, $key)
  {
    echo "$key=>$item<br>\n";
  }
```

```
  /* теперь применим функцию print_array
  к каждому элементу массива $name */
  array_walk ($name, 'print_array');
?>
```

Результат:

```
m => maksim
i => igor
s => sergey
```

Функция `array_map())` также предназначена для применения пользовательской функции к элементам массивов. Функция имеет следующий синтаксис:

```
array_map ($callback, $arr1 [, ...])
```

Функция применяет пользовательскую функцию `$callback` ко всем элементам массивов, указанных в последующих параметрах. Количество параметров, передаваемых функции обратного вызова, должно совпадать с количеством массивов, переданным функции `array_map()`. В листинге 3.41 демонстрируется работа функции — массив `$arr` преобразуется в массив `$cub`, элементы которого представляют кубическую степень от элементов массива `$arr`.

Листинг 3.41. Возведение элементов массива в куб

```
<?php
  // Пользовательская функция для возведения
  // элементов массива в куб
  function cube($n)
  {
    return $n*$n*$n;
  }
  // Исходный массив
  $arr = array(1, 2, 3, 4, 5);

  // Возводим элементы массива в куб
  $cub = array_map("cube", $arr);

  // Выводим дамп массива
  echo "<pre>";
  print_r($cub);
  echo "</pre>";
?>
```

Результат:

```
Array
(
    [0] => 1
    [1] => 8
```

```
    [2] => 27
    [3] => 64
    [4] => 125
)
```

3.13. Получение уникальных элементов массива

Массивы могут содержать повторяющиеся элементы, часто требуется получить копию массива, содержащую только уникальные значения. Для этого удобно воспользоваться функцией `array_unique()`, которая имеет следующий синтаксис:

```
array_unique ($array)
```

В листинге 3.42 приводится пример использования функции.

Листинг 3.42. Получение массива с уникальными значениями

```
<?php
  $input = array("a" => "green", "red", "b" => "green", "blue", "red");
  $result = array_unique($input);
  echo "<pre>";
  print_r($result);
  echo "</pre>";
?>
```

Результатом работы скрипта будет следующий дамп массива `$result`:

```
Array
(
    [a] => green
    [0] => red
    [1] => blue
)
```

3.14. Преобразование элементов массива в переменные

Массивы, как и обычные переменные, могут выступать в качестве аргументов и возвращаемых значений пользовательских функций (листинг 3.43).

Листинг 3.43. Функция, возвращающая массив

```
<?php
  // Функция, возвращающая массив с годом, месяцем и днем
  function get_date($time)
```

```
  {
    return array(date("Y", $time), date("m", $time), date("d", $time));
  }
  // Получаем текущую дату
  $arr = get_date(time());
  // Выводим дамп массива $arr
  echo "<pre>";
  print_r($arr);
  echo "<pre>";
?>
```

В качестве аргумента функция `get_date()` принимает время в формате UNIXSTAMP, которое при помощи функции `date()` преобразует в три числа (год, месяц и день) составляющие элементы результирующего массива. В качестве результата, скрипт из листинга 3.43 выводит следующий дамп массива `$arr`:

```
Array
(
    [0] => 2010
    [1] => 05
    [2] => 12
)
```

Получение результата выполнения функции в виде массива не всегда удобно, особенно, если количество элементов в массиве не велико, или не все они требуются для дальнейшей работы. В PHP для преобразования элементов массива в переменные предназначена специальная конструкция `list()`. В листинге 3.44 демонстрируется преобразование результирующего массива функции `get_date()` в три переменные `$year`, `$month` и `$day`, соответствующие году, месяцу и дню.

Листинг 3.44. Использование конструкции `list()`

```
<?php
  // Функция, возвращающая массив с годом, месяцем и днем
  function get_date($time)
  {
    return array(date("Y", $time), date("m", $time), date("d", $time));
  }
  // Получаем текущую дату
  list($year, $month, $day) = get_date(time());
  echo "Год — $year<br />";     // Год — 2010
  echo "Месяц — $month<br />"; // Месяц — 05
  echo "День — $day<br />";     // День — 12
?>
```

Следует отметить, что конструкция `list()` работает только с числовыми массивами, нумерация индексов которых начинается с нуля. В листинге 3.45 приводится

альтернативная реализация функции `get_date()`, которая возвращает ассоциативный массив. Однако попытка использования конструкции `list()` для заполнения переменных `$year`, `$month` и `$day` приводит к выдаче замечания о том, что переменные не были инициализированы.

Листинг 3.45. Конструкция `list()` не работает с ассоциативными массивами

```
<?php
  // Функция, возвращающая массив с годом, месяцем и днем
  function get_date($time)
  {
    return array("year"  => date("Y", $time),
                 "month" => date("m", $time),
                 "day"   => date("d", $time));
  }

  // Получаем текущую дату
  list($year, $month, $day) = get_date(time());
  echo "Год — $year<br />";    // Notice: Undefined offset
  echo "Месяц — $month<br />"; // Notice: Undefined offset
  echo "День — $day<br />";    // Notice: Undefined offset
?>
```

Еще одной интересной особенностью конструкции `list()` является тот факт, что количество элементов в разбираемом массиве и в круглых скобках конструкции `list()` может не совпадать. Если элементов в массиве больше, чем нужно, лишние просто будут отброшены, если меньше — часть переменных останется неинициализированной. Более того, часть первых элементов массива может быть пропущена, для этого достаточно указать соответствующее количество запятых (листинг 3.46).

Листинг 3.46. Получаем лишь месяц

```
<?php
// Функция, возвращающая массив с годом, месяцем и днем
function get_date($time)
{
 return array(date("Y", $time), date("m", $time), date("d", $time));
}
// Получаем лишь месяц
list(, $month) = get_date(time());
echo "Месяц — $month<br />"; // Месяц — 05
?>
```

3.15. Сортировка массивов

PHP предоставляет разработчику широкие возможности для сортировки массивов (табл. 3.6).

Таблица 3.6. *Функции для сортировки массивов*

Функция	Описание
`sort ($array [, $sort_flags]) )`	Сортирует массив `$array` в прямом порядке. Параметр `$sort_flags` задает параметры сортировки: `SORT_REGULAR` — обычная сортировка; `SORT_NUMERIC` — сравнивать элементы как числа; `SORT_STRING` — сравнивать элементы как строки; `SORT_LOCALE_STRING` — сравнивать элементы как строки, основываясь на текущей локали
`rsort ($array [, $sort_flags])`	Сортирует массив `$array` в обратном порядке
`asort ($array [, $sort_flags])`	Сортирует массив `$array` в прямом порядке с сохранением отношения ключ-значение
`arsort ($array [, $sort_flags] )`	Сортирует массив `$array` в обратном порядке с сохранением отношения ключ-значение
`natsort ($array)`	Осуществляет "естественную" сортировку массива `$array`, учитывая регистр
`natcasesort ($array)`	Осуществляет "естественную" сортировку массива `$array`, без учета регистра
`ksort ($array [, $sort_flags])`	Сортирует массив `$array` в прямом порядке по ключам с сохранением отношения ключ-значение
`krsort ($array [, $sort_flags])`	Сортирует массив `$array` в обратном порядке по ключам с сохранением отношения ключ-значение
`usort ($array, $cmp_function)`	Сортирует массив `$array` с использованием пользовательской функции сравнения элементов `$cmp_function`
`uasort ($array, $cmp_function)`	Сортирует массив `$array` с использованием пользовательской функции сравнения элементов `$cmp_function` и сохранением отношения ключ-значение
`uksort ($array, $cmp_function)`	Сортирует массив `$array` по ключам с использованием пользовательской функции сравнения элементов `$cmp_function` и сохранением отношения ключ-значение
`array_multisort ($ar1 [, $arg [, ... [, ...]]])`	Сортирует один многомерный массив или подвергает сортировке несколько массивов сразу

Функция `sort()` позволяет сортировать массив `$arr` по возрастанию и имеет следующий синтаксис:

```
sort($arr [, $sort_flags])
```

Необязательный аргумент `$sort_flags` указывает, как именно должны сортироваться элементы (задает флаги сортировки). Допустимы следующие значения этого аргумента:

- `SORT_REGULAR` — задает нормальное сравнение элементов;
- `SORT_NUMERIC` — сравнивает элементы как числа;
- `SORT_STRING` — сравнивает элементы как строки;
- `SORT_LOCALE_STRING` — сравнивает элементы как строки, основываясь на текущей локали.

В листинге 3.47 приводится пример использования функции `sort()`.

Листинг 3.47. Сортировка массива по возрастанию

```
<?php
$number = array("2", "1", "4", "3","5"); // исходный массив
echo "до сортировки: <br>";
for($i=0; $i < count($number); $i++)
{
 echo "$number[$i] ";
}
sort($number); // сортируем массив по возрастанию
echo "<br> после сортировки: <br>";
for($i = 0; $i < count($number); $i++)
{
 echo "$number[$i] ";
}
?>
```

Результат:

```
до сортировки:
2 1 4 3 5
после сортировки:
1 2 3 4 5
```

Сортировка строк происходит по старшинству первой буквы в алфавите. Такую сортировку часто называют сортировкой в альфа-бета-порядке. К примеру, если имеется массив:

```
array("one",
  "two",
  "abs",
  "three",
  "uic",
  "for",
  "five"),
```

то применение к нему функции sort приведет к следующему результату:

```
[0] => abs
[1] => five
[2] => for
[3] => one
[4] => three
[5] => two
[6] => uic
```

Функция rsort() сортировки массива по убыванию имеет следующий синтаксис:

```
rsort ($arr [, $sort_flags])
```

Во всем остальном ведет себя аналогично функции sort(). В листинге 3.48 приводится пример использования функции rsort().

Листинг 3.48. Сортировка массива по убыванию

```
<?php
$number = array("2", "1", "4", "3","5"); // исходный массив
echo("до сортировки: <br>");
for($i=0; $i < count($number); $i++)
{
 echo "$number[$i] ";
}
rsort($number); // сортируем массив по убыванию
echo "<br> после сортировки: <br>";
for($i=0; $i < count($number); $i++)
{
 echo "$number[$i] ";
}
?>
```

Результат:

```
до сортировки:
2 1 4 3 5
после сортировки:
5 4 3 2 1
```

Функция asort() осуществляет сортировку ассоциативного массива по возрастанию и имеет следующий синтаксис:

```
asort ($arr [, $sort_flags])
```

Функция asort() сортирует массив $arr так, чтобы его значения шли в алфавитном (если это строки) или возрастающем (для чисел) порядке. Значения необязательного аргумента $sort_flags такие же, как и для функции sort(). Важное отличие этой функции от функции sort() состоит в том, что при применении функции asort()

сохраняются связи между ключами и соответствующими им значениями (листинг 3.49), чего нет в функции `sort()` (там эти связи попросту разрываются).

Листинг 3.49. Сортировка функцией `assort()`

```
<?php
  $arr = array("a" => "one",
               "b" => "two",
               "c" => "three",
               "d" => "four");
  asort($arr);
  foreach($arr as $key => $val)
  {
    echo " $key => $val ";
  }
?>
```

Результат:

```
d => four a => one c => three b => two
```

Как видно, связи "ключ-значение" сохранились при сортировке. При сортировке массива при помощи функции `sort()`, результат бы выглядел следующим образом:

```
0 => four 1 => one 2 => three 3 => two
```

Функция `arsort()` аналогична функции `asort()`, только она упорядочивает массив не по возрастанию, а по убыванию.

```
arsort($arr [, $sort_flags])
```

Функция `ksort()` осуществляет сортировку не по значениям, а по ключам массива в порядке их возрастания, при этом связи между ключами и значениями сохраняются. Функция имеет следующий синтаксис:

```
ksort ($arr [, $sort_flags])
```

Значения аргумента `$sort_flags` такие же, как и функции `sort()`. В листинге 3.50 приводится пример использования функции `ksort()`.

Листинг 3.50. Сортировка по индексам массива

```
<?php
  $arr = array("a" => "one",
               "d" => "four",
               "c" => "three",
               "b" => "two"
               );
  ksort($arr);
```

```
  foreach($arr as $key => $val)
  {
    echo (" $key => $val ");
  }
?>
```

Результат:

```
a => one b => two c => three d => four
```

Функция `krsort()` осуществляет сортировку элементов массива по ключам в обратном порядке (по убыванию). Во всем остальном аналогична функции `ksort()`. Функция имеет следующий синтаксис:

```
krsort ($arr [, $sort_flags])
```

Функция `natsort()` выполняет "естественную" сортировку массива и имеет следующий синтаксис:

```
natsort($arr)
```

Под естественной сортировкой понимается сортировка элементов таким образом, как их отсортировал бы человек. Приведем пример. Пусть у нас есть несколько файлов с именами

```
file1.txt
file10.txt
file2.txt
file12.txt
file20.txt
```

При обычной сортировке файлы будут расположены в следующем ("машинном") порядке:

```
file1.txt
file10.txt
file12.txt
file2.txt
file20.txt
```

Естественная же сортировка приведет к следующему результату:

```
file1.txt
file2.txt
file10.txt
file12.txt
file20.txt
```

Производится сортировка с помощью функции `natsort()`, принимающей в качестве параметра сортируемый массив (листинг 3.51).

Листинг 3.51. Естественная сортировка

```
<?php
$arr_first = $arr_second =
     array( "file12.txt",
         "file10.txt",
         "file2.txt",
         "file1.txt");
sort($arr_first);
echo "Обычная сортировка<br>";
echo "<pre>";
print_r($arr_first);
echo "</pre>";

natsort($arr_second);
echo "<br>Естественная сортировка<br>";
echo "<pre>";
print_r($arr_second);
echo "</pre>";
?>
```

Результат:

```
Обычная сортировка
Array
(
    [0] => file1.txt
    [1] => file10.txt
    [2] => file12.txt
    [3] => file2.txt
)

Естественная сортировка
Array
(
    [3] => file1.txt
    [2] => file2.txt
    [1] => file10.txt
    [0] => file12.txt
)
```

Функция `array_multisort()` может быть использована для сортировки сразу нескольких массивов или одного многомерного массива. Функция имеет следующий синтаксис:

```
array_multisort($ar1 [,$arg [, ... [, ...]]] )
```

Массивы, которые передаются функции `array_multisort()`, должны содержать одинаковое количество аргументов и все вместе воспринимаются как своеобразная

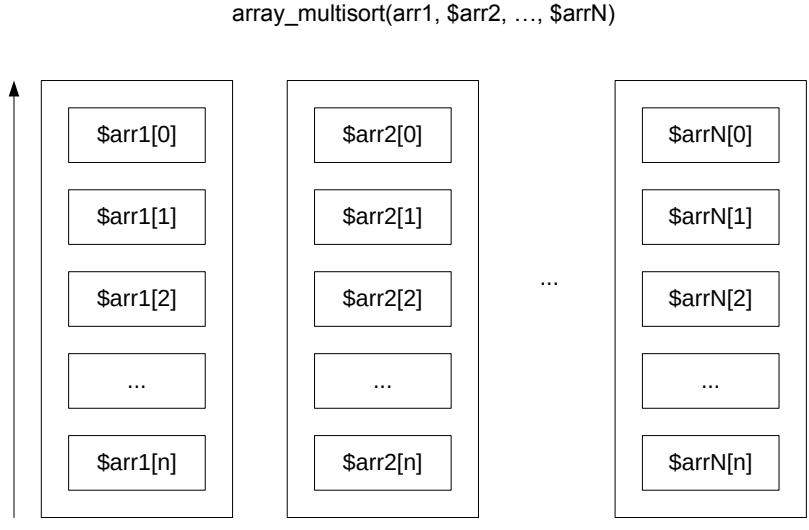


Рис. 3.2. Аргументы функции `array_multisort()` образуют таблицу

таблица. Сортировке подвергается лишь первый массив, а значения последующих массивов выстраиваются в соответствии с ним (рис. 3.2).

В листинге 3.52 приводится пример сортировки двух массивов `$arr1` и `$arr2`.

Листинг 3.52. Использование функции `array_multisort()`

```
<?php
  $arr1 = array(3, 4, 2, 7, 1, 5);
  $arr2 = array("world", "Hello", "yes", "no", "apple", "wet");
  array_multisort($arr1, $arr2);
  echo "<pre>";
  print_r($arr1);
  print_r($arr2);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 3.52 будут следующие дампы массивов:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
    [5] => 7
)
```

```
Array
(
    [0] => apple
    [1] => yes
    [2] => world
    [3] => Hello
    [4] => wet
    [5] => no
)
```

Как видно из результатов, первый массив был отсортирован по возрастанию, а элементы второго массива выстроены в соответствии с первым так, как если бы между элементами двух массивов существовала связь.

Помимо массивов функция `array_multisort()` в качестве аргументов может принимать константы, задающие порядок сортировки (табл. 3.7).

> ***Замечание***
>
> Группы констант `SORT_ASC` и `SORT_DESC`, а также `SORT_REGULAR`, `SORT_NUMERIC` и `SORT_STRING` являются взаимоисключающими.

Таблица 3.7. *Константы, определяющие поведение функции* `array_multisort()`

Константа	Описание
`SORT_ASC`	Сортировать в возрастающем порядке
`SORT_DESC`	Сортировать в убывающем порядке
`SORT_REGULAR`	Сравнивать элементы обычным образом
`SORT_NUMERIC`	Сравнивать элементы в предположении, что это числа
`SORT_STRING`	Сравнивать элементы в предположении, что это строки

В листинге 3.53 приводится модифицированный вариант скрипта, сортирующий таблицу в обратном порядке.

Листинг 3.53. Использование констант

```
<?php
  $arr1 = array(3, 4, 2, 7, 1, 5);
  $arr2 = array("world", "Hello", "yes", "no", "apple", "wet");
  array_multisort($arr1, SORT_DESC, SORT_NUMERIC, $arr2);
  echo "<pre>";
  print_r($arr1);
  print_r($arr2);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 3.53 будут следующие дампы массивов:

```
Array
(
    [0] => 7
    [1] => 5
    [2] => 4
    [3] => 3
    [4] => 2
    [5] => 1
)
Array
(
    [0] => no
    [1] => wet
    [2] => Hello
    [3] => world
    [4] => yes
    [5] => apple
)
```

3.16. Вывод иерархических данных

Для организации сложных структур не обязательно создавать массивы глубокой степени вложенности. Сложные иерархические данные (такие как сообщения на форуме или в комментариях в виде лестницы), меню сайта можно создать в рамках двумерного массива, предусмотрев отдельный элемент для указания предка. На рис. 3.3 представлен типичная лестничная структура, где ответ под сообщением немного сдвинут вправо.

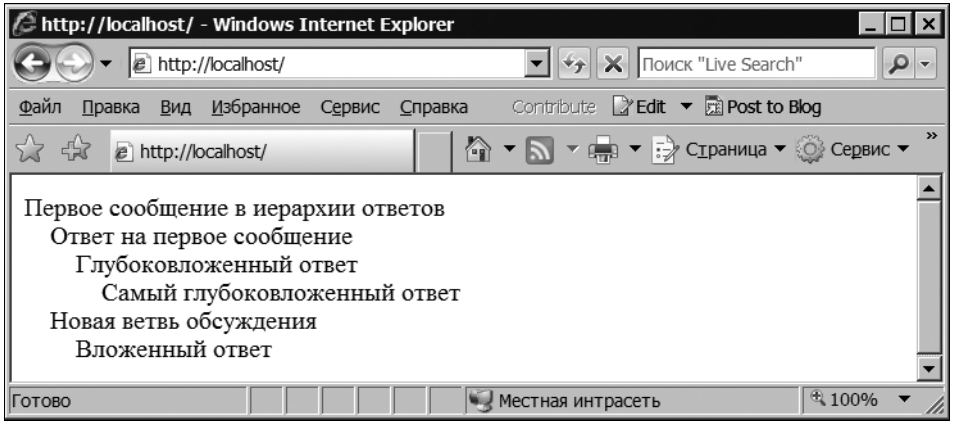


Рис. 3.3. Лестничная структура ответов

Для формирования такой структуры достаточно двумерного массива `$comments`, представленного в листинге 3.54. Каждый элемент такого массива, помимо текста сообщения в элементе с ключом `"text"`, в дополнительном элементе с ключом `"id_parent"` содержит ссылку на индекс родительского сообщения.

Листинг 3.54. Массив для лестничной структуры

```
<?php
  $comments = array(
    1 => array(
      "text" => "Первое сообщение в иерархии ответов",
      "id_parent" => 0),
    2 => array(
      "text" => "Ответ на первое сообщение",
      "id_parent" => 1),
    3 => array(
      "text" => "Глубоковложенный ответ",
      "id_parent" => 2),
    4 => array(
      "text" => "Новая ветвь обсуждения",
      "id_parent" => 1),
    5 => array(
      "text" => "Вложенный ответ",
      "id_parent" => 4),
    6 => array(
      "text" => "Самый глубоковложенный ответ",
      "id_parent" => 3)
  );
?>
```

Для удобства оперирования элементами из массива `$comments`, как правило, создают дополнительный двумерный массив `$parents`, который соотносит индексы родительских сообщений и ответов на них (листинг 3.55).

Листинг 3.55. Вспомогательный массив `$parents`

```
<?php
  // Формируем массив родителей и их потомков
  $parents = array();
  foreach($comments as $id => $elements)
  {
    $parents[$elements['id_parent']][] = $id;
  }
  echo "<pre>";
  print_r($parents);
  echo "</pre>";
?>
```

Массив $parents служит своеобразным справочником, запросив в котором индекс сообщения, можно узнать индексы всех непосредственных ответов на него. Дамп массива $parents для созданного ранее массива $comments представлен далее:

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

Теперь можно приступить к выводу иерархии сообщений. Для этого, как правило, создают рекурсивную функцию (листинг 3.56). Для удобства и снижения издержек массивы $comments и $parents объявляются внутри функции глобальными при помощи ключевого слова global, что позволяет не передавать их через параметры, а использовать напрямую.

Листинг 3.56. Рекурсивная функция вывода иерархии сообщений

```
<?php
  ...
  // $id_parent — индекс родительского сообщения
  // $indent    — отступ
  function print_hierarchy($id_parent = 0, $indent = "")
  {
    // Объявляем внешние массив $comments и $parents
    // глобальным и доступным для использования
    // внутри функции
    global $comments, $parents;
```

```
    // Проверяем, имеются ли у элемента с индексом
    // $id_parent потомки, иначе покидаем функцию
    if(!isset($parents[$id_parent])) return;
    // Выводим все элементы массива с родителем,
    // имеющим индекс $id_parent
    for($i = 0; $i < count($parents[$id_parent]); $i++)
    {
      // Выводим сообщение с отступом
      echo "<div style='padding-left:{$indent}px'>".
           $comments[$parents[$id_parent][$i]]['text']."</div>";
      // Выводим иерархию ответов для текущего
      // сообщения
      print_hierarchy($parents[$id_parent][$i], $indent + 20);
    }
  }
  // Выводим сообщения, начиная с первого
  print_hierarchy();
?>
```

Функция `print_hierarchy()` из листинга 3.56 принимает два необязательных параметра, первый из которых `$id_parent` передает индекс сообщения. Если параметр не указан или принимает значение 0, вывод начинается с первого сообщения иерархии. Второй параметр `$indent` передает целое число пикселов, на которое необходимо сдвинуть ответы вправо относительно родительского сообщения. Сдвиг осуществляется при помощи CSS-параметра `padding-left`. Для каждого из выводимых сообщений, функция вызывает сама себя, передавая новый индекс и увеличивая значения отступа на 20 пикселов. Результат выполнения скрипта из листинга 3.56 представлен на рис. 3.3.

3.17. Суперглобальные массивы

Помимо пользовательских PHP предоставляет разработчику ряд предопределенных массивов, заполнение которых осуществляется ядром PHP и Web-сервером. Такие массивы изначально являются глобальными, и их не требуется объявлять при помощи ключевого слова `global`. Наличие таких массивов позволяет значительно упростить Web-разработку и быстро сделало PHP одним из самых популярных языков программирования для Web. В табл. 3.8 представлены краткие характеристики суперглобальных массивов.

Таблица 3.8. *Суперглобальные массивы*

Массив	Описание
`$_GET`	Содержит параметры, переданные скрипту через строку запроса (метод GET)
`$_POST`	Содержит параметры, переданные скрипту методом POST (в теле HTTP-запроса)
`$_FILES`	Содержит информацию о загруженных методом POST на сервер файлах

Таблица 3.8 *(окончание)*

Массив	Описание
`$_COOKIE`	Содержит параметры, сохраненные при помощи HTTP-заголовка Set-Cookie на стороне клиента
`$_REQUEST`	Содержит параметры суперглобальных массивов `$_GET`, `$_POST` и `$_COOKIE`
`$_SESSION`	Содержит набор параметров, закрепленных за каждым конкретным клиентом и сохраненных на стороне сервера
`$_SERVER`	Содержит переменные окружения, установленные Web-сервером, а также вспомогательную информацию по HTTP-заголовкам и путям к скрипту
`$_ENV`	Содержит переменные окружения, вместо этого суперглобального массива рекомендуется использовать `$_SERVER`
`$GOBALS`	Содержит ссылки на все переменные в глобальной видимости (предопределенные массивы), переменные, объявленные вне функций, и переменные функций, объявленных с использованием ключевого слова `global`

В следующих разделах суперглобальные переменные рассматриваются более подробно.

3.18. Суперглобальный массив *$_GET*

GET-параметры передаются в строке запроса после символа вопроса (`?`).

http://localhost/index.php?id=1

В приведенном адресе URL последовательность `id=1` задает GET-параметр с именем `id` и значением 1. GET-параметры автоматически помещаются в суперглобальный массив `$_GET`. Имена параметров выступают в качестве ключей массива. В листинге 3.57 выводится значение GET-параметра `id`.

> ***Замечание***
>
> Стандарт HTTP, в отличие от PHP, допускает использование в качестве имени GET-параметра последовательности, начинающиеся с цифры или содержащие тире. Такие параметры не попадают в суперглобальный массив `$_GET`, если отказаться от их использования нет никакой возможности, их значения следует самостоятельно извлекать из переменной окружения `$_SERVER['QUERY_STRING']`, содержащей GET-параметры.

Листинг 3.57. Извлечение GET-параметра `id_forum`

```
<?php
  echo $_GET['id']; // 1
?>
```

Если имеется необходимость передать скрипту одновременно несколько GET-параметров, они разделяются символом амперсанда `&`:

http://localhost/index.php?id =1&number=2&page=10

В приведенном выше URL передаются три GET-параметра: `id` со значением 1, `number` со значением 2 и `page` со значением 10 (листинг 3.58).

Листинг 3.58. Вывод дампа суперглобального массива `$_GET`

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

В результате выполнения скрипта из листинга 3.58 может быть выведен следующий дамп массива `$_GET`:

```
Array
(
    [id] => 1
    [number] => 2
    [page] => 10
)
```

В качестве GET-параметров могут выступать элементы массива, в этом случае суперглобальный массив `$_GET` становится двумерным. Так, для строки запроса

http://localohost/index.php?id[]=1&id[]=2&id[]=10

скрипт из листинга 3.58 выведет следующий дамп:

```
Array
(
    [id] => Array
        (
            [0] => 1
            [1] => 2
            [2] => 10
        )
)
```

Здесь элементам были автоматически назначены числовые индексы, начиная с 0, однако индексы и ключи могут назначаться элементам непосредственно в строке запроса. Для запроса

http://localhost/index.php?id[a]=1&id[b]=2&id[c]=10

скрипт из листинга 3.58 выведет следующий дамп:

```
Array
(
    [id] => Array
        (
            [a] => 1
```

```
            [b] => 2
            [c] => 10
        )
)
```

GET-параметры и их значения могут содержать недопустимые с точки зрения URL символы (пробелы, русские символы), которые при формировании URL обязательно должны преобразовываться в безопасный формат. Для такого преобразования в PHP предусмотрены специальные функции (табл. 3.9).

Таблица 3.9. *Функции для работы с URL*

Функция	Описание
`urlencode($str)`	Возвращает строку `$str`, в которой все символы, отличные от чисел, букв английского алфавита и символа подчеркивания, кодируются двумя шестнадцатеричными числами, предваренными символом `%`. В отличие от всех остальных символов пробел кодируется плюсом `+`
`urldecode($str)`	Обратная функция `urlencode()`, осуществляющая декодирование символов, начинающихся с `%`
`rawurlencode($str)`	Возвращает строку `$str`, в которой все символы, отличные от чисел, букв английского алфавита и символа подчеркивания, кодируются двумя шестнадцатеричными числами, предваренными символом `%`. Функция аналогична `urlencode()`, однако пробел кодируется не символом плюса `+`, а последовательностью `%20`
`rawurldecode($str)`	Обратная `rawurlencode()` функция, осуществляющая декодирование символов, начинающихся с `%`
`parse_url($url [, $component])`	Разбирает строку запроса `$url` на отдельные компоненты, которые возвращаются как элементы ассоциативного массива. Если необязательный параметр `$component` принимает одну из следующих констант: `PHP_URL_SCHEME`, `PHP_URL_HOST`, `PHP_URL_PORT`, `PHP_URL_USER`, `PHP_URL_PASS`, `PHP_URL_PATH`, `PHP_URL_QUERY` или `PHP_URL_FRAGMENT`, то один из компонентов возвращается в виде строки
`http_build_query ($formdata [, $numeric_prefix [, $arg_separator]])`	Позволяет сформировать строку запроса из элементов массива `$formdata`. Если указан необязательный параметр `$numeric_prefix`, он добавляется в качестве префикса GET-параметрам. Необязательный параметр `$arg_separator` позволяет задать разделитель между GET-параметрами, отличный от амперсанда `&`

В листинге 3.59 формируется ссылка на файл test.php, через GET-параметр `phrase` которому передается фраза "Привет, мир!".

Листинг 3.59. Использование функции `urlencode()`

```
<?php
  echo "<a href='test.php?phrase=".
          urlencode("Привет, мир!")."'>ссылка</a>";
?>
```

Исходный код результирующей HTML-страницы будет содержать следующую строку:

```
<a href='test.php?phrase=%CF%F0%E8%E2%E5%F2%2C+%EC%E8%F0%21'>ссылка</a>
```

Помимо GET-параметров в скрипте может понадобиться информация о текущей странице, номере порта, хосте и т. п. Для разбора строки запроса предназначена специальная функция `parse_url()`, которая принимает в качестве первого параметра строку запроса и возвращает отдельные его компоненты в виде ассоциативного массива со следующими ключами:

- `scheme` — префикс, например, http, https, ftp и т. п.;
- `host` — домен;
- `port` — номер порта;
- `user` — пользователь;
- `pass` — его пароль;
- `path` — путь от корневого каталога;
- `query` — все, что расположено после символа вопроса (`?`);
- `fragment` — все, что расположено после символа `#`.

В листинге 3.60 приводится пример разбора URL при помощи функции `parse_url()`.

Листинг 3.60. Использование функции `parse_url()`

```
<?php
  $url = 'http://user:pass@www.site.ru/path/index.php?par=value#anch';
  $arr = parse_url($url);

  echo "<pre>";
  print_r($arr);
  echo "<pre>";
?>
```

Результатом выполнения скрипта будет следующий дамп массива `$arr`:

```
Array
(
    [scheme] => http
    [host] => www.site.ru
    [user] => user
    [pass] => pass
    [path] => /path/index.php
    [query] => par=value
    [fragment] => anch
)
```

Если функция parse_url() принимает второй параметр, вместо массива возвращается строка с одним из компонентов строки запроса. Параметр может принимать следующие константы:

- PHP_URL_SCHEME — префикс, например, http, https, ftp и т. п.;
- PHP_URL_HOST — домен;
- PHP_URL_PORT — номер порта;
- PHP_URL_USER — пользователь;
- PHP_URL_PASS — его пароль;
- PHP_URL_PATH — путь от корневого каталога;
- PHP_URL_QUERY — все, что расположено после символа вопроса (?);
- PHP_URL_FRAGMENT — все, что расположено после символа #.

В листинге 3.61 из строки запроса при помощи функции parse_url() и константы PHP_URL_HOST извлекается лишь доменное имя.

Листинг 3.61. Использование второго параметра в функции parse_url()

```
<?php
  $url = 'http://user:pass@www.site.ru/path/index.php?par=value#anch';
  echo parse_url($url, PHP_URL_HOST); // www.site.ru
?>
```

3.19. Постраничная навигация

Количество элементов в массиве может принимать огромные значения. Для того чтобы предоставить пользователю возможность удобно просматривать значения при выводе содержимого массива на печать, часто прибегают к разбиению результата на отдельные страницы. Организацию набора ссылок для удобного просмотра такого набора страниц называют *постраничной навигацией*.

Пусть имеется массив, содержащий список языков программирования (листинг 3.62), причем список необходимо вывести таким образом, чтобы на одной странице размещалось по два элемента массива. Для решения этой задачи необходимо весь массив разбить на пары и передавать номер текущей пары через GET-параметр (в примере использован GET-параметр с именем page). В зависимости от переданного номера скрипт вычисляет начальный $start и конченый $end индексы, между которыми необходимо вывести элементы массива на текущей странице.

Листинг 3.62. Постраничная навигация

```
<?php
  $language[] = "PHP";
  $language[] = "C++";
  $language[] = "Java";
```

```
$language[] = "Ruby";
$language[] = "Python";
$language[] = "Perl";
$language[] = "JavaScript";
$language[] = "Visual Basic";
$language[] = "Fortran";
$language[] = "Pascal";
$language[] = "Assembler";
$language[] = "Lisp";
$language[] = "Haskell";
$language[] = "C#";

// Определяем количество элементов на одной странице
$pnumber = 2;

// Проверяем, передан ли номер текущей страницы
if(isset($_GET['page'])) $page = intval($_GET['page']);
else $page = 1;

// Количество элементов в массиве
$total = count($language);
// Вычисляем количество страниц
$number = (int)($total/$pnumber);
if((float)($total/$pnumber) — $number != 0) $number++;

// Начальный индекс массива $language
// для вывода на текущей странице
$start = (($page — 1)*$pnumber + 1);
// Конечный индекс массива $language
// для вывода на текущей странице
$end = $page*$pnumber + 1;
if($end > $total) $end = $total;

// Выводим содержимое страниц
for($i = $start; $i < $end; $i++)
{
  echo $language[$i]."<br />";
}

// Постраничная навигация
for($i = 1; $i <= $number; $i++)
{
  // Если это произвольная страница
  if($i != $number)
  {
    if($page == $i)
```

```
      {
        // Текущую страницу не подсвечиваем ссылкой
        echo "[".(($i — 1)*$pnumber + 1)."-".$i*$pnumber."] ";
      }
      else
      {
        echo "<a href='index.php?page=$i'>[".
             (($i — 1)*$pnumber + 1)."-".$i*$pnumber."]</a> ";
      }
    }
    // Если это последняя страница, заменяем последнюю цифру
    // максимальным числом позиций в массиве $temp
    else
    {
      if($page == $i)
      {
        // Текущую страницу не подсвечиваем ссылкой
        echo "[".(($i — 1)*$pnumber + 1)."-".($total — 1)."] ";
      }
      else
      {
        echo "<a href='index.php?page=$i'>[".
             (($i — 1)*$pnumber + 1)."-".($total — 1)."]</a> ";
      }
    }
  }
?>
```

Результат работы скрипта из листинга 3.62 представлен на рис. 3.4.

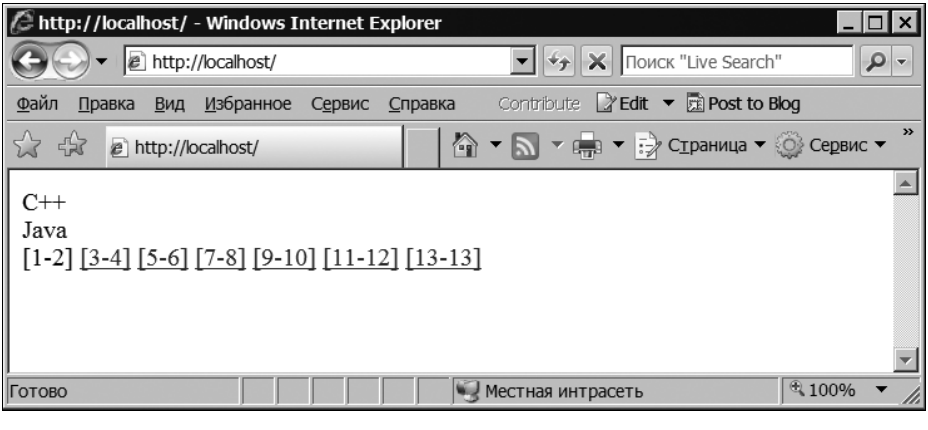


Рис. 3.4. Постраничная навигация

3.20. Суперглобальный массив *$_POST*

Передача параметров метода GET имеет ряд ограничений, в частности количество символов отводимых под строку запроса, ограничено несколькими килобайтами, кроме того, при формировании строки запроса не все символы допустимы. Если на сервер необходимо передать большой объем данных, содержащий помимо текста спецсимволы, прибегают к методу POST, который в отличие от метода GET передает параметры не в HTTP-заголовке, а в теле HTTP-документа.

Самый простой способ отправить что-то методом POST на сервер — это сформировать HTML-форму, поместив в тег `<form>` атрибут `method` со значением "post" (листинг 3.63). HTML-форма содержит три элемента управления: два текстовых поля с именами `first` и `second` и кнопку, позволяющую отправить данные.

Листинг 3.63. HTML-форма

```
<form action='index.php' method='post'>
  <input type='text' name='first' /><br>
  <input type='text' name='second' /><br>
  <input type='submit' value="Отправить">
</form>
```

В атрибуте `action` тега `<form>` указывается обработчик, которому отправляются данные. Если данный атрибут оставить незаполненным или присвоить ему имя текущего файла, то данные будут отправлены скрипту, где расположена форма.

Для того чтобы получить данные после их отправки, следует обратиться к суперглобальному массиву `$_POST`. В листинге 3.64 приводится модифицированная HTML-форма, перед которой расположен обработчик.

Листинг 3.64. Использование суперглобального массива `$_POST`

```
<?php
  // Обработчик HTML-формы
  if(!empty($_POST))
  {
    echo "<pre>";
    print_r($_POST);
    echo "</pre>";
    exit();
  }
?>
<form action='index.php' method='post'>
  <input type='text' name='first' /><br>
  <input type='text' name='second' /><br>
  <input type='submit' value="Отправить">
</form>
```

Обработчик проверяет, заполнен ли суперглобальный массив $_POST. Если он пуст, отображается HTML-форма, а если массив содержит какие-то данные, выводится его дамп и работа скрипта прекращается.

3.21. Передача файлов на сервер. Суперглобальный массив *$_FILES*

Для загрузки файлов на сервер также используется метод POST, однако содержимое файла не помещается в массив $_POST, файлы помещаются во временную папку, а информация о них помещается в специальный суперглобальный массив $_FILES.

Для того чтобы пользователь имел возможность загрузить файл на сервер, в HTML-форму вставляется специальный элемент управления, позволяющий указать путь к загружаемому файлу (при помощи кнопки **Обзор**). Элемент управления имеет следующий синтаксис:

```
<input type='file' />
```

Помимо атрибута type элемент управления допускает указания атрибутов name, задающего имя поля. Простая форма для отправки файла на сервер демонстрируется в листинге 3.65.

Листинг 3.65. HTML-форма для загрузки файлов на сервер (index.html)

```
<html><head><title> Загрузка файлов на сервер </title></head><body>
  <h2><b> Форма для загрузки файлов </b></h2>
   <form action="upload.php" method="post" enctype="multipart/form-data">
   <input type="file" name="filename"><br>
   <input type="submit" value="Загрузить"><br>
   </form>
 </body>
</html>
```

Атрибут entype формы определяет вид кодировки, которую браузер применяет к параметрам формы. Для того чтобы отправка файлов на сервер действовала, атрибуту entype необходимо присвоить значение multipart/form-data. По умолчанию этот атрибут имеет значение application/x-www-form-urlencoded. Если этот атрибут не задан, файл загружен не будет.

Если все сделано правильно, форма для отправки файлов на сервер должна выглядеть так, как это показано на рис. 3.5.

После того как получен HTTP-запрос, содержимое загруженного файла записывается во временный файл, который создается в каталоге сервера, заданном по умолчанию для временных файлов, если другой каталог не задан в файле php.ini (директива upload_tmp_dir).

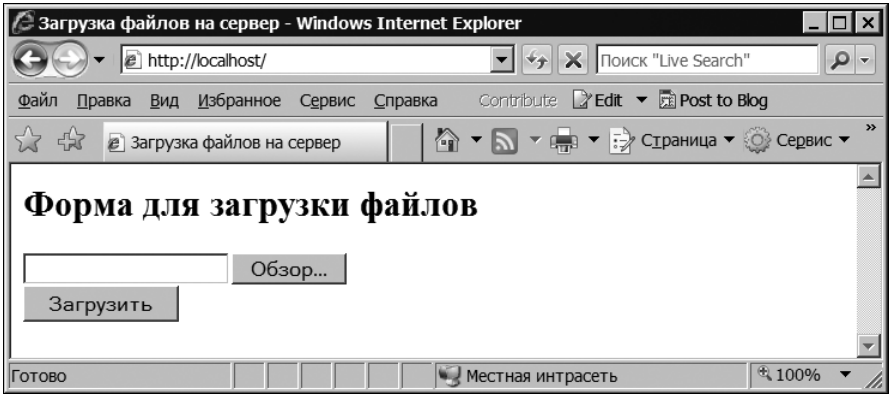


Рис. 3.5. Загрузка файла на сервер

Суперглобальный массив `$_FILES` является двумерным, в первых квадратных скобках указывается название элемента управления `file`, второй ключ определяет различные характеристики от названия до размера:

- `$_FILES['filename']['name']` (содержит исходное имя файла на клиентской машине);
- `$_FILES['filename']['size']` (содержит размер загруженного файла в байтах);
- `$_FILES['filename']['type']` (содержит MIME-тип файла);
- `$_FILES['filename']['tmp_name']` (содержит имя временного файла, в который сохраняется загруженный файл).

В листинге 3.66 приведен скрипт upload.php, который загружает файл на сервер и копирует его из временного каталога в каталог temp.

> ***Замечание***
>
> Проверить успешность загрузки файла на сервер можно при помощи специальной функции `is_uploaded_file()`, которая принимает в качестве единственного параметра имя файла (`$_FILES['filename']['name']`) и возвращает `TRUE` в случае успешной загрузки и `FALSE` в случае неудачи.

Листинг 3.66. Загрузка файлов на сервер (upload.php)

```
<html>
<head>
<title> Результат загрузки файла </title>
</head>
<body>
<?php
  if (move_uploaded_file($_FILES['filename']['tmp_name'],
     "temp/".$_FILES['filename']['name']))
  {
    echo "Файл успешно загружен";
  }
```

```
  else
  {
   echo "Ошибка загрузки файла";
  }
?>
</body>
</html>
```

Для перемещения из временной папки в папку назначения используется специальная функция `move_uploaded_file()`, которая принимает в качестве первого значения путь к файлу во временной папке, а в качестве второго — путь назначения файла, куда обработчик должен его переместить.

После выполнения этого скрипта выбранный для загрузки файл будет помещен в подкаталог temp каталога, в котором расположен скрипт, а браузер выдаст фразу "Файл успешно загружен".

Скрипт из листинга 3.67 позволяет вывести характеристики загруженного файла.

Листинг 3.67. Характеристики файла

```
<html>
<head>
<title> Результат загрузки файла </title>
</head>
<body>
<?php
  if (move_uploaded_file($_FILES["filename"]["tmp_name"],
     "temp/".$_FILES["filename"]["name"]))
  {
    echo "Файл успешно загружен <br>";
    // далее выводится информация о файле
    echo "Характеристики файла: <br>";
    echo "Имя файла: ";
    echo $_FILES["filename"]["name"];
    echo "<br>Размер файла: ";
    echo $_FILES["filename"]["size"];
    echo "<br>Каталог для загрузки: ";
    echo $_FILES["filename"]["tmp_name"];
    echo "<br>Тип файла: ";
    echo $_FILES["filename"]["type"];
  }
  else
  {
    echo "Ошибка загрузки файла";
  }
?>
</body>
</html>
```

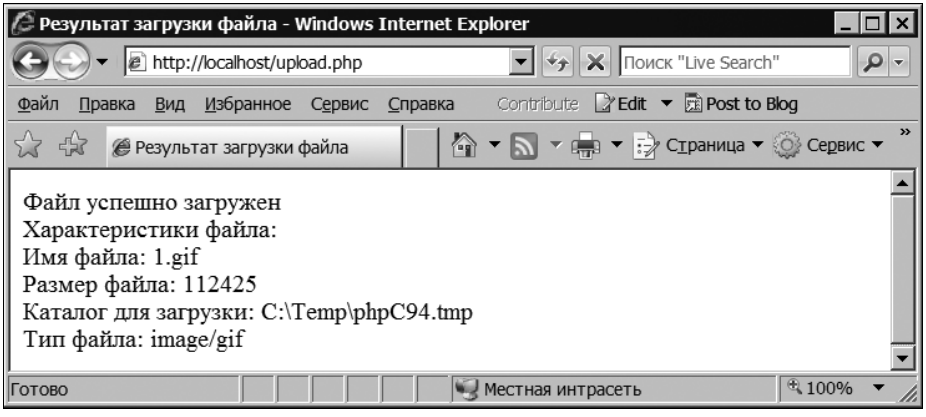


Рис. 3.6. Информация о загруженном файле

Результат выглядит так, как представлено на рис. 3.6.

В некоторых случаях требуется ограничить размер файла, который может быть загружен на сервер. К примеру, чтобы разрешить загрузку на сервер файлов размером не более 3 Мбайт, следует изменить скрипт так, как это представлено в листинге 3.68.

Листинг 3.68. Ограничение размера загружаемого файла

```
<?php
  if ($_FILES['filename']['size'] > 3*1024*1024)
  {
    exit "Размер файла превышает три мегабайта";
  }
  if (copy(_FILES['filename']['tmp_name'],
     "temp/".$_FILES['filename']['name']))
  {
    echo "Файл успешно загружен <br>";
  }
  else
  {
    echo "Ошибка загрузки файла";
  }
?>
```

Максимальный размер загружаемого файла можно также задать при помощи директивы `upload_max_filesize`, значение которой по умолчанию равно 2 Мбайт:

```
if ($_FILES['filename']['size'] > upload_max_filesize)
```

Замечание

Значение директивы `upload_max_filesize` можно изменить в конфигурационном файле php.ini.

3.22. Загрузка произвольного количества файлов

Одной из распространенных задач является создание динамических Web-форм, которые позволяют посетителю самому выбрать число полей в форме (рис. 3.7).

ЗАМЕЧАНИЕ

В *главе 9* приводится AJAX-версия данного приложения.

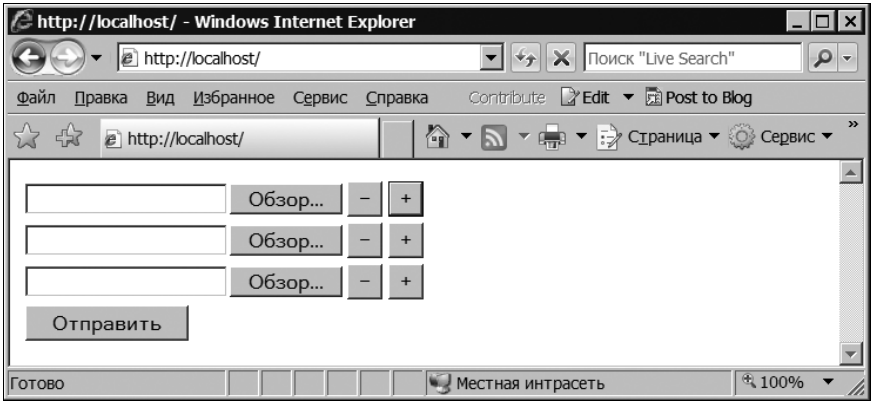


Рис. 3.7. HTML-форма для загрузки произвольного числа файлов на сервер

Для того что обеспечить произвольное число полей под загружаемые файлы, необходимо воспользоваться клиентским языком JavaScript, который позволит динамически сформировать HTML-форму (листинг 3.69). Для работы скрипта необходимо, чтобы в том же каталоге, где расположено Web-приложение, имелся подкаталог files, и права доступа были настроены так, чтобы скрипт имел право записи в подкаталог.

ЗАМЕЧАНИЕ

Web-приложение строится таким образом, чтобы и HTML-форма, и обработчик находились в одном файле. Вообще допускается разделение HTML-формы и обработчика.

Листинг 3.69. Загрузка произвольного количества файлов на сервер

```
<?php
  if(!empty($_FILES))
  {
    // Обработчик HTML-формы
    // Загружаем все файлы на сервер
    for($i = 0; $i < count($_FILES['filename']['name']); $i++)
    {
      // Перемещаем файл из временного каталога сервера
      // в каталог /files Web-приложения
      if (!move_uploaded_file($_FILES['filename']['tmp_name'][$i],
               "files/".$_FILES['filename']['name'][$i]))
```

```
      {
        echo "Ошибка загрузки файла";
      }
    }

    // Осуществляем автоматическую перезагрузку страницы,
    // если содержимое суперглобального массива $_POST
    // не является пустым
    header("Location: index.php");
    exit();
  }
?>
<form action="index.php" method="post" enctype="multipart/form-data">
<table>
<tr>
  <td><input class='files' type='file' name='filename[]' /></td>
  <td><input type='button' name='drop'
             value=' &minus; ' onclick='dropline(this);' />
      <input type='button' value=' + '
             onclick='addline(this);' /></td></tr>
<tr><td><input type='submit' value='Отправить' /></td><td></td></tr>
</table>
</form>
<script language='JavaScript1.1' type='text/javascript'>
<!--
function dropline(btn)
{
  if(document.getElementById)
  {
    while (btn.tagName != 'TR') btn = btn.parentNode;
    btn.parentNode.removeChild(btn);
  }
}
function addline(btn)
{
  if(document.getElementById)
  {
    while (btn.tagName != 'TR') btn = btn.parentNode;
    var newTr = btn.parentNode.insertBefore(
                  btn.cloneNode(true),
                  btn.nextSibling);
    thisChilds = newTr.getElementsByTagName('td');
    for (var i = 0; i < thisChilds.length; i++)
    {
      if (thisChilds[i].className == 'files')
      thisChilds[i].innerHTML =
        '<input class="files" type="file" name="filename[]" />';
```

```
    }
  }
}
//-->
</script>
```

HTML-форма состоит из произвольного количества полей типа `file`. Если в качестве элемента управления HTML-формы выступает массив, в обработчике формы значения данного элемента можно получить, обратившись к суперглобальному элементу `$_POST['имя_массива'][]`. В HTML-форме все поля типа `file` объединяются в массив под именем `att[]`, доступ к которому в обработчике осуществляется при помощи массива `$_POST['att']`. Индексы массива нумеруются с 0. Это приводит к тому, что обычно двумерный массив `$_FILES` превращается в трехмерный, и обработка файлов производится в цикле. При помощи функции `move_uploaded_file()` файл перемещается из временного каталога в каталог назначения `files`.

3.23. Cookie. Суперглобальный массив *$_COOKIE*

HTTP-протокол, лежащий в основе Интернета, не сохраняет информацию о состоянии сеанса. Это означает, что любое обращение клиента сервер воспринимает как обращение нового клиента, и даже если клиент формирует запрос для загрузки картинок с текущей страницы, сервером он воспринимается как запрос нового клиента, никак не связанного с тем, который только что загрузил страницу. Данная схема достаточно хорошо работала для статических страниц, но стала совершенно неприемлемой для динамических. В связи с этим в протокол HTTP были введены механизмы cookie, который в настоящий момент поддерживают все участники Интернета: клиенты, прокси-серверы и конечные серверы.

Cookies — это небольшие файлы, сохраняемые просматриваемыми серверами на машине посетителя, содержащие текстовую информацию о настройках пользователя, доступную для считывания создавшему их серверу. Слово "cookie" означает кекс, или сладкий бонус, выдаваемый клиентам ресторана, чтобы они запомнили его и посетили вторично. Для cookie так и не было подобрано адекватного русского перевода, поэтому в данной и последующих главах будет использоваться английское название.

Для создания cookie предназначена функция `setcookie()`, которая имеет следующий синтаксис:

```
setcookie($name [, $value [, $expire [, $path [, $domain [, $secure]]]]])
```

Функция принимает следующие аргументы:

- `$name` — имя cookie;
- `$value` — значение, хранящееся в cookie с именем `name`;
- `$expire` — время в секундах, прошедшее с 0 часов 00 минут 1 января 1970 г. По истечении этого времени cookie удаляется с машины клиента;

- $path — путь, по которому доступен cookie;
- $domain — домен, из которого доступен cookie;
- $secure — директива, определяющая, доступен ли файл cookie по защищенному протоколу HTPPS. По умолчанию эта директива имеет значение 0, что означает возможность доступа к cookie по стандартному незащищенному протоколу HTTP.

Функция setcookie() возвращает TRUE при успешной установке cookie на машине клиента и FALSE — в противном случае. После того как cookie установлен, его значение можно получить на всех страницах Web-приложения, обращаясь к суперглобальному массиву $_COOKIE и используя в качестве ключа имя cookie.

Так как cookie передается в заголовке HTTP-запроса, то вызов функции setcookie() необходимо размещать до начала вывода информации в окно браузера функциями echo(), print() и т. п., а также до включения в файл HTML-тегов. Работа с cookie без установки времени жизни продемонстрирована в листинге 3.70. Получить доступ к установленной на клиентской машине cookie можно через суперглобальный массив $_COOKIES.

> ***Замечание***
>
> Файл на жестком диске для cookie создается только в том случае, если выставляется время жизни cookie; в противном случае cookie действует только до конца сеанса, т. е. до того момента, пока пользователь не закроет окно браузера. В этом случае говорят о *сессионном cookie*.

Листинг 3.70. Подсчет количества обращений к странице

```
<?php
  // Выставляем уровень обработки ошибок
  error_reporting(E_ALL & ~E_NOTICE);

  // Увеличиваем значение cookie
  $_COOKIE['counter']++;
  // Устанавливаем cookie
  setcookie("counter", $_COOKIE['counter']);
  // Выводим значение cookie
  echo "Вы посетили эту страницу $_COOKIE[counter] раз";
?>
```

Результат выполнения скрипта, приведенного в листинге 3.70, показан на рис. 3.8.

В листинге 3.70 реализована установка cookie с именем counter, значение которого увеличивается при каждой перезагрузке страницы.

Время жизни cookie задается при помощи функций time() или mktime() (листинг 3.71).

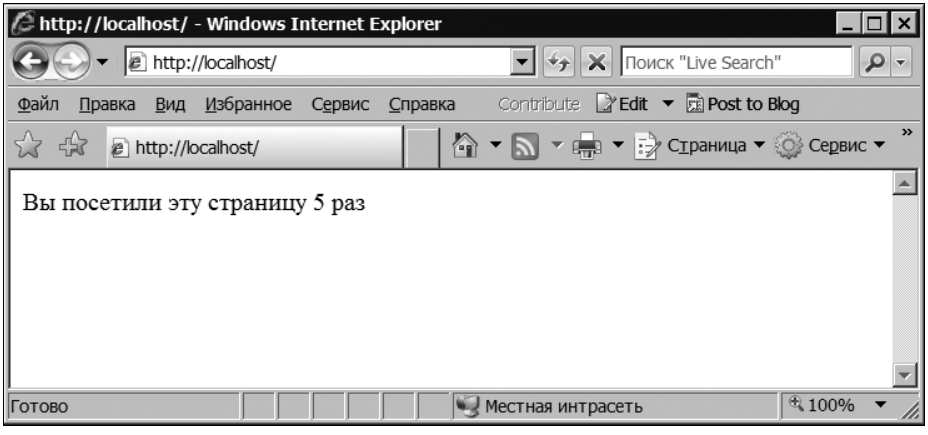


Рис. 3.8. Подсчет количества посещений при помощи cookies

Листинг 3.71. Установка cookie с определенным временем жизни

```
<?php
  // Cookie действителен в течение 10 мин после создания
  setcookie("name", "value", time() + 600);
  // Действие этого cookie прекращается в полночь 25 января 2010 года
  setcookie("name", "value", mktime(0,0,0,1,25,2010));
  // Действие этого cookie прекращается в 18:00 25 января 2010 года
  setcookie("name", "value", mktime(18,0,0,1,25,2010));
?>
```

ЗАМЕЧАНИЕ

В большинстве современных систем, где время представляется 32-битным целым числом, допустимыми являются значения года между 1901 и 2038. Типичной ошибкой при работе с cookie является установка года, выходящего за пределы этого интервала, например, 2100. В этом случае cookie будут устанавливаться только как сессионные.

Ограничить доступ к cookie со всех страниц, кроме расположенных в определенном каталоге (например, /web), можно при помощи четвертого параметра (листинг 3.72).

Листинг 3.72. Ограничение доступа к cookie

```
<?php
  setcookie("name", "value", time() + 600, "/web");
?>
```

Уничтожить cookie можно, установив нулевое время жизни, как это сделано в листинге 3.73.

Листинг 3.73. Удаление cookie

```
<?php
  setcookie("name", "value");
?>
```

> ***Замечание***
>
> Следует помнить, что для уничтожения cookie с ограничениями по каталогу и самому каталогу необходимо выставлять точно такие же ограничения для его удаления, в противном случае он не будет уничтожен.

Cookie можно установить в обход функции `setcookie()`, используя HTTP-заголовок `Set-Cookie` и функцию `header()` (листинг 3.74).

Листинг 3.74. Ручная отправка cookie клиенту

```
<?php
  $cookie = "Set-Cookie: name=value; ".
            "expires=Thu, 30-Aug-2008 13:00:04 GMT; ".
            "path=/; ".
            "domain=www.softtime.ru";
  header($cookie);
?>
```

3.24. Включен ли механизм Cookie в браузере?

Нередко посетители отключают cookies в настройках своих браузеров. Для корректной работы в Web-приложение, использующее cookie, необходимо помещать код, проверяющий, включены ли cookies у посетителя. Такая проверка позволит использовать другой механизм аутентификации, например, сессии, или просто разрешит вывести сообщение о необходимости включить cookies. Пример такой проверки приведен в листинге 3.75.

Листинг 3.75. Проверка, включены ли cookies

```
<?php
  // Выставляем уровень обработки ошибок
  // (http://www.softtime.ru/info/articlephp.php?id_article=23)
  error_reporting(E_ALL & ~E_NOTICE);

  if(!isset($_GET['probe']))
  {
    // Устанавливаем cookie с именем "test"
    if(setcookie("test","set"));
    // Отправляем заголовок переадресации на страницу,
    // с которой будет предпринята попытка установить cookie
    header("Location: $_SERVER[PHP_SELF]?probe=set");
  }
```

```
  else
  {
    if(!isset($_COOKIE['test']))
    {
      echo "Для корректной работы приложения необходимо
            включить cookies";
    }
    else
    {
      echo "Cookies включены";
    }
  }
?>
```

ЗАМЕЧАНИЕ

Суперглобальный массив `$_SERVER` содержит переменные окружения, которые Web-сервер передает скрипту. Среди переменных окружения наибольшей популярностью пользуется `$_SERVER['PHP_SELF']`, сообщающая имя текущего файла. Текущие значения переменных окружения можно посмотреть в отчете, формируемом функцией `phpinfo()`.

В листинге 3.75 для проверки корректности работы с cookies при помощи функции `setcookie()` устанавливается пробное значение cookie. Функция `setcookie()` в данном случае принимает два параметра, первый из которых представляет собой имя, а второй — значение cookie. Далее осуществляется перегрузка текущей страницы с передачей через GET-параметр `probe` значения `set`, сообщающая скрипту, что проверка выполнена, и необходимо выяснить, содержит ли что-нибудь элемент `$_COOKIE['test']`. Если данный элемент не установлен, cookies отключены, если он содержит значение — включены.

3.25. Сессии. Суперглобальный массив *$_SESSION*

Сессия во многом походит на cookie и представляет собой текстовый файл, хранящий пары "ключ/значение", но уже не на машине клиента, а на сервере. Во многих случаях сессии являются более предпочтительным вариантом, чем cookies. Для каждого из посетителей сохраняется собственный набор данных в отдельном файле. Изменение данных одного посетителя никак не отражается на данных другого посетителя.

Поскольку на сервере скапливается большое количество файлов, принадлежащих сессиям разных клиентов, то для их идентификации каждому новому клиенту назначается уникальный номер — идентификатор сессии (SID), который передается либо через строку запроса, либо через cookies, если они доступны. К недостаткам сессий относится невозможность контроля времени их жизни из PHP-скриптов, т. к. этот параметр задается в конфигурационном файле php.ini директивой `session.cookie_lifetime` и может быть запрещен к изменению из скрипта.

Оба механизма, сессии и cookies, взаимно дополняют друг друга. Cookies хранятся на компьютере посетителя, и продолжительность их жизни определяет разработчик. Обычно они применяются для долгосрочных задач (от нескольких часов) и хранения информации, которая относится исключительно к конкретному посетителю (личные настройки, логины, пароли и т. п.). В свою очередь, сессии хранятся на сервере, и продолжительность их существования (обычно небольшую) определяет администратор сервера. Они предназначены для краткосрочных задач (до нескольких часов) и хранения и обработки информации обо всех посетителях в целом (количество посетителей on-line и т. п.). Поэтому использовать тот или иной механизм следует в зависимости от преследуемых целей.

Директива `session.save_path` в конфигурационном файле php.ini позволяет задать путь к каталогу, в котором сохраняются файлы сессий; это может быть удобным для отладки Web-приложений на локальном сервере. Если данная директива не задана, сессии хранятся в оперативной памяти сервера.

Перед тем как начать работать с сессией, клиенту должен быть установлен cookie с уникальным идентификатором SID, а на сервере должен быть создан файл с данными сессии. Все эти начальные действия выполняет функция `session_start()`. Эта функция должна вызываться на каждой странице, где происходит обращение к переменным сессии.

Точно так же, как и функция `setcookie()`, функция `session_start()` должна вызываться до начала вывода информации в окно браузера, т. к. уникальный идентификатор сессии SID передается через cookie, т. е. посредством HTTP-заголовка `Set-Cookie`.

> ***Замечание***
>
> Установив значение директивы `session.auto_start` конфигурационного файла php.ini в 1, можно добиться автоматического старта сессии, без вызова функции `session_start()`.

После инициализации сессии появляется возможность сохранять информацию в суперглобальном массиве `$_SESSION`. В листинге 3.76 на странице index.php в массив `$_SESSION` сохраняется переменная и массив.

Листинг 3.76. Помещаем данные в суперглобальный массив `$_SESSION`

```
<?php
  // Инициируем сессию
  session_start();

  // Помещаем значение в сессию
  $_SESSION['name'] = "value";

  // Помещаем массив в сессию
  $arr = array("first", "second", "third");
  $_SESSION['arr'] = $arr;
```

```
  // Выводим ссылку на другую страницу
  echo "<a href='other.php'>другая страница</a>";
?>
```

На страницах, где происходит вызов функции `session_start()`, значения данных переменных можно извлечь из суперглобального массива `$_SESSION`. В листинге 3.77 приводится содержимое страницы other.php, где извлекаются данные, ранее помещенные на странице index.php.

> ***Замечание***
>
> Массив `$_SESSION` перед сохранением в файл подвергается сериализации, поэтому хранить сериализованные данные в сессии нельзя, т. к. массивы, два раза подряд подвергающиеся действию функции `serialize()`, восстановлению не подлежат.

Листинг 3.77. Извлечение данных из суперглобального массива `$_SESSION`

```
<?php
  // Инициируем сессию
  session_start();
  // Выводим содержимое суперглобального массива $_SESSION
  echo "<pre>";
  print_r($_SESSION);
  echo "</pre>";
?>
```

Результат работы скрипта выглядит следующим образом:

```
Array
(
    [name] => value
    [arr] => Array
        (
            [0] => first
            [1] => second
            [2] => third
        )
)
```

Завершить работу сессии можно, вызвав функцию `session_destroy()`, которая имеет следующий синтаксис:

```
session_destroy()
```

Функция возвращает `TRUE` при успешном уничтожении сессии и `FALSE` в противном случае. Если достаточно не уничтожать текущую сессию, а лишь обнулить все значения, хранящиеся в сессии, следует вызвать функцию `unset()`, которая уничтожит все элементы в суперглобальном массиве `$_SESSION` текущей сессии. Данная функ-

ция точно так же, как и функция session_destroy(), не принимает ни одного параметра и возвращает TRUE в случае успеха и FALSE в случае неудачи.

Точно так же уничтожается отдельный элемент суперглобального массива $_SESSION:

```
unset($_SESSION['name'])
```

Еще одной полезной функцией является функция session_id(), позволяющая узнать текущий идентификатор сессии или задать собственный идентификатор и имеющая следующий синтаксис:

```
session_id([$id])
```

Функция возвращает текущий идентификатор сессии. Необязательный параметр $id позволяет задать собственный идентификатор сессии, который должен состоять из строчных или прописных букв латинского алфавита и цифр. При задании собственного идентификатора функция session_id() должна вызываться до функции session_start() (листинг 3.78).

Листинг 3.78. Узнаем текущий идентификатор сессии

```
<?php
  // Инициируем сессию
  session_start();

  // Узнаем текущий идентификатор сессии
  echo session_id();
?>
```

Возможным результатом работы скрипта в листинге 3.78 может быть строка:

```
a27e8c4724f07cae913c50e55cea1b38
```

3.26. Суперглобальные массивы. Массив *$_SERVER*

PHP содержит ряд предопределенных массивов, которые доступны из любой части программы. Такие массивы заполняются автоматически интерпретатором PHP либо данными, полученными от клиента, либо переменными окружения. По мере изложения материала будут рассмотрены все переменные окружения, тут остановимся на предопределенном массиве $_SERVER — в него PHP-интерпретатор помещает переменные окружения Web-сервера Apache. Без данных переменных сложно организовать полноценную поддержку Web-приложений. Приведем описание наиболее важных элементов суперглобального массива $_SERVER.

Замечание

Просмотреть полный список элементов массива $_SERVER можно либо при помощи функции print_r(), которая распечатывает дамп массива, либо при помощи функции phpinfo(), которая выводит информацию о PHP-интерпретаторе.

3.26.1. Элемент *$_SERVER['DOCUMENT_ROOT']*

Элемент `$_SERVER['DOCUMENT_ROOT']` содержит путь корневого каталога сервера. Если скрипт выполняется в виртуальном хосте, в данном элементе, как правило, указывается путь к корневому каталогу виртуального хоста. В конфигурационном файле `httpd.conf` виртуальный хост имеет директиву `DocumentRoot`, которой присвоено значение "`D:/main`" (листинг 3.79), элемент `$_SERVER['DOCUMENT_ROOT']` будет содержать значение "`D:\main`".

Листинг 3.79. Виртуальный хост

```
NameVirtualHost 127.0.0.1:80
<VirtualHost 127.0.0.1:80>
    ServerAdmin admin@localhost
    DocumentRoot "D:/main"
    ServerName localhost
    ErrorLog logs/ error_log
    CustomLog logs/ access_log common
</VirtualHost>
```

3.26.2. Элемент *$_SERVER['HTTP_ACCEPT']*

В элементе `$_SERVER['HTTP_ACCEPT']` описываются предпочтения клиента относительно типа документа. Содержимое этого элемента извлекается из HTTP-заголовка `Accept`, который присылает клиент серверу. Содержимое данного заголовка может выглядеть следующим образом:

```
image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/
x-shockwave-flash, application/vnd.ms-excel, application/msword, */*
```

Заголовок `Accept` позволяет уточнить медиатип, который предпочитает получить клиент в ответ на свой запрос. Этот заголовок позволяет сообщить серверу, что ответ ограничен небольшим множеством предпочитаемых типов.

Символ `*` используется для группирования типов в медиаряду. К примеру, символом `*/*` задается использование всех типов, а обозначение `type/*` определяет использование всех подтипов выбранного типа `type`.

> ***ЗАМЕЧАНИЕ***
>
> Медиатипы отделяются друг от друга запятыми.

Каждый медиаряд характеризуется также дополнительным набором параметров. Одним из них является так называемый относительный коэффициент предпочтения `q`, который принимает значения от 0 до 1 соответственно, от менее предпочитаемых типов к более предпочитаемым. Использование нескольких параметров `q` позволяет клиенту сообщить серверу относительную степень предпочтения для того или иного медиатипа.

Замечание

По умолчанию параметр q принимает значение 1. Кроме того, от медиатипа он отделяется точкой с запятой.

Пример заголовка типа `Accept`:

```
Accept: audio/*; q=0.2, audio/basic
```

В данном заголовке первым идет тип `audio/*` (музыкальные документы), характеризующийся коэффициентом предпочтения 0,2. Тип `audio/basic`, для которого коэффициент предпочтения не указан, принимает значение по умолчанию 1. Данный заголовок можно интерпретировать следующим образом: предпочитаю тип audio/basic, но мне можно также слать документы любого другого audio-типа, если они будут доступны, после снижения коэффициента предпочтения более чем на 80 %.

Замечание

Загрузить RFC2616, где описывается протокол HTTP, можно по адресу **http://www.ysn.ru/docs/cie/RFC/2616/index.htm**.

Пример может быть более сложным:

```
Accept: text/plain; q=0.5, text/html,
        text/x-dvi; q=0.8, text/x-c
```

Замечание

Следует учитывать, что элемент `$_SERVER['HTTP_ACCEPT']` содержит точно такую же информацию, но без начального заголовка `Accept:`.

Этот заголовок интерпретируется следующим образом: типы документов `text/html` и `text/x-c` являются предпочтительными, но если они недоступны, тогда клиент, отсылающий данный запрос, предпочтет `text/x-dvi`, а если и его нет, то он может принять тип `text/plain`.

3.26.3. Элемент *$_SERVER['HTTP_ACCEPT_LANGUAGE']*

В элементе `$_SERVER['HTTP_ACCEPT_LANGUAGE']` описываются предпочтения клиента относительно языка. Данная информация извлекается из HTTP-заголовка `Accept-Language`, который присылает клиент серверу. Можно привести следующий пример:

```
Accept-Language: ru, en; q=0.7
```

Интерпретировать его можно следующим образом: клиент предпочитает русский язык, но в случае его отсутствия согласен принимать документы на английском. Элемент `$_SERVER['HTTP_ACCEPT_LANGUAGE']` будет содержать точно такую же информацию, но без заголовка `Accept-Language`:

```
ru, en; q=0.7
```

Содержимое элемента `$_SERVER['HTTP_ACCEPT_LANGUAGE']` можно использовать лишь приблизительно, т. к. многие пользователи используют английские варианты браузеров (посетитель предпочитает лишь один язык — английский).

3.26.4. Элемент *$_SERVER['HTTP_HOST']*

В элементе `$_SERVER['HTTP_HOST']` содержится имя сервера, которое, как правило, совпадает с доменным именем сайта, расположенного на сервере — `$_SERVER['SERVER_NAME']`. В параметре приводится лишь доменное имя без названия протокола (http://), например:

```
www.sofftime.ru
```

> ***Замечание***
>
> Ряд хост-провайдеров помещают в данную переменную окружения домен третьего уровня www.sofftime.ru, ряд — домен второго уровня sofftime.ru.

3.26.5. Элемент *$_SERVER['HTTP_REFERER']*

В элемент `$_SERVER['HTTP_REFERER']` помещается адрес страницы, откуда посетитель пришел на данную страницу. Переход должен осуществляться по ссылке. Создадим две страницы `index.php` (листинг 3.80) и `page.php` (листинг 3.81).

Листинг 3.80. Страница `index.php`

```
<?php
  echo "<a href=page.php>Ссылка на страницу PHP</a><br>";
  echo "Содержимое \$_SERVER['HTTP_REFERER'] — ".
       $_SERVER['HTTP_REFERER']
?>
```

Страница `page.php` будет аналогичного содержания, но ссылка будет указывать на страницу `index.php` (листинг 3.81).

Листинг 3.81. Страница `page.php`

```
<?php
  echo "<a href=index.php>Ссылка на страницу PHP</a><br>";
  echo "Содержимое \$_SERVER['HTTP_REFERER'] — ".
       $_SERVER['HTTP_REFERER']
?>
```

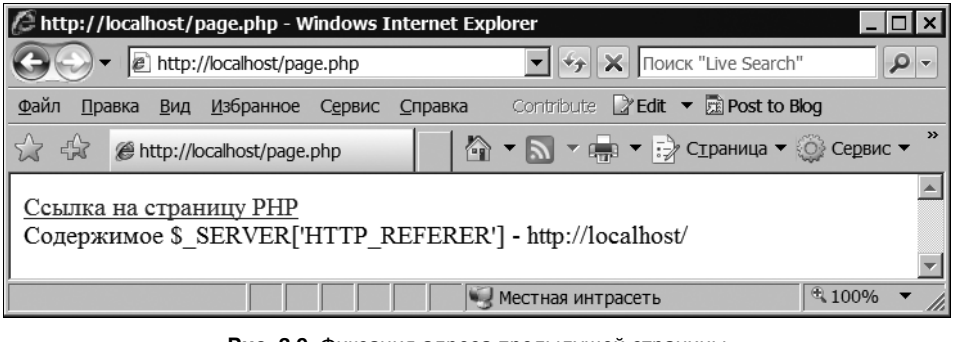


Рис. 3.9. Фиксация адреса предыдущей страницы

При переходе с одной страницы на другую, под ссылкой будет выводиться адрес страницы, с которой был осуществлен переход (рис. 3.9).

3.26.6. Элемент $_SERVER['HTTP_USER_AGENT']

Элемент `$_SERVER['HTTP_USER_AGENT']` содержит информацию о типе и версии браузера и операционной системы посетителя.

Вот типичное содержание этой строки: "`Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)`". Наличие подстроки "`MSIE 6.0`" говорит о том, что посетитель просматривает страницу при помощи Internet Explorer версии 6.0. Строка "`Windows NT 5.1`" сообщает, что в качестве операционной системы используется Windows XP.

> ***Замечание***
>
> Для Windows 2000 элемент `$_SERVER['HTTP_USER_AGENT']` выглядит следующим образом: "`Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)')`", в то время как для Windows XP — "`Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)`".

Если посетитель воспользуется браузером Opera, то содержание `$_SERVER['HTTP_USER_AGENT']` может выглядеть следующим образом: "`Mozilla/4.0 (compatible; MSIE 5.0; Windows 98) Opera 6.04 [ru]`". Подстрока "`MSIE 6.0`" здесь так же присутствует, сообщая, что браузер Opera является совместимым с браузером Internet Explorer и использует те же динамические библиотеки Windows. Поэтому при анализе строки, возвращаемой браузером, следует иметь в виду, что к Internet Explorer относится строка, содержащая подстроку "`MSIE 6.0`" и не содержащая подстроки "`Opera`". Кроме того, из данной строки можно заключить, что пользователь использует операционную систему Windows 98.

При использовании браузера Netscape содержание элемента `$_SERVER['HTTP_USER_AGENT']` может выглядеть следующим образом: "`Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.4) Gecko/20030624 Netscape/7.1`". Принадлежность к этому браузеру можно определить по наличию подстроки "`Netscape`". Кроме того, можно узнать, что посетитель выходит в Интернет, используя операционную версию Linux, с ядром, оптимизированным под Pentium IV, находясь в графической оболочке X-Window. Этот механизм удобно использовать для сбора статистической информации, которая позволяет дизайнерам оптимизировать страницы под наиболее распространенные браузеры.

3.26.7. Элемент $_SERVER['REMOTE_ADDR']

В элемент `$_SERVER['REMOTE_ADDR']` помещается IP-адрес клиента. При тестировании на локальной машине этот адрес будет равен 127.0.0.1. Однако при тестировании в сети переменная вернет IP-адрес клиента или последнего прокси-сервера, через который клиент попал на сервер. Если клиент использует прокси-сервер, узнать его IP-адрес можно при помощи переменной окружения `$_SERBER['HTTP_X_FORWARDED_FOR']`.

Замечание

Прокси-серверы являются специальными промежуточными серверами, предоставляющими специальный вид услуг: сжатие трафика, кодирование данных, адаптация под мобильные устройства и т. п. Среди множества прокси-серверов различают так называемые анонимные прокси-серверы, которые позволяют скрывать истинный IP-адрес клиента, такие серверы не возвращают переменной окружения `HTTP_X_FORWARDED_FOR`.

3.26.8. Элемент *$_SERVER['SCRIPT_FILENAME']*

В элемент `$_SERVER['SCRIPT_FILENAME']` помещается абсолютный путь к файлу от корня диска. Так если сервер работает под управлением операционной системы Windows, то такой путь может выглядеть следующим образом `d:\main\test\index.php`, т. е. путь указывается от диска, в UNIX-подобной операционной системы путь указывается от корня /, например: `/var/share/www/test/index.php`.

3.26.9. Элемент *$_SERVER['SERVER_NAME']*

В элемент `$_SERVER['SERVER_NAME']` помещается имя сервера, как правило, совпадающее с доменным именем сайта, расположенного на нем. Например:

```
www.softtime.ru
```

Содержимое элемента `$_SERVER['SERVER_NAME']` часто совпадает с содержимым элемента `$_SERVER['HTTP_HOST']`. Помимо имени сервера суперглобальный массив `$_SERVER` позволяет выяснить еще ряд параметров сервера, например IP-адрес сервера, прослушиваемый порт, какой Web-сервер установлен и версию HTTP-протокола. Эта информация помещается в элементы `$_SERVER['SERVER_ADDR']`, `$_SERVER['SERVER_PORT']`, `$_SERVER['SERVER_SOFTWARE']` и `$_SERVER['SERVER_PROTOCOL']` соответственно. В листинге 3.82 приводится пример с использованием данных элементов.

Результат работы скрипта из листинга 3.82 представлен на рис. 3.10.

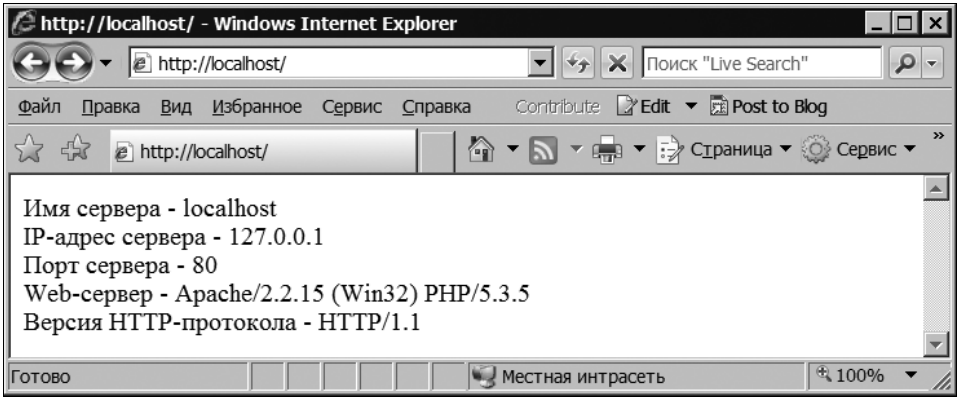


Рис. 3.10. Результат работы скрипта из листинга 3.82

Листинг 3.82. Использование элементов массива $_SERVER

```
<?php
  echo "Имя сервера — ".$_SERVER['SERVER_NAME']."<br>";
  echo "IP-адрес сервера — ".$_SERVER['SERVER_ADDR']."<br>";
  echo "Порт сервера — ".$_SERVER['SERVER_PORT']."<br>";
  echo "Web-сервер — ".$_SERVER['SERVER_SOFTWARE']."<br>";
  echo "Версия HTTP-протокола — ".$_SERVER['SERVER_PROTOCOL']."<br>";
?>
```

3.26.10. Элемент *$_SERVER['REQUEST_METHOD']*

В элемент `$_SERVER['REQUEST_METHOD']` помещается метод запроса, который применяется для вызова скрипта: GET или POST (листинг 3.83).

Листинг 3.83. Определение метода запроса

```
<?php
  echo $_SERVER['REQUEST_METHOD']; // GET
?>
```

3.26.11. Элемент *$_SERVER['QUERY_STRING']*

В элемент `$_SERVER['QUERY_STRING']` заносятся параметры, переданные скрипту, если строка запроса представляет собой адрес:

```
http://www.mysite.ru/test/index.php?id=1&test=wet&id_theme=512
```

то в элемент `$_SERVER['QUERY_STRING']` попадет весь текст после знака `?`. Например, при обращении к скрипту, представленному в листинге 3.84, помещая в строке запроса произвольный текст после знака `?`, получим страницу с введенным текстом.

Результат работы скрипта из листинга 3.84 представлен на рис. 3.11.

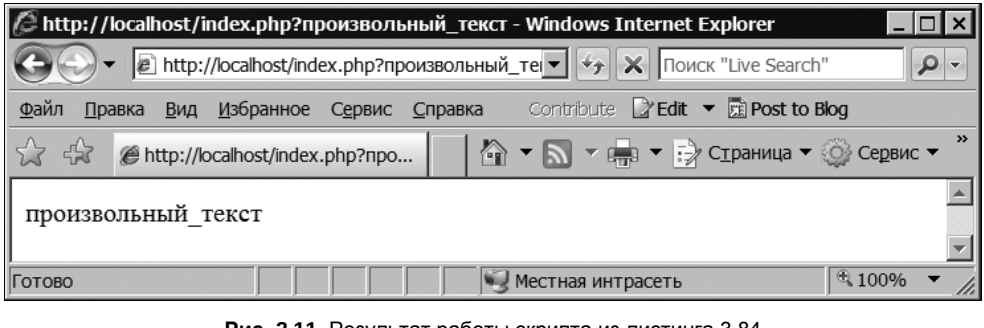


Рис. 3.11. Результат работы скрипта из листинга 3.84

Листинг 3.84. Использование элемента $_SERVER['QUERY_STRING']

```
<?php
  echo $_SERVER['QUERY_STRING'];
?>
```

3.26.12. Элемент $_SERVER['PHP_SELF']

В элемент `$_SERVER['PHP_SELF']` помещается имя скрипта, начиная от корня виртуального хоста:

```
http://www.mysite.ru/test/index.php?id=1&test=wet&id_theme=512
```

то элемент `$_SERVER['PHP_SELF']` будет содержать фрагмент `/test/index.php`. Как правило, этот же фрагмент помещается в элемент `$_SERVER['SCRIPT_NAME']`.

3.26.13. Элемент $_SERVER['REQUEST_URI']

Элемент `$_SERVER['REQUEST_URI']` содержит имя скрипта, начиная от корня виртуального хоста, и параметры:

```
http://www.mysite.ru/test/index.php?id=1&test=wet&id_theme=512
```

Элемент `$_SERVER['REQUEST_URI']` будет содержать фрагмент `/test/index.php?id=1&test=wet&id_theme=512`. Для того чтобы восстановить в скрипте полный адрес, который помещен в строке запроса, достаточно использовать комбинацию элементов массива `$_SERVER`, представленную в листинге 3.85.

Листинг 3.85. Полный адрес к скрипту

```
<?php
  echo "http://".$_SERVER['SERVER_NAME'].$_SERVER['REQUEST_URI'];
?>
```

ГЛАВА 4

Файлы и каталоги

В данной главе рассматриваются приемы работы с файлами и каталогами. *Файл* представляет собой последовательность байтов на каком-либо физическом носителе информации. Каждый файл имеет абсолютный путь, по которому определяется его местонахождение.

Каталогом называют файл специального вида, который хранит список других файлов и каталогов, входящих в его состав. Как правило, для обработки каталогов предназначен специальный набор функций.

4.1. Создание файлов

Первыми рассмотрим функции, позволяющие создать файлы (табл. 4.1).

Таблица 4.1. *Функции создания файлов*

Функция	Описание
`fopen($filename, $mode [, $use_include_path [, $context]])`	Открывает файл с именем `$filename` в режиме `$mode`. Ряд режимов позволяет создать файл, ряд требует, чтобы файл уже существовал. Необязательный параметр `$use_include_path` позволяет задать путь для поиска файла. В случае успеха функция возвращает дескриптор открытого файла, в случае неудачи — `FALSE`. Параметр `$context` позволяет задать HTTP-заголовки, отправляемые сервером при обращении к сетевому файлу
`fclose($handle)`	Закрывает файл, открытый ранее при помощи функции `fopen()`. В качестве единственного параметра `$handler` функция принимает дескриптор открытого файла. Возвращает `TRUE` при успешном закрытии файла и `FALSE` в противном случае
`touch($filename [, $time [, $atime]])`	Функция `touch()` предназначена для изменения времени последней модификации файла `$time` и времени последнего доступа `$atime`. Если файл не существует, он создается, поэтому функция часто используется для создания файлов. Подробнее функция обсуждается в *разд. 13.5*
`tempnam($dir, $prefix)`	Создает в каталоге `$dir` файл с уникальным именем, при этом в имени файла используется префикс `$prefix`. В случае успеха возвращает имя нового файла, в случае неудачи — `FALSE`

Таблица 4.1 *(окончание)*

Функция	Описание
`tmpfile()`	Создает временный файл с уникальным именем и возвращает его дескриптор. Файл автоматически удаляется после его закрытия. Если создать файл не удалось, функция возвращает `FALSE`

Один скрипт может оперировать множеством файлов. Чтобы их как-то отличать друг от друга, используется либо имя файла, либо его дескриптор. Имя файла может содержать полный путь, начиная от корня диска, такой путь называется *абсолютным*:

```
/var/home/www/htdocs/text.txt
C:\www\htdocs\text.txt
```

> ***Замечание***
>
> В операционных системах Windows разделы обозначаются символами английского алфавита, буквы A и B традиционно резервируются под гибкие диски, остальные буквы используются для именования разделов жесткого диска. В UNIX-подобных операционных системах любой путь начинается с корневого раздела (root-раздела) `/`, к каталогам которого монтируются физические разделы. Таким образом, если в Windows может быть несколько корневых точек, обозначенных буквами, в UNIX-подобных операционных системах такая точка всегда одна.
>
> Функции `fopen()` и `tmpfile()` предназначены не столько для создания файла, сколько для его открытия, т. е. получения дескриптора, который позволяет осуществлять манипуляции с содержимым файла.

В различных операционных системах используются разные разделители между каталогами и файлами. В UNIX-подобных операционных системах в качестве разделителя используется прямой слэш `/`, в Windows — обратный `\`. Обратный слэш традиционно служит для экранирования в строках, поэтому его использование доставляет массу неудобств Windows-разработчикам. К сожалению, исправить ситуацию не представляется возможным — такая нотация используется свыше 20 лет во всех операционных системах компании Microsoft, начиная с DOS. В языке PHP эта проблема частично разрешена — при формировании путей к файлам допускается применять как прямой `/`, так и обратный слэши `\`. Однако при использовании обратного слэша следует помнить, что его необходимо удваивать в строках. В листинге 4.1 приводится пример формирования абсолютного Windows-пути, обе строки полностью эквивалентны.

Листинг 4.1. Использование прямого и обратного слэша при формировании пути

```
<?php
  $windows = "C:\\www\\htdocs\\text.txt";
  $php     = "C:/www/htdocs/text.txt";
?>
```

Как видно из листинга 4.1, удобнее оперировать прямыми слэшами `/`, именно они и будут использоваться при дальнейшем изложении.

Помимо абсолютного пути различают *относительный* путь, который выстраивается относительно текущего каталога (как правило, каталога, в котором расположен скрипт).

На рис. 4.1 представлена гипотетическая структура файловой системы, при этом текущий каталог C:/www/ выделен серым цветом. Для доступа к нижележащим каталогам достаточно указать путь относительно текущего каталога. В листинге 4.2 приводятся абсолютный и относительный пути к файлу C:/www/base/1.txt, который можно использовать в скрипте C:/www/scr.php.

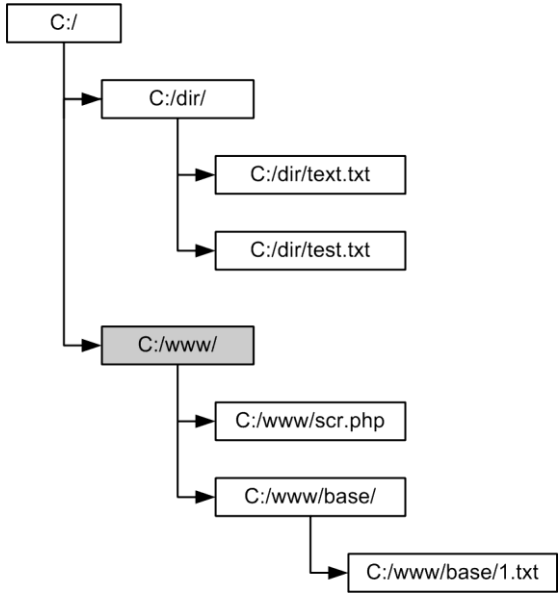


Рис. 4.1. Гипотетическая структура файловой системы

Листинг 4.2. Абсолютный и относительный путь к файлу 1.txt

```
<?php
  // Абсолютный путь к файлу 1.txt
  $absolute = "C:/www/base/1.txt";
  // Относительный путь к файлу 1.txt
  $relative = "base/1.txt";
?>
```

Следует отметить, что относительный путь не предваряет ни буква раздела, ни символ корневого раздела / — он начинается либо с имени каталога, либо с имени файла.

Как видно из рис. 4.1, для того чтобы использовать относительный путь для доступа к файлу C:/dir/text.txt, необходимо подняться на один уровень выше и спуститься в каталог dir. Для формирования таких путей используется последовательность из двух точек .., обозначающая родительский каталог (листинг 4.3).

Замечание

Помимо родительского каталога, обозначаемого двумя точками (`..`), большинство файловых систем поддерживает текущий каталог, который обозначается одной точкой.

Листинг 4.3. Использование родительского каталога

```
<?php
  // Абсолютный путь к файлу text.txt
  $absolute = "C:/dir/text.txt";
  // Относительный путь к файлу text.txt
  $relative = "../dir/1.txt";
?>
```

Если необходимо подняться еще выше, можно использовать несколько последовательностей `..`, например, `../../../result/text.txt`.

Абсолютный и относительный пути позволяют оперировать именами файлов в пределах файловой системы. В Web также используется *сетевой* путь, который начинается с имени протокола, например, **http://www.softtime.ru/index.php**.

Теперь, когда правила формирования путей к файлам определены, создадим файл в текущем каталоге. Для создания пустого файла удобнее всего воспользоваться функцией `touch()` (листинг 4.4).

Листинг 4.4. Использование функции `touch()`

```
<?php
  // Имя файла
  $filename = "text.txt";
  // Создаем пустой файл с именем $filename
  touch($filename);
?>
```

В случае успешного выполнения скрипта из листинга 4.4 в каталоге, где он расположен, появится новый файл text.txt. Функция `touch()` является типичной функцией, которая ориентируется на название файла. Однако при доступе к файлам более распространен дескрипторный вариант — имя файла связывается с дескриптором (целым числом, уникальным в пределах скрипта), который и используется в дальнейшем для доступа к этому файлу. Процесс связывания имени файла с дескриптором называется "открытием" файла и осуществляется при помощи функции `fopen()`, которая имеет следующий синтаксис:

```
fopen($filename, $mode [, $use_include_path [, $context]])
```

Функция открывает файл с абсолютным, относительным или сетевым путем `$filename` в режиме `$mode`. Если файл отсутствует в текущем каталоге, его поиск может продолжиться в каталогах, заданных в параметре `$use_include_path` (если этот параметр используется). При успешном выполнении функция возвращает дескриптор открытого файла, в противном случае возвращается `FALSE`.

В табл. 4.2 приводится описание режимов `$mode`, в которых может открываться файл.

Таблица 4.2. *Режимы открытия файла функцией* `fopen()`

Режим	Чтение	Запись	Файловый указатель	Очистка файла	Создать, если файла нет	Ошибка, если файл есть
`r`	Да	Нет	В начале	Нет	Нет	Нет
`r+`	Да	Да	В начале	Нет	Нет	Нет
`w`	Нет	Да	В начале	Да	Да	Нет
`w+`	Да	Да	В начале	Да	Да	Нет
`a`	Нет	Да	В конце	Нет	Да	Нет
`a+`	Да	Да	В конце	Нет	Да	Нет
`x`	Нет	Да	В начале	Нет	Да	Да
`x+`	Да	Да	В начале	Нет	Да	Да

Как видно из таблицы, для создания файлов подходят любые режимы, кроме `r` и `r+`. При попытке открыть несуществующий файл в одном из этих режимов функция `fopen()` возвращает `FALSE`, в случае остальных режимов функция пытается создать несуществующий файл. В листинге 4.5 приводится пример создания файла с именем text.txt при помощи функции `fopen()`.

Листинг 4.5. Создание файла при помощи функции `fopen()`

```
<?php
  // Имя файла
  $filename = "text.txt";
  // Получаем дескриптор открытого файла
  $fd = fopen($filename, "w");
  // Если файл успешно открыт, сообщаем об этом и закрываем его
  if (!$fd)
  {
    echo "Файл успешно создан<br>";
    fclose($fd);
  }
?>
```

Закрывать файл при помощи функции `fclose()` необязательно, т. к. все открытые дескрипторы закрываются автоматически после завершения работы скрипта. Однако лучше все-таки использовать функцию `fclose()`, особенно при записи информации в файл: информация записывается в файл не побайтово, а блоками по мере заполнения буфера записи. Содержимое буфера сбрасывается на диск либо при его заполнении, либо при использовании функции `fflush()`, либо при закрытии файла

функцией `fclose()`. Если работа скрипта будет неожиданно остановлена, информация из буфера, не сброшенная на жесткий диск, пропадет.

4.2. Создание файлов с уникальными именами

Часто файлы используются как временные хранилища информации, в этом случае удобно прибегать к созданию временных файлов. Функция `tempnam()` позволяет создать в каталоге файл с уникальным именем и имеет следующий синтаксис:

```
tempnam($dir, $prefix)
```

Функция создает файл с уникальным именем в каталоге `$dir` и с префиксом `$prefix`. В качестве результата функция `tempnam()` возвращает имя только что созданного файла. Если создать файл по каким-то причинам не удается, функция возвращает `FALSE`.

В листинге 4.6 скрипт создает в текущем каталоге новый файл с уникальным именем при каждом нажатии на кнопку.

Листинг 4.6. Создание файлов с уникальными именами

```
<form method='post'>
  <input type='submit' value='Записать'>
</form>
<?php
  // Обработчик HTML-формы
  if (isset($_POST))
  {
    // Создаем файл со случайным именем
    $filename = tempnam("./", "fl");
    // Открываем файл
    $fd = fopen($filename, "w");
    // Закрываем файл
    fclose($fd);
  }
?>
```

Помимо функции `tempnam()` PHP предоставляет в распоряжение разработчика функцию `tmpfile()`, которая также создает временный файл с уникальным именем, но возвращает дескриптор открытого файла. После того как файл закрывается, файл автоматически удаляется. В отличие от функции `tmpfile()`, файл, созданный функцией `tempnam()`, остается существовать и после вызова функции.

4.3. Копирование, переименование и удаление файлов

Помимо операции создания файлов очень часто возникают задачи, связанные с их перемещением, переименованием и удалением. Функции, ответственные за эти операции, представлены в табл. 4.3.

Таблица 4.3. *Функции манипуляции файлами*

Функция	Описание
`copy($source, $destination)`	Копирует файл с именем `$source` в файл с именем `$destination`. В случае успешного копирования функция возвращает `TRUE`, в противном случае возвращает `FALSE`
`unlink($filename)`	Удаляет файл с именем `$filename`. В случае успешного удаления функция возвращает `TRUE`, в противном случае возвращает `FALSE`
`rename($oldname, $newname)`	Переименовывает файл с именем `$oldname`, назначая ему новое имя `$newname`. В случае успешного переименования функция возвращает `TRUE`, в противном случае возвращает `FALSE`

В листинге 4.7 приводится пример использования функции `copy()`, которая копирует файл /etc/my.cnf в текущий каталог. Если на момент копирования файл my.cnf существовал в текущем каталоге, он будет перезаписан без предупреждений.

Листинг 4.7. Использование функции `copy()`

```
<?php
  if (copy("/etc/my.cnf", "my.cnf")) echo "Файл успешно скопирован";
  else echo "Не удалось скопировать файл";
?>
```

Если файл необходимо переместить, то его можно удалить из точки копирования при помощи функции `unlink()` (листинг 4.8).

Листинг 4.8. Перемещение файла

```
<?php
  // Копируем файл
  if (!copy("/etc/my.cnf", "my.cnf")) exit "Не удалось переместить файл";
  // Удаляем исходный файл
  if (unlink("/etc/my.cnf")) echo "Файл успешно перемещен";
  else echo "Не удалось переместить файл";
?>
```

Впрочем, операцию по переносу файла можно осуществить в один прием при помощи функции `rename()`, которая предназначена для переименования файлов, одна-

ко вполне может осуществлять их перенос — для этого достаточно указать новый путь, оставив имя файла без изменения (листинг 4.9).

Листинг 4.9. Использование функции `rename()`

```
<?php
  // Перенос файла
  if (rename("/etc/my.cnf", "my.cnf")) echo "Файл успешно перемещен";
  else echo "Не удалось переместить файл";
?>
```

Функции `copy()` и `rename()` могут копировать не только локальные файлы, но и сетевые. В листинге 4.10 демонстрируется копирование файла **http://www.softtime.ru/index.php**.

Листинг 4.10. Копирование файла из сети

```
<?php
  if (copy("http://www.softtime.ru/index.php", "index.htm"))
    echo "Файл успешно скопирован";
  else echo "Не удалось скопировать файл";
?>
```

Загруженный из сети файл не будет содержать исходных PHP-кодов, лишь HTML-представление (даже если скрипт и копируемый файл находятся на одном сервере). Это связано с тем, что обращение идет к Web-серверу, а не к файловой системе. По этой же причине при помощи функции `copy()` или любой другой функции невозможно загрузить файл на сервер.

4.4. Чтение содержимого файлов

При открытии файла при помощи функции `fopen()` он может существовать или отсутствовать; существующий файл может содержать информацию или нет. Режимы открытия файла при помощи функции `fopen()` представлены в табл. 4.2.

Файл можно представить как последовательность байтов, по которой перемещается файловый указатель (рис. 4.2). Чтение и запись производится с текущего положения файлового указателя от начала к концу. Чтение и запись файлов в обратном порядке не допускается.

При помощи режимов `r`, `w` и `x` файл открывается с установленным в начало файловым указателем. Это позволяет читать содержимое файла с начала, однако запись в файл также производится с начальной позиции, и все старое содержимое файла перезаписывается. Для того чтобы записать файл, не стирая его содержимое, файл следует открыть в режиме `a`, при котором файловый указатель помещается в конец файла. Несмотря на то, что режим `a+` допускает чтение, сразу после открытия файла прочитать ничего не удастся, т. к. файловый указатель находится в конце.

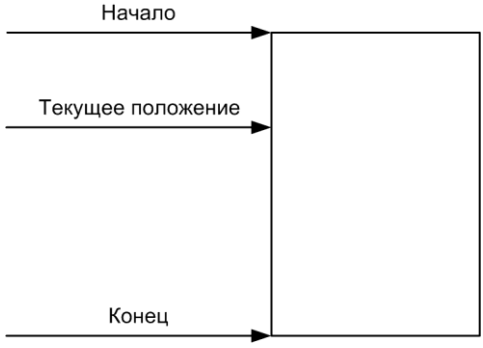


Рис. 4.2. Последовательное перемещение файлового указателя по файлу

Функции чтения, читая порцию информации из файла, перемещают файловый указатель в конец прочитанного блока, поэтому, последовательно применяя их к одному и тому же файлу, можно просмотреть его содержимое от начала до конца.

В листинге 4.11 демонстрируется скрипт, открывающий текстовый файл text.txt, читающий из него символ за символом (при помощи функции `fgetc()`) и выводящий полученные символы в окно браузера.

Листинг 4.11. Чтение файла

```
<?php
  // Открываем файл на чтение
  $fd = fopen("text.txt", "r");
  if (!$fd) exit("Невозможно открыть файл text.txt");
  // Последовательно читаем байты из файла, пока не достигнем конца файла
  while(($char = fgetc($fd)) !== FALSE)
  {
    echo $char;
  }
  // Закрываем файл
  fclose($fd);
?>
```

Оператор равенства = возвращает результат присвоения, поэтому цикл `while()` действует до тех пор, пока функция `fgetc()` не достигнет конца файла и не вернет значение `FALSE`. Сравнение следует осуществлять при помощи оператора неэквивалентности `!==`, т. к. оператор неравенства `!=` остановит цикл, если будет встречен символ, который автоматически приводится к `FALSE` (например, 0).

Помимо способа остановки цикла, продемонстрированного в листинге 4.11, часто используется альтернативный вариант, основанный на явной проверке положения файлового указателя. Для такой проверки предназначена функция `feof()`, которая возвращает `TRUE`, если файловый указатель указывает на конец файла, и `TRUE` в противном случае. В листинге 4.12 приводится содержимое листинга 4.11, переписанное с применением функции `feof()`.

Листинг 4.12. Использование функции `feof()`

```
<?php
  // Открываем файл на чтение
  $fd = fopen("text.txt", "r");
  if (!$fd) exit("Невозможно открыть файл text.txt");
  // Последовательно читаем байты из файла, пока не достигнем конца файла
  while(!feof($fd))
  {
    $char = fgetc($fd);
    echo $char;
  }
  // Закрываем файл
  fclose($fd);
?>
```

При открытии файла при помощи функции `fopen()` следует обратить внимание на режим во втором параметре функции (см. табл. 4.2). Примеры из листингов 4.11 и 4.12 будут работать лишь в том случае, если открытие файла осуществляется в режимах `r`, `r+`, `w+`, `a+` и `x+`, при попытке открыть файл в режимах `w`, `a` и `x` чтение из файла будет невозможным. Впрочем, режим `x+` создает пустой несуществующий файл, и читать из него нечего, режим `w+` стирает предыдущее содержимое, а `a+` устанавливает файловый дескриптор в конец файла, и чтение невозможно. Поэтому при чтении из файла разумно открывать его в режимах `r` и `r+`.

К побайтовому чтению файлов прибегают достаточно редко. Так как при помощи PHP чаще решаются прикладные задачи, разработчики чаще работают с текстовыми файлами, которые удобнее читать построчно. Для этого используют функцию `fgets()`, которая имеет следующий синтаксис:

```
fgets($handle [, $length])
```

Функция считывает из открытого файла `$handler` строку. Чтение заканчивается по достижению строки длиной `$handle` – 1 (по умолчанию `$handle` принимает значение 1000), по достижению символа перевода строки или символа конца файла. В случае успеха функция возвращает прочитанную строку, в случае неудачи — `FALSE`. В листинге 4.13 приводится пример использования функции `fgets()` для чтения текстового файла text.txt.

Листинг 4.13. Использование функции `fgets()`

```
<?php
  // Открываем файл на чтение
  $fd = fopen("text.txt", "r");
  if (!$fd) exit("Невозможно открыть файл text.txt");
  // Построчно читаем данные из файла, пока не достигнем конца файла
  while(!feof($fd))
```

```
  {
    $str = fgets($fd, 1024);
    echo "$str<br>";
  }
  // Закрываем файл
  fclose($fd);
?>
```

Какое бы большое значение не принимал второй параметр функции `fgets()`, чтение из файла будет прекращено при встрече первого перевода строки. Если такое поведение не удовлетворяет требованиям разработчика, можно воспользоваться функцией `fread()`, лишенной такой особенности. Функция имеет следующий синтаксис:

```
fread($handle, $length)
```

Функция `fread()` читает из открытого файла с дескриптором `$handle` блок информации длиной `$length`. В листинге 4.14 приводится пример использования функции `fread()`.

Листинг 4.14. Использование функции `fread()`

```
<?php
  // Открываем файл на чтение
  $fd = fopen("text.txt", "r");
  if (!$fd) exit("Невозможно открыть файл text.txt");
  // Построчно читаем данные из файла, пока не достигнем конца файла
  while(!feof($fd))
  {
    $str = fread($fd, 1024);
    echo "$str<br>";
  }
  // Закрываем файл
  fclose($fd);
?>
```

Если заранее известен размер файла, то при помощи функции `fread()` можно прочитать содержимое файла за один раз (листинг 4.15).

Листинг 4.15. Чтение файла за один раз

```
<?php
  // Открываем файл на чтение
  $fd = fopen("text.txt", "r");
  if (!$fd) exit("Невозможно открыть файл text.txt");
  // Читаем содержимое файла за один раз
  $str = fread($fd, filesize($fd));
  echo $str;
```

```
  // Закрываем файл
  fclose($fd);
?>
```

Для определения размера файла в листинге 4.15 использовалась функция `filesize()`. Вместо точного размера файла в качестве второго параметра функции `fread()` можно подставить размер, заведомо превышающий размер файла. Однако следует проявлять аккуратность при загрузке сетевого файла: второй параметр определяет размер сетевого пакета, оптимальным считается 4 096 байтов (листинг 4.16).

Листинг 4.16. Загрузка файла из сети

```
<?php
  // Открываем файл на чтение
  $fd = fopen("http://www.softtime.ru", "r");
  if (!$fd) exit("Невозможно открыть сетевой файл");
  // Построчно читаем данные из файла, пока не достигнем конца файла
  $str = "";
  while(!feof($fd))
  {
    $str .= fread($fd, 4096);
  }
  // Выводим полученную строку
  echo $str;
  // Закрываем файл
  fclose($fd);
?>
```

Листинги 4.15 и 4.16 демонстрируют типичную задачу — получение содержимого файла в переменную. Эта задача настолько распространена, что для ее решения в PHP введена специализированная функция `file_get_contents()`, которая имеет следующий синтаксис:

```
file_get_contents($filename [, $flags [,
                  $context [, $offset [, $maxlen]]]])
```

Функция читает файл с именем `$filename` и возвращает его содержимое в виде строки. В отличие от большинства функций данного раздела, в качестве первого аргумента выступает путь к файлу, а не дескриптор, полученный при помощи функции `fopen()`.

Второй необязательный параметр `$flags` позволяет задать режимы работы функции. Допустимые значения этого параметра представлены в табл. 4.4. Можно указать несколько параметров, соединив их оператором `|`. Необязательный параметр `$context` позволяет задать HTTP-заголовки, отправляемые сервером при обращении к сетевому файлу. Если это не требуется, вместо него можно передать `NULL`. Необя-

зательные параметры `$offset` и `$maxlen` задают соответственно начальную позицию и количество символов, которые следует прочитать.

> ***Замечание***
>
> Константы `FILE_TEXT` и `FILE_BINARY` являются взаимоисключающими и не могут использоваться одновременно.

Таблица 4.4. *Возможные значения параметра* `$flags` *функции* `file_get_contents()`

Значение	Описание
`FILE_USE_INCLUDE_PATH`	Поиск файла в каталогах, указанных в директиве `include_path` конфигурационного файла php.ini
`FILE_TEXT`	Данные возвращаются в кодировке UTF-8 (если этот режим включен в конфигурационном файле php.ini)
`FILE_BINARY`	Чтение осуществляется в бинарном режиме, без преобразования содержимого файла

В листинге 4.17 демонстрируется пример, аналогичный по результату скрипту из листинга 4.15.

> ***Замечание***
>
> Функция `file_get_contents()`, как правило, используется для работы с небольшими файлами. Это связано с тем, что на один скрипт отводится ограниченный объем памяти — 8 или 16 Мбайт в зависимости от значения директивы `memory_limit` в конфигурационном файле php.ini. При попытке чтения файла объемом больше `memory_limit` работа скрипта будет остановлена, и возвращена ошибка "Fatal error: Allowed memory size of 8388608 bytes exhausted" ("Критическая ошибка: израсходовано 8 388 608 байтов зарезервированной памяти").

Листинг 4.17. Использование функции `file_get_contents()`

```
<?php
  $str = file_get_contents("text.txt");
  echo $str;
?>
```

Помимо локальных файлов, функция `file_get_contents()` способна к чтению сетевых файлов. В листинге 4.18 приводится альтернативная реализация скрипта.

Листинг 4.18. Чтение сетевых файлов при помощи функции `file_get_contents()`

```
<?php
  $str = file_get_contents("http://www.softtime.ru");
  echo $str;
?>
```

Если полученное при помощи функции `file_get_contents()` содержимое файла необходимо разбить построчно, можно воспользоваться функцией `explode()` (листинг 4.19).

Листинг 4.19. Получение содержимого файла в виде массива

```
<?php
  // Получаем содержимое файла
  $contents = file_get_contents("text.txt");
  // Разбиваем содержимое файла на отдельные строки
  $arr = explode("\n", $contents);
  // Выводим дамп массива $arr
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 4.19 будет массив `$arr`, каждый элемент которого будет соответствовать строке файла. Для получения содержимого файла в виде массива в PHP предусмотрена специальная функция `file()`, которая имеет следующий синтаксис:

```
file($filename [, $flags [, $context]])
```

Функция читает файл с именем `$filename` и возвращает его содержимое в виде массива, каждый элемент которого соответствует отдельной строке. Необязательный параметр `$flags` позволяет задать режим чтения файла и может принимать значения из табл. 4.5. Параметр `$context` позволяет задать HTTP-заголовки, отправляемые сервером при обращении к сетевому файлу.

ЗАМЕЧАНИЕ

Константы `FILE_TEXT` и `FILE_BINARY` являются взаимоисключающими и не могут использоваться одновременно.

***Таблица 4.5.** Возможные значения параметра `$flags` функции `file()`*

Значение	Описание
`FILE_USE_INCLUDE_PATH`	Поиск файла в каталогах, указанных в директиве `include_path` конфигурационного файла php.ini
`FILE_IGNORE_NEW_LINES`	Не помещать в конец каждого элемента результирующего массива пустой строки
`FILE_SKIP_EMPTY_LINES`	Игнорировать пустые строки
`FILE_TEXT`	Данные возвращаются в кодировке UTF-8 (если этот режим включен в конфигурационном файле php.ini)
`FILE_BINARY`	Чтение осуществляется в бинарном режиме, без преобразования содержимого файла

В листинге 4.20 приводится альтернативный вариант для скрипта из листинга 4.19.

Листинг 4.20. Использование функции `file()`

```
<?php
  // Получаем содержимое файла в виде массива
  $arr = file ("text.txt", FILE_IGNORE_NEW_LINES |
                           FILE_SKIP_EMPTY_LINES |
                           FILE_BINARY);
  // Выводим дамп массива $arr
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

4.5. Запись файлов

Для записи в файл его необходимо открыть (при помощи функции `fopen()`) в любом режиме, кроме `r`. Впрочем, для записи нового файла традиционно используются режимы `w` или `w+`, которые помещают файловый указатель в начало файла. Записывать информацию в файл можно при помощи функции `fwrite()`, которая имеет следующий синтаксис:

```
fwrite($handle, $str [, $length])
```

Функция записывает в открытый файл `$handle` содержимое строки `$str`. Необязательный параметр `$length` позволяет задать количество байтов, предназначенных для записи (по достижению этого количества запись останавливается). В случае успеха функция возвращает количество записанных байтов, в случае неудачи — `FALSE`. В листинге 4.21 демонстрируется скрипт, записывающий в файл text.txt строку "Hello world!".

Листинг 4.21. Запись строки в файл

```
<?php
  // Открываем файл для записи
  $fd = fopen("text.txt", "w");
  if (!$fd) exit("Невозможно открыть файл на запись");
  // Записываем строку "Hello world!"
  fwrite($fd, "Hello world!");
  // Закрываем файл
  fclose($fd);
?>
```

Помимо режимов, представленных в табл. 4.2, существуют дополнительные режимы `b` и `t` для бинарного и текстовых режимов открытия файлов. Эти режимы не яв-

ляются самостоятельными и применяются только с одним из основных режимов, представленных в табл. 4.2.

Эти флаги введены из-за того, что в разных операционных системах используются различные обозначения конца строки. В UNIX-подобных операционных системах это последовательность \n, в Windows — \r\n, в Mac OS — \n\r. *Режим трансляции* t позволяет автоматически транслировать UNIX-перевод строки \n в перевод строки текущей операционной системы. *Бинарный режим* b отключает такое поведение и записывает файлы как есть.

> ***Замечание***
>
> Программа Блокнот в Windows признает только Windows-переводы строки \r\n, а UNIX-перевод строки \n отображается как нечитаемый символ (квадратик).

В повседневной практике рекомендуется придерживаться бинарного режима b, т. к. режим трансляции зачастую приводит к искажению бинарных файлов.

Различие между режимом трансляции и бинарным режимом удобно отслеживать в операционной системе Windows. Пусть имеется строка "Hello\nworld!", содержащая UNIX-перевод строки \n. Запись такой строки в бинарном режиме (листинг 4.22) создаст файл длиной в 12 символов.

Листинг 4.22. Запись файла в бинарном режиме

```
<?php
  // Открываем файл для записи
  $fd = fopen("text.txt", "wb");
  if (!$fd) exit("Невозможно открыть файл на запись");
  // Записываем строку в файл
  fwrite($fd, "Hello\nworld!");
  // Закрываем файл
  fclose($fd);
  // Подсчитываем количество символов в файле
  $str = file_get_contents("text.txt");
  echo strlen($str); // 12
?>
```

Запись строки "Hello\nworld!" в режиме трансляции создаст файл длиной 13 символов, т. к. UNIX-перевод строки \n будет преобразован в \r\n (листинг 4.23).

Листинг 4.23. Запись файла в режиме трансляции

```
<?php
  // Открываем файл для записи
  $fd = fopen("text.txt", "wt");
  if (!$fd) exit("Невозможно открыть файл на запись");
  // Записываем строку в файл
  fwrite($fd, "Hello\nworld!");
```

```
  // Закрываем файл
  fclose($fd);
  // Подсчитываем количество символов в файле
  $str = file_get_contents("text.txt");
  echo strlen($str); // 13
?>
```

Помимо записи файла при помощи файлового дескриптора, допускается перезапись всего содержимого файла посредством специализированной функции `file_put_contents()`, которая имеет следующий синтаксис:

```
file_put_contents($filename, $data [, $flags [, $context]])
```

Функция записывает в файл с именем `$filename` данные из параметра `$data` (либо строку, либо одномерный массив). Необязательный параметр `$flags` определяет режим записи файла и может принимать значения из табл. 4.6. Параметр `$context` позволяет задать HTTP-заголовки, отправляемые сервером при обращении к сетевому файлу.

> ***Замечание***
>
> Константы `FILE_TEXT` и `FILE_BINARY` являются взаимоисключающими и не могут использоваться одновременно.

Таблица 4.6. *Возможные значения параметра* `$flags` *функции* `file_put_contents ()`

Значение	Описание
`FILE_USE_INCLUDE_PATH`	Поиск файла в каталогах, указанных в директиве `include_path` конфигурационного файла php.ini
`FILE_APPEND`	Данные не перезаписывают содержимое файла, а добавляются в конец файла
`LOCK_EX`	Запрашивает исключительный доступ к файлу для записи информации, сторонние скрипты не смогут открыть файл для записи
`FILE_TEXT`	Данные возвращаются в кодировке UTF-8 (если этот режим включен в конфигурационном файле php.ini)
`FILE_BINARY`	Чтение осуществляется в бинарном режиме, без преобразования содержимого файла

В листинге 4.24 демонстрируется альтернативный вариант скрипта из листинга 4.19.

Листинг 4.24. Использование функции `file_put_contents()`

```
<?php
  file_put_contents("text.txt", "Hello world!");
?>
```

В качестве второго аргумента функции допускается одномерный массив, элементы которого перед записью объединяются в строку (листинг 4.25).

Листинг 4.25. Запись одномерного массива в файл

```
<?php
  // Формируем массив
  $arr[] = "Hello ";
  $arr[] = "world!";
  // Записываем массив в файл
  file_put_contents("text.txt", $arr);
?>
```

4.6. Размер файла

Для определения размера файла предназначена функция `filesize()`, которая возвращает размер файла в байтах. Для того чтобы возвратить размер в байтах, килобайтах или мегабайтах в зависимости от размера, необходимо разработать функцию, схожую с той, что приведена в листинге 4.26.

Листинг 4.26. Функция определения размера файла

```
function getfilesize($filename)
{
  // Проверяем, существует ли файл
  if (!file_exists($filename)) return "файл не существует";
  // Определяем размер файла
  $filesize = filesize($filename);
  // Если размер файла превышает 1024 байта,
  // пересчитываем размер в килобайты
  if ($filesize > 1024)
  {
    $filesize = (float)($filesize/1024);
    // Если размер файла превышает 1024 Кбайт,
    // пересчитываем размер в мегабайты
    if ($filesize > 1024)
    {
      $filesize = (float)($filesize/1024);
      // Округляем дробную часть до первого знака после запятой
      $filesize = round($filesize, 1);
      return $filesize." Мб";
    }
    else
    {
      // Округляем дробную часть до первого знака после запятой
      $filesize = round($filesize, 1);
```

```
      return $filesize." Кб";
    }
  }
  else
  {
    return $filesize." байт";
  }
}
?>
```

Как видно из листинга 4.26, перед тем как перейти к вычислениям, при помощи функции `file_exists()` была осуществлена проверка файла. Функция возвращает `TRUE`, если файл или каталог существует, и `FALSE` в противном случае. Скрипт принимает в качестве параметра имя файла и последовательно проверяет, не превышает ли объем файла, возвращенный функцией `filesize()`, 1 Кбайт, затем 1 Мбайт. В каждом из случаев возвращается соответствующий результат.

4.7. Разбивка файла на части

Пусть имеется файл site.rar, который необходимо разбить на части по 10 000 байтов. Скрипт, выполняющий эту задачу, может выглядеть следующим образом (листинг 4.27).

Листинг 4.27. Разбивка файла на части

```
<?php
  // Имя файла
  $filename = "site.rar";
  // Разбиваем файл на части по 10 000 байтов
  $piece = 10000;
  // Открываем исходный файл для чтения
  $fp = fopen($filename, "r");
  // Читаем содержимое файла в буфер
  $bufer = file_get_contents($filename);
  // Закрываем файл
  fclose($fp);
  // Подсчитываем количество частей, на которые необходимо разбить файл
  $count = (int)filesize($filename)/$piece;
  if((float)(filesize($filename)/$piece) — $count != 0) $count++;
  // В цикле разбиваем содержимое файла в переменной
  // $bufer на части
  for($i=0; $i<$count; ++$i)
  {
    $part = substr($bufer,$i*$piece,$piece);
    // Сохраняем текущую часть в отдельном файле
    file_put_contents("site.tm".$i, $part);
  }
?>
```

Скрипт разбивает файл на несколько частей, каждая из которых получает расширение tmN, где N — номер части. Обратную задачу по сбору файла из отдельных частей выполняет скрипт, приведенный в листинге 4.28.

Листинг 4.28. Объединение частей файла в единое целое

```
<?php
  $bufer = "";
  for($i = 0; $i < 100000; ++$i)
  {
    // Генерируем имя файла
    $filename = "site.tm".$i;
    // Собираем содержимое файла
    if(file_exists($filename))
      // Если такой файл существует,
      // добавляем его содержимое к $bufer
      $bufer .= file_get_contents($filename);
    else
      // Если файл с таким именем не
      // существует — выходим из цикла
      break;
    // Склеенные в переменной $bufer
    // части помещаем в конечный файл
    file_put_contents("site_final.rar", $bufer);
  }
?>
```

4.8. Редактирование файлов на удаленном сервере

Все приведенные выше примеры решали задачу автоматического преобразования файлов. Однако зачастую требуется ручное редактирование файлов, особенно если речь заходит о редактировании конфигурационных файлов и скриптов. На рис. 4.3 приводится пример небольшого Web-приложения, которое отображает содержимое в текстовой области и позволяет записать результаты редактирования.

В листинге 4.29 приводится одна из возможных реализаций такого скрипта.

Листинг 4.29. Загрузка произвольного количества файлов на сервер

```
<?php
  // Выставляем уровень обработки ошибок
  error_reporting(E_ALL & ~E_NOTICE);
  // Если передано исправленное содержимое файла,
  // открываем файл и перезаписываем его
  if(isset($_POST['content']))
```

```
  {
    // Проверяем, существует ли файл
    if(!file_exists($_POST['filename']))
      exit("Файл с таким именем отсутствует");
    // Проверяем, включен ли режим "магических кавычек"
    if(get_magic_quotes_gpc())
    {
      // Удаляем преобразование режима "магических кавычек"
      $_POST['content'] = stripslashes($_POST['content']);
    }
    // Перезаписываем содержимое файла
    file_put_contents($_POST['filename'], $_POST['content']);
    // Помещаем в суперглобальный массив $_GET
    // имя файла
    $_GET['filename'] = $_POST['filename'];
  }
  echo "<form method='get'>".
       "Имя файла ".
       "<input type='text' name='filename' ".
       "value='".htmlspecialchars($_GET['filename'])."' />";
  echo "<input type='submit' value='Открыть' />";
  echo "</form>";
  // Если в строке запроса передано имя
  // файла, открываем его для редактирования
  if(isset($_GET['filename']))
  {
    // Проверяем, существует ли файл
    if(!file_exists($_GET['filename']))
      exit("Файл с таким именем отсутствует");
    // Помещаем содержимое файла в переменную $bufer
    $bufer = file_get_contents($_GET['filename']);
    echo "<form method='post'>".
         "<textarea cols='50' rows='5' name='content'>".
         htmlspecialchars($bufer)."</textarea>".
         "<input type='hidden' name='filename' ".
         "value='".htmlspecialchars($_GET['filename'])."' />";
    echo "<br /><br />";
    echo "<input type='submit' value='Редактировать' />";
    echo "</form>";
  }
?>
```

При первой загрузке скрипта открывается форма, имеющая единственное текстовое поле `filename` для имени файла и кнопку, отправляющую данные методом GET обработчику, который расположен в этом же файле. При помощи функции `file_get_contents()` происходит загрузка содержимого файла во временную переменную `$bufer`, содержимое которой помещается в текстовую область второй фор-

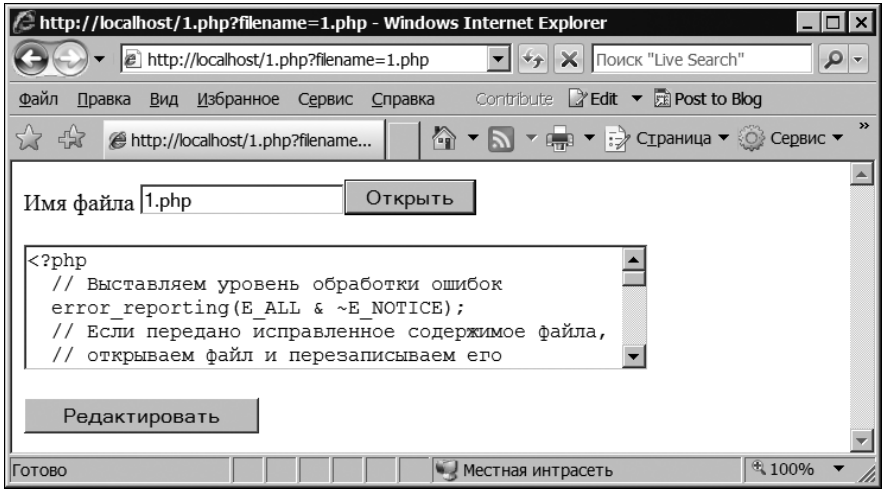


Рис. 4.3. Редактирование файлов на удаленном сервере

мы. Помимо текстовой области `content` форма содержит скрытое поле `filename`, через которое передается имя файла. После редактирования и нажатия на кнопку второй формы данные повторно отправляются текущему скрипту, но уже методом POST. В начале скрипта размещен обработчик, который в том случае, если элемент `$_POST['content']` принимает непустое значение, переписывает содержимое редактируемого файла. Для корректного поведения скрипта в конце обработчика имя файла из суперглобального массива `$_POST` переписывается в суперглобальный массив `$_GET`.

Выяснить, включен режим "магических кавычек" или нет, можно при помощи функции `get_magic_quotes_gpc()`. Режим "магических кавычек" *(см. разд. 2.1.9)* удобен, если переменная, подвергнутая такому преобразованию, вставляется в строку. В листинге 4.29 такое преобразование является лишним и должно быть отменено, например, при помощи функции `stripslashes()`. Если бы переменная `$_POST['content']` была заключена в кавычки, то потребовалась бы обратная задача — в случае, если режим "магических кавычек" не включен, добавить экранирование (листинг 4.30).

> ***Замечание***
>
> Следует обратить внимание, что в листинге 4.30 функцию `get_magic_quotes_gpc()` предваряет отрицание !, т. к. действие в теле оператора `if` необходимо выполнить, если режим "магических кавычек" отключен, в то время как в листинге 4.31 отрицание отсутствует, т. к. действие необходимо выполнить, если режим "магических кавычек" включен.

Листинг 4.30. Фрагмент скрипта

```
<?php
  ...
  // Проверяем, включен ли режим "магических кавычек"
  if(!get_magic_quotes_gpc())
```

```
  {
    // Удаляем преобразование режима "магических кавычек"
    $_POST['content'] = addslashes($_POST['content']);
  }
  // Перезаписываем содержимое файла
  file_put_contents($_POST['filename'], "$_POST[content]");
  ...
?>
```

4.9. Счетчик загрузок файлов

Работа всех счетчиков загрузок файлов основана на том, что посетителю в качестве ссылки для загрузки предоставляется не сам файл, а ссылка на скрипт, который учитывает загрузку и отправляет файл браузеру пользователя.

Будем строить наш счетчик таким образом, чтобы ссылки на загрузку файла представляли собой ссылки на текущую страницу с передачей в качестве параметра имени файла, например, `index.php?down=archive.zip`. Скрипт будет проверять, передан ли параметр `down`, и если это так, то будет регистрировать загрузку архива в файле filecount.txt. При повторной загрузке страницы из файла будут извлекаться значения счетчиков для каждого из архивов для вывода в окно браузера. Передача файла на загрузку будет осуществляться отсылкой посетителю HTTP-заголовка `Location` с указанием пути к загружаемому архиву. Скрипт счетчика загрузок файлов может выглядеть так, как это представлено в листинге 4.31.

Листинг 4.31. Счетчик загрузок файлов

```
<?php
// Выставляем уровень обработки ошибок
error_reporting(E_ALL & ~E_NOTICE);
// Регистрируем имена файлов в массиве
$file_name = array('archive1.zip','archive2.zip','archive3.zip');

// Имя файла, где хранится статистика
$countname = "filecount.txt";
// Если файл существует,
// считываем текущую статистику в массив
if(file_exists($countname))
{
 // Получаем содержимое счетчика
 $content = file_get_contents($countname);
 // Распаковываем массив
 $count = unserialize($content);
}
// Если такого файла нет — создаем его,
// а статистику обнуляем
```

```
else
{
 // Заполняем массив $count нулевыми значениями
 foreach($file_name as $file)
 {
  $count[$file] = 0;
 }
 // Упаковываем массив и помещаем его в счетчик
 file_put_contents($countname, serialize($count));
}

// Проверяем, не передано ли значение параметра down
// через метод GET
if(isset($_GET['down']))
{
 // Проверяем, входит ли значение параметра $_GET['down']
 // в массив $file_name
 if(in_array($_GET['down'],$file_name))
 {
  // Регистрируем факт загрузки данного файла
  // Увеличиваем значение счетчика с ключом
  // $_GET['down'] на единицу
  $count[$_GET['down']]++;
  // Перезаписываем файл счетчика
  file_put_contents($countname, serialize($count));

  // Предоставляем ссылку на его загрузку
  header("Location: $_GET[down]");
  exit();
 }
}
// Выводим ссылки на файлы
foreach($file_name as $file)
{
 echo "Файл <a href='$_SERVER[PHP_SELF]?down=$file'>$file</a>
    был загружен ".intval($count[$file])." раз<br />";
}
?>
```

Имена загружаемых файлов хранятся в массиве `$file_name`, добавление нового архива приводит к автоматическому его учету в системе. Предварительная регистрация в массиве необходима по нескольким причинам. Во-первых, принимая имя массива через параметр `down`, необходимо проверить, входит ли он в число файлов, разрешенных к загрузке. Во-вторых, обрабатывать имена файлов в массиве гораздо удобнее. Так массив `$count`, в котором хранится количество загрузок файлов, автоматически строится на основании массива зарегистрированных в системе файлов,

массив удобно упаковывать при помощи функции `serialize()` в строку, а затем распаковывать обратно в массив при помощи функции `unserialize()`.

> ***Замечание***
>
> Важно помнить, что отправка всех HTTP-заголовков должна осуществляться до отправки основного содержимого, иначе их отправка будет невозможна и интерпретатор PHP выдаст предупреждение "Warning: Cannot modify header information — headers already sent by" (Предупреждение: невозможно модифицировать заголовочную информацию — заголовки уже отправлены). Это диктуется протоколом HTTP: сначала отправляются заголовки, затем содержимое документа, поэтому любой вывод в окно браузера воспринимается как окончание отправки заголовков и начало отправки тела документа. Если вывод в окно браузера до отправки заголовков неизбежен, необходимо прибегать к функциям управления выводом, помещая весь вывод в буфер и отправляя его в конце работы скрипта.

Как видно из листинга 4.31, в скрипте обрабатывается ситуация первого запуска, когда отсутствует файл filecount.txt — он автоматически создается при первой загрузке страницы, инициированный нулевыми значениями для каждого файла из массива `$file_name`. Результат работы скрипта из листинга 4.31 можно видеть на рис. 4.4.

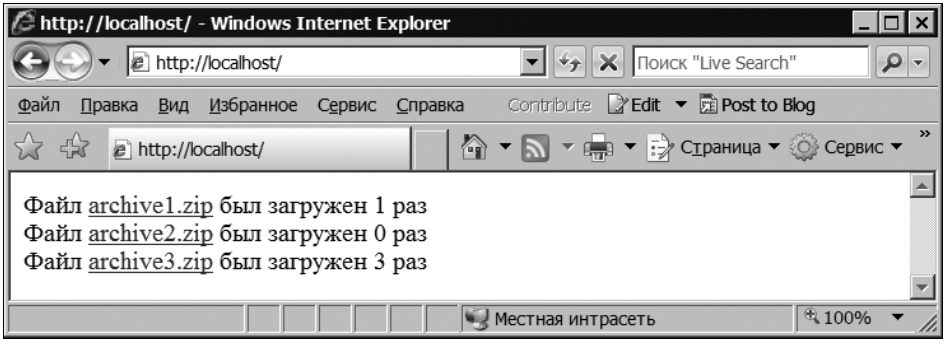


Рис. 4.4. Результат работы счетчика файлов

4.10. Сохранение текстовых и графических файлов

Переход по ссылке на текстовые файлы или файлы HTML приводит к отображению их в окне браузера, что не всегда удобно, особенно, если файл предназначен для загрузки. Та же участь ожидает графические файлы и вообще любые файлы, которые браузер может отобразить. О содержимом файла браузер посетителя "узнает" от сервера, т. к. каждый файл сопровождается HTTP-заголовками, сообщающими клиенту о содержимом, размере загружаемого файла, необходимости установить cookie и т. п. Если тип файла не удалось определить, он отправляется просто как двоичный поток.

Подавить такое поведение можно, отправив HTTP-заголовки, представленные в листинге 4.32.

Листинг 4.32. Скрипт, позволяющий сохранять текстовые и графические файлы

```
<?php
  $filename = basename($_GET['down']);
  header("Content-Disposition: attachment; filename=$filename");
  header("Content-type: application/octet-stream");
  header("Content-length: ".filesize($_GET['down']));
  echo file_get_contents($_GET['down']);
?>
```

Скрипт в листинге 4.32 принимает в качестве GET-параметра имя файла, например, `index.php?down=filetext.txt`. При помощи функции `basename()` извлекается имя файла (на тот случай, если GET-параметр `down` содержит путь к файлу). HTTP-заголовок `Content-Disposition` задает имя сохраняемого файла, которое определяется атрибутом `filename`. В приведенном скрипте параметр `filename` совпадает с именем отправляемого файла, однако в качестве параметра `filename` может быть передано произвольное имя. HTTP-заголовок `Content-type` сообщает о том, что передаваемые данные являются двоичными, и браузеру их не следует интерпретировать. HTTP-заголовок `Content-length` передает клиенту размер файла. В последней строке выводится содержимое файла, переданное через параметр `$_GET['down']`, которое извлекается при помощи функции `file_get_contents()`. Результат работы скрипта из листинга 4.32 представлен на рис. 4.5.

ЗАМЕЧАНИЕ

Важно, чтобы после вывода содержимого файла больше ничего не выводилось в поток: ни конструкцией `echo`, ни непосредственным выводом — иначе все будет прикреплено в конец файла. Это относится и к возможным пробелам, и к переводам строк после завершающего тега `?>`.

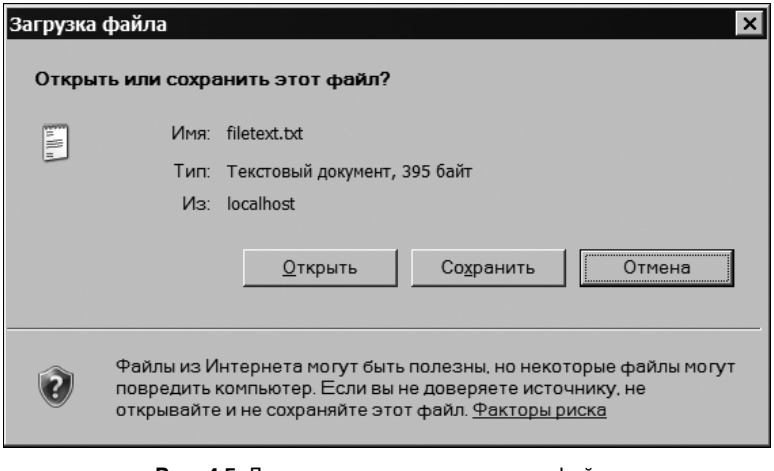


Рис. 4.5. Диалоговое окно для загрузки файла

4.11. Определение количества строк в файле

Обычно файлы, с которыми приходится иметь дело Web-разработчикам, являются текстовыми. Такие файлы, как правило, обрабатываются построчно, и одной из распространенных для них является операция подсчета количества строк в файле.

Самым простым решением проблемы будет разбивка содержимого файла на строки при помощи стандартной функции `file()`, которая принимает в качестве аргумента имя файла и возвращает массив, каждый элемент которого содержит отдельную строку файла. После этого для подсчета строк в файле остается только подсчитать количество элементов полученного массива (листинг 4.33).

Листинг 4.33. Определение количества строк в файле

```
<?php
echo getlinecount($_GET['name']);

function getlinecount($filename)
{
  // Проверяем, существует ли файл
  if(!file_exists($filename)) return "файл не существует";
  // Разбиваем содержимое массива на отдельные строки
  // при помощи функции file(), которая возвращает массив,
  // каждый элемент которого содержит строку файла
  $filearr = file($filename);
  // Возвращаем количество строк в файле
  return count($filearr);
}
?>
```

Решение, представленное в листинге 4.33, успешно работает в том случае, если размер текстового файла не превышает размера оперативной памяти, отводимой скрипту. В противном случае для подсчета количества строк в файле следует прибегать к файловому дескриптору (листинг 4.34).

Листинг 4.34. Альтернативное решение

```
<?php
echo getlinecount($_GET['name']);

function getlinecount($filename)
{
  // Проверяем, существует ли файл
  if(!file_exists($filename)) return "файл не существует";
  // Открываем файл
  $fd = fopen($filename, "r");
```

```
  // Проверяем корректность открытия файла
  if(!$fd) return "невозможно открыть файл";

  $count = 0;
  while(($line = fgets($fd)) !== false) $count++;

  // Возвращаем количество строк в файле
  return $count;
}
?>
```

Функция `getlinecount()` не загружает содержимое файла в оперативную память, а последовательно считывает строку за строкой. Поэтому в каждый момент времени объем, занимаемый скриптом, не критичен.

4.12. Случайный вывод из файла

Пусть имеется текстовый файл text.txt (листинг 4.35), и стоит задача вывести любую выбранную в случайном порядке строку.

Листинг 4.35. Содержимое файла text.txt

```
PHP
C++
Java
Ruby
Python
Perl
JavaScript
Visual Basic
Fortran
Pascal
Assembler
Lisp
Haskell
C#
```

Для решения данной задачи необходимо использование функции `rand()`, которая имеет следующий синтаксис:

```
rand([$min, $max])
```

Функция генерирует случайное целое число между двумя целочисленными параметрами `$min` и `$max`. Если необязательные параметры `$min` и `$max` не указаны, число будет расположено между 0 и значением `RAND_MAX` (равным 32 768).

Скрипт, осуществляющий случайный вывод из файла, представлен в листинге 4.36.

Листинг 4.36. Случайный вывод из файла

```
<?php
  // Имя файла
  $filename = "text.txt";
  // Помещаем содержимое файла count.txt
  // в массив $lines
  $lines = file($filename);
  // Генерируем случайный индекс массива $lines
  $index = rand(0, count($lines) — 1);
  // Выводим строку номер $index
  echo $lines[$index];
?>
```

4.13. Сортировка содержимого текстового файла

Одной из распространенных задач при работе с текстовыми файлами является сортировка содержимого файла. Для тестирования сортировки подготовим вспомогательный скрипт, перемешивающий записи файла в случайном порядке. Для этого строки извлекаются при помощи функции `file()`, а затем перемешиваются при помощи функции `shuffle()` (листинг 4.37).

Листинг 4.37. Перемешивание записей в файле

```
<?php
  // Имя файла
  $filename = "text.txt";
  // Читаем содержимое файла
  $lines = file($filename);
  // Перемешиваем записи случайным образом
  shuffle($lines);
  // Удаляем все пробельные символы
  // в конце строки
  array_walk($lines, 'trim_array');
  // Сохраняем результат в файле
  file_put_contents($filename, implode("\r\n", $lines));

  function trim_array(&$item, $key)
  {
    $item = trim($item);
  }
?>
```

При помощи функции `array_walk()` и функции обратного вызова `trim_array()` каждый элемент промежуточного массива `$lines` пропускается через функцию `trim()`,

которая удаляет возможные пробельные символы в начале и конце строки, в том числе и символы перевода строки.

В результате работы скрипта из листинга 4.37 файл text.txt может выглядеть так, как это представлено в листинге 4.38.

Листинг 4.38. Записи файла text.txt в случайном порядке

```
C#
Ruby
Assembler
Haskell
Java
Pascal
JavaScript
Python
Visual Basic
Perl
PHP
C++
Lisp
Fortran
```

Для сортировки полученного файла достаточно разбить его содержимое на строки при помощи функции `file()` и отсортировать полученный массив стандартной функцией `sort()` или `rsort()`, в прямом или обратном порядке (листинг 4.39).

Листинг 4.39. Сортировка записей файла text.txt по индексу

```
<?php
  // Имя файла
  $filename = "text.txt";
  // Читаем содержимое файла
  $lines = file($filename);
  // Сортируем записи по индексу
  sort($lines);
  // Удаляем все пробельные символы
  // в конце строки
  array_walk($lines, 'trim_array');
  // Сохраняем результат в файле
  file_put_contents($filename, implode("\r\n", $lines));

  function trim_array(&$item, $key)
  {
    $item = trim($item);
  }
?>
```

4.14. Каталоги

Каталогами называют специальный тип файлов, которые могут содержать ссылки на другие файлы и каталоги. Однако править содержимое таких файлов непосредственно невозможно — это задача ядра операционной системы. Разработчикам предоставляется набор функций, позволяющих создавать, удалять каталоги, а также читать их содержимое (табл. 4.7).

Таблица 4.7. *Функции для работы с каталогами*

Функция	Описание
`mkdir($pathname [, $mode [, $recursive [, $context]]])`	Создает каталог `$pathname` с правами доступа `$mode`. Если необязательный параметр `$recursive` принимает значение `TRUE`, все несуществующие каталоги в пути `$pathname` создаются. Параметр `$context` позволяет задать заголовки, отправляемые сервером при обращении к сетевому файлу. При успешном создании каталога функция возвращает `TRUE`, в противном случае — `FALSE`
`rmdir($dirname [, $context])`	Удаляет пустой каталог с именем `$dirname`. При успешном удалении каталога функция возвращает `TRUE`, в противном случае — `FALSE`
`getcwd()`	Возвращает полный путь к текущему каталогу
`chdir ($directory)`	Изменяет текущий каталог на новый, указанный в параметре `$directory`. При успешной смене текущего каталога функция возвращает `TRUE`, в противном случае — `FALSE`
`chroot($directory)`	Изменяет корневой каталог текущего процесса на `$directory`. При успешной смене корневого каталога возвращает `TRUE`, в противном случае — `FALSE`
`opendir($path [, $context])`	"Открывает" каталог — функция возвращает дескриптор, позволяющий читать содержимое каталога `$path`
`readdir($dir_handle)`	Читает одну позицию из каталога и передвигает дескриптор на одну позицию вперед. Последовательный вызов функции позволяет прочитать содержимое каталога файл за файлом. Функция принимает в качестве единственного параметра `$dir_handle` дескриптор, полученный ранее при помощи функции `opendir()`. Если достигнут конец каталога, функция возвращает `FALSE`
`closedir($dir_handle)`	Закрывает дескриптор `$dir_handle`, открытый ранее при помощи функции `closedir()`
`rewinddir($dir_handle)`	Помещает дескриптор `$dir_handle` в начало каталога
`scandir($directory [, $sorting_order [, $context]])`	Возвращает массив с содержимым каталога `$directory` или `FALSE`, если `$directory` не является каталогом. Содержимое каталога сортируется в алфавитном порядке, если необязательный параметр `$sorting_order` принимает значение `TRUE`, элементы результирующего массива сортируются в обратном алфавитном порядке
`glob($pattern [, $flags])`	Возвращает массив с именами файлов и подкаталогов, путь к которым удовлетворяет шаблону `$pattern`. Необязательный параметр `$flags` позволяет задать режим интерпретации шаблона

В начале выполнения скрипта ему назначается текущий каталог, именно от него отсчитывается относительный путь, в нем создаются файлы и каталоги. При работе интерпретатора PHP в составе Web-сервера Apache в качестве такого каталога зачастую выступает каталог, в котором расположен скрипт. В любом случае всегда можно уточнить текущий полный путь при помощи функции `getcwd()` (листинг 4.40).

Листинг 4.40. Использование функции `getcwd()`

```
<?php
  // Получаем абсолютный путь к текущему каталогу
  echo getcwd();
?>
```

При необходимости можно назначить новый текущий каталог при помощи функции `chdir()` (листинг 4.41). Такое переназначение особенно актуально при запуске скриптов по cron-заданию или из командной строки, когда текущий каталог по умолчанию может не совпадать с каталогом, где расположен скрипт.

Листинг 4.41. Использование функции `chdir()`

```
<?php
  // Устанавливаем новый каталог по умолчанию
  chdir("/var/www/html/test/");
?>
```

В листинге 4.42 при помощи функции `mkdir()` в текущем каталоге создается новый подкаталог каталог new. Особенностью функции `mkdir()` является то, что второй параметр позволяет сразу же выставить права доступа на вновь создаваемый каталог.

Листинг 4.42. Использование функции `mkdir()`

```
<?php
  if (mkdir("new", 0700)) echo "Каталог успешно создан";
  else echo "Ошибка создания каталога";
?>
```

4.15. Список файлов и подкаталогов в каталоге

Пожалуй, самой распространенной задачей, связанной с каталогами, является получение списка входящих в него файлов и подкаталогов (листинг 4.43).

Листинг 4.43. Вывод списка файлов каталога

```
<?php
  // Открываем каталог
  $dir = opendir(".");
  // В цикле выводим его содержимое
  while (($file = readdir($dir)) !== FALSE) echo "$file<br>";
  // Закрываем каталог
  closedir($dir);
?>
```

При просмотре каталога необходимо явно сравнивать результат операции `$file = readdir($dir)` с `FALSE`, поскольку если файл или подкаталог называется любым именем, начинающимся с нуля, то возвращаемое функцией `readdir()` имя может быть приведено к `FALSE`, и цикл обхода может закончиться раньше времени.

Вместо использования комбинации функций `opendir()` и `readdir()` имена файлов и подкаталогов можно также получить при помощи функции `scandir()`, которая имеет следующий синтаксис:

```
scandir($directory [, $sorting_order [, $context]])
```

Функция возвращает массив с содержимым каталога `$directory` или `FALSE`, если `$directory` не является каталогом. Содержимое каталога сортируется в алфавитном порядке, если необязательный параметр `$sorting_order` принимает значение `TRUE`, элементы результирующего массива сортируются в обратном алфавитном порядке.

В листинге 4.44 демонстрируется простой пример, возвращающий массив с именами файлов и каталогов в заданном каталоге.

Листинг 4.44. Использование функции `scandir()`

```
<?php
  $dir = 'dir';
  // сортировка по возрастанию
  $files = scandir($dir);
  // сортировка по убыванию
  $sort_files = scandir($dir, 1);
  // Выводим содержимое массивов
  echo "<pre>";
  print_r($files);
  print_r(sort_files);
  echo "</pre>";
?>
```

Результат работы скрипта из листинга 4.44 может выглядеть следующим образом:

```
Array
(
    [0] => array.php
```

```
    [1] => file.txt
    [2] => somedir
)
Array
(
    [0] => somedir
    [1] => file.txt
    [2] => array.php
)
```

Еще более гибкие возможности для получения содержимого массива предоставляет функция `glob()`, которая имеет следующий синтаксис:

```
glob($pattern [, $flags])
```

Функция возвращает массив с именами файлов и подкаталогов, путь к которым удовлетворяет шаблону `$pattern`. Необязательный параметр `$flags` принимает одно или несколько значений из табл. 4.8, позволяя задать режим интерпретации шаблона.

***Таблица 4.8.** Возможные значения параметра `$flags` функции `glob()`*

Значение	Описание
`GLOB_MARK`	Добавляет прямой слэш `/` в UNIX-подобных операционных системах и обратный слэш `\` в Windows к тем элементам результирующего массива, которые являются каталогами
`GLOB_NOSORT`	По умолчанию результирующий массив сортируется по алфавиту, использование этого флага отменяет такую сортировку
`GLOB_NOCHECK`	Возвращает шаблон поиска, если не было найдено ни одно соответствие
`GLOB_NOESCAPE`	Имена файлов могут содержать специальные символы, такие как `*`,`?` или `$`. По умолчанию эти символы экранируются при помощи обратного слэша. Использование флага `GLOB_NOCHECK` позволяет отменить экранирование
`GLOB_BRACE`	Позволяет задать несколько вариантов в фигурных скобках через запятую, например `{*.html,*.htm,*.php}` вернет все файлы, которые имеют расширения html, htm или php
`GLOB_ONLYDIR`	В результирующий массив попадают только подкаталоги — файлы игнорируются
`GLOB_ERR`	По умолчанию ошибки доступа игнорируются. Флаг `GLOB_ERR` предписывает остановить сканирование каталога и вывести сообщение об ошибке

В листинге 4.45 демонстрируется пример использования функции `glob()`.

Листинг 4.45. Использование функции `glob()`

```
<?php
  // Получаем все HTML- и PHP-файлы текущего каталога
  $arr = glob("{*.html,*.htm,*.php}", GLOB_BRACE | GLOB_MARK);
```

```
  // Выводим список файлов
  if (is_array($arr))
  {
    foreach ($arr as $filename)
      echo "$filename (".filesize($filename)." байт)<br>";
  }
?>
```

При обходе каталога и всех вложенных подкаталогов удобнее использовать рекурсивный спуск по дереву каталогов (листинг 4.46). Для этого будем вызывать функцию `scan_dir()`, которая будет извлекать содержимое текущего каталога, осуществляя вывод имен файлов и каталогов. Если функция будет встречать подкаталог, то она будет рекурсивно вызывать себя, передавая в качестве параметра имя подкаталога.

Листинг 4.46. Список файлов и подкаталогов в каталоге

```
<?php
  ////////////////////////////////////////////////////////
  // Рекурсивная функция — спускаемся вниз по каталогу
  ////////////////////////////////////////////////////////
  function scan_dir($dirname)
  {
    // Открываем текущий каталог
    $dir = opendir($dirname);
    // Читаем в цикле каталог
    while (($file = readdir($dir)) !== false)
    {
      // Проверяем, не равно ли значение переменной
      // $file имени текущего или вышележащего каталога
      if($file != "." && $file != "..")
      {
        echo "$dirname/$file<br>";
        // Если перед нами каталог, вызываем рекурсивно
        // функцию scan_dir
        if(is_dir("$dirname/$file"))
        {
          scan_dir("$dirname/$file");
       }
     }
   }
   // Закрываем каталог
   closedir($dir);
  }

  // Имя каталога
  $dirname = ".";
```

```
  // Вызов функции, осуществляющей рекурсивный спуск по подкаталогам
  // корневого каталога
  scan_dir($dirname);
?>
```

При обходе дерева каталогов важно следить, чтобы в рекурсивный спуск не попадали текущий и родительский каталоги, иначе произойдет зацикливание.

4.16. Количество файлов в каталогах

Создадим скрипт, позволяющий выводить список подкаталогов в текущем каталоге и количество файлов в них. Для каждого подкаталога должен выводиться список его подкаталогов (рис. 4.6).

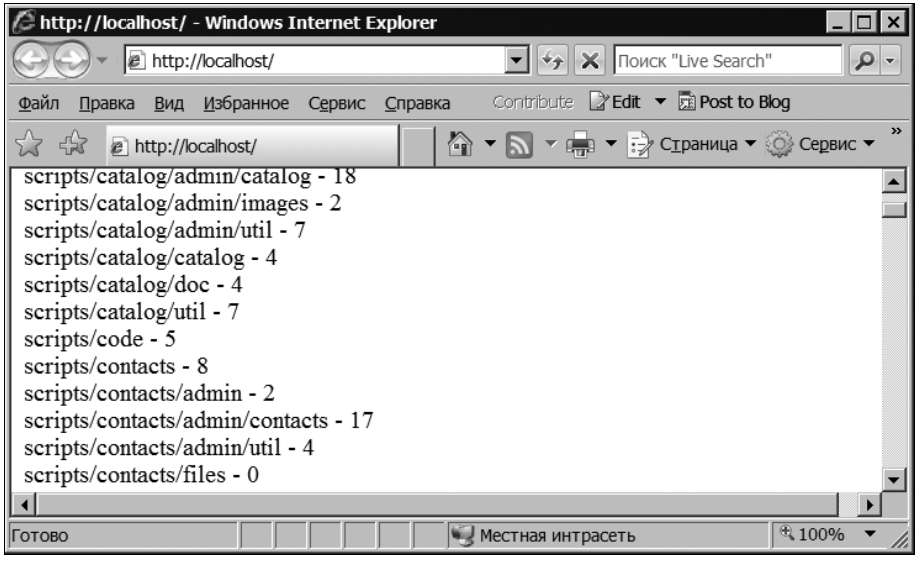


Рис. 4.6. Количество файлов в каталоге и подкаталогах

Для решения данной задачи также используется рекурсивный спуск. Только вывод информации производится лишь в том случае, если текущим элементом является каталог. Подсчет файлов в каталоге осуществляется при помощи специальной функции `file_count()` (листинг 4.47).

Листинг 4.47. Количество файлов в каталогах

```
<?php
  ////////////////////////////////////////////////////////
  // Рекурсивная функция — спускаемся вниз по каталогу
  ////////////////////////////////////////////////////////
  function scan_dir($dirname)
```

```
  {
    // Открываем текущий каталог
    $dir = opendir($dirname);
    // Читаем в цикле каталог
    while (($file = readdir($dir)) !== false)
    {
      // Проверяем, не равно ли значение переменной
      // $file имени текущего или вышележащего каталога
      if($file != "." && $file != "..")
      {
        // Если перед нами каталог, вызываем рекурсивно
        // функцию scan_dir
        if(is_dir("$dirname/$file"))
        {
          echo "$dirname/$file — ".
               file_count("$dirname/$file")."<br>";
          scan_dir("$dirname/$file");
       }
     }
   }
   // Закрываем каталог
   closedir($dir);
  }
  ////////////////////////////////////////////////////////
  // Функция, вычисляющая количество файлов в каталоге
  ////////////////////////////////////////////////////////
  function file_count($dirname)
  {
    // Переменная для подсчета
    $count = 0;
    // Открываем каталог
    $dir = opendir($dirname);
    // В цикле считываем его содержимое
    while(($file = readdir($dir)))
    {
      // Если текущий объект является файлом — считаем его
      if(is_file("$dirname/$file")) ++$count;
    }
    // Закрываем каталог
    closedir($dir);
    return $count;
  }

  // Имя каталога
  $dirname = "scripts";
  // Вызов функции, осуществляющей рекурсивный спуск по подкаталогам
  // корневого каталога
  scan_dir($dirname);
?>
```

4.17. Копирование содержимого одного каталога в другой

Для копирования содержимого одного каталога в другой вместе со всеми вложенными подкаталогами необходимо воспользоваться рекурсивным обходом каталога (листинг 4.48).

Листинг 4.48. Копирование содержимого одного каталога в другой

```
<?php
  // Копируем содержимое каталога home в home2
  lowering("home","home2");
  ////////////////////////////////////////////////////////
  // Рекурсивная функция спуска
  ////////////////////////////////////////////////////////
  function lowering($dirname,$dirdestination)
  {
    // Открываем каталог
    $dir = opendir($dirname);
    // В цикле выводим его содержимое
    while (($file = readdir($dir)) !== false)
    {
      echo $file."<br>";
      if(is_file("$dirname/$file"))
      {
        copy("$dirname/$file.", "$dirdestination/$file");
      }
      // Если это каталог — создаем его
      if(is_dir("$dirname/$file") &&
         $file != "." &&
         $file != "..")
      {
        // Создаем каталог
        if(!mkdir($dirdestination."/".$file))
        {
          echo "Невозможно создать каталог $dirdestination/$file";
        }
        // Вызываем рекурсивно функцию lowering
        lowering("$dirname/$file", "$dirdestination/$file");
      }
    }
    // Закрываем каталог
    closedir($dir);
  }
?>
```

4.18. Удаление каталога со всем содержимым

Если требуется удалить каталог, можно воспользоваться функцией `rmdir()` (листинг 4.49).

Листинг 4.49. Использование функции `rmdir()`

```
<?php
  if (rmdir("c:/temp/test")) echo("Каталог успешно удален");
  else echo("Ошибка удаления каталога");
?>
```

Однако если удаляемый каталог содержит хотя бы один файл или вложенный подкаталог, функция не удалит каталог и вернет `FALSE`. Для того чтобы удалить непустой каталог, необходимо рекурсивно спуститься по всем его подкаталогам и удалить все файлы. Для этого каждый из подкаталогов открывается при помощи функции `opendir()`, после чего его содержимое в цикле читается функцией `readdir()`. Пример такой функции демонстрируется в листинге 4.50.

Листинг 4.50. Рекурсивная функция для удаления каталога

```
<?php
  // Рекурсивная функция удаления каталога
  // с произвольной степенью вложенности
  function full_del_dir($directory)
  {
    // Открываем каталог
    $dir = opendir($directory);
    // В цикле выводим его содержимое
    while(($file = readdir($dir)) !== FALSE)
    {
      // Если функция readdir() вернула файл — удаляем его
      if (is_file("$directory/$file")) unlink("$directory/$file");
      // Если функция readdir() вернула каталог, и он
      // не равен текущему или родительскому — осуществляем
      // рекурсивный вызов full_del_dir() для этого каталога
      else if (is_dir("$directory/$file") &&
               $file != "." &&
               $file != "..")
      {
        full_del_dir("$directory/$file");
      }
    }
    closedir($dir);
    rmdir($directory);
```

```
    echo("Каталог успешно удален");
  }
  // Удаляем каталог temp
  full_del_dir("temp");
?>
```

4.19. Подсчет объема памяти, занимаемой каталогом

Как и в случае предыдущих задач, необходимо осуществить рекурсивный спуск по каталогу и при помощи стандартной функции `filesize()` подсчитать объем памяти, занимаемой всеми встречающимися файлами. Текущий объем каталога хранится в переменной `$size`, которая передается в качестве второго необязательного параметра рекурсивной функции `size_dir()` (листинг 4.51).

Листинг 4.51. Подсчет объема памяти, занимаемой каталогом

```
<?php
  // Рекурсивная функция удаления каталога
  // с произвольной степенью вложенности
  function size_dir($directory, $size = 0)
  {
    $dir = opendir($directory);
    while(($file = readdir($dir)))
    {
      // Если функция readdir() вернула файл — учитываем его размер
      if(is_file("$directory/$file"))
      {
        $size += filesize("$directory/$file");
      }
      // Если функция readdir() вернула каталог и он
      // не равен текущему или родительскому — осуществляем
      // рекурсивный вызов full_del_dir() для этого каталога
      else if (is_dir("$directory/$file") &&
               $file != "." &&
               $file != "..")
      {
        size_dir("$directory/$file", $size);
      }
    }
    closedir($dir);
    return $size;
  }

  // Подсчитываем объем текущего каталога
  echo size_dir(".");
?>
```

ГЛАВА 5

Сетевые возможности

Практически сразу после своего появления PHP приобрел широкое распространение в построении Web-приложений. Одним из главных факторов такой экспансии эксперты называют широкие сетевые возможности, встроенные в язык. В данной главе мы рассмотрим примеры использования PHP как инструмента для сетевого мониторинга.

5.1. Загрузка удаленного файла

Все файловые функции PHP способны открывать не только локальные файлы, но файлы, расположенные на удаленном хосте. В листинге 5.1 приводится пример скрипта, который загружает изображение с удаленного хоста **http://site.dev/image.jpg** и сохраняет его в локальной папке image/image.jpg.

Листинг 5.1. Загрузка удаленного файла (вариант 1)

```
<?php
  $content = file_get_contents("http://site.dev/image.jpg");
  file_put_contents("image/image.jpg", $contents);
?>
```

Как видно из листинга 5.1, содержимое файла при помощи функции `file_get_contents()` сохраняется в промежуточную переменную `$content`, после чего при помощи функции `file_put_contents()` — в локальном файле.

Задачу загрузки удаленного файла можно решить в одну строку, если воспользоваться функцией `copy()`, указав в качестве первого аргумента адрес удаленного файла, а в качестве второго аргумента локальный адрес (листинг 5.2).

ЗАМЕЧАНИЕ

При выполнении скриптов под управлением операционной UNIX-подобной операционной системы следует иметь в виду, что у скрипта должно быть достаточно прав доступа для создания файлов в каталоге image.

Листинг 5.2. Загрузка удаленного файла (вариант 2)

```
<?php
  copy("http://site.dev/image.jpg", "image.jpg");
?>
```

5.2. Что такое сокеты

Файловые функции не всегда удобны, т. к. удаленный хост может быть оптимизирован для работы с браузером, который должен поддерживать все особенности HTTP-протокола. В этом случае прибегают к сокетам. *Сокеты* — это файловый интерфейс к сетевым ресурсам (т. к. если бы это были обычные файлы). Файловые функции, сетевые возможности которых обсуждались в *разд. 5.1*, также используют сокеты при обращении к удаленным серверам.

> ***Замечание***
>
> На заре становления компьютерной индустрии файловый интерфейс очень интенсивно использовался для самых разных задач, начиная от доступа к принтеру и заканчивая преобразованием строки из одного формата в другой. Наследие тех времен очень хорошо видно в UNIX-подобных операционных системах, а также в библиотеках, которые реализуются в разных операционных системах на протяжении уже десятка лет.

Для открытия одиночного сокета используется функция `fsockopen()`, которая имеет следующий синтаксис:

```
fsockopen($target, $port [, $errno [, $errstr [, $timeout]]])
```

Первый параметр `$target` содержит адрес соединения, а второй `$port` — номер порта. В случае удачи функция возвращает дескриптор соединения, в противном случае — значение `FALSE`. Если при этом функции переданы два необязательных параметра `$errno` и `$errstr`, в них размещается код и текстовое описание ошибки соответственно. Пятый, также необязательный параметр `$timeout` задает значение тайм-аута, определяющего максимальное время ожидания ответа сервера в секундах. При успешной установке соединения функция возвращает дескриптор соединения, при неудаче — `FALSE`.

> ***Замечание***
>
> Для того чтобы функция `fsockopen()` была доступна из PHP-скрипта, в конфигурационном файле php.ini необходимо подключить расширение php_sockets.dll, сняв комментарий (`;`) с соответствующей директивы `extension`.

Пример работы с функцией `fsockopen()` приведен в листинге 5.3, где происходит загрузка главной страницы портала **http://www.php.net**.

Листинг 5.3. Пример использования функции `fsockopen()`

```
<?php
  function get_content($hostname, $path)
  {
    $line = "";
```

```
    // Устанавливаем соединение, имя которого
    // передано в параметре $hostname
    $fp = fsockopen($hostname, 80, $errno, $errstr, 30);
    // Проверяем успешность установки соединения
    if (!$fp) echo "$errstr ($errno)<br />\n";
    else
    {
      // Формируем HTTP-запрос для передачи его серверу
      $headers = "GET $path HTTP/1.1\r\n";
      $headers .= "Host: $hostname\r\n";
      $headers .= "Connection: Close\r\n\r\n";
      // Отправляем HTTP-запрос серверу
      fwrite($fp, $headers);
      // Получаем ответ
      while (!feof($fp))
      {
        $line .= fgets($fp, 1024);
      }
      fclose($fp);
    }
    return $line;
  }
  $hostname = "www.php.net";
  $path = "/";
  // Устанавливаем большее время работы скрипта.
  // Пока вся страница не загружена, она не будет отображена
  set_time_limit(180);
  // Вызываем функцию
  echo get_content($hostname, $path);
?>
```

При работе с сокетами скрипт вынужден брать на себя всю черновую работу по отправке серверу HTTP-заголовков, получению и обработке ответов на них. В приведенном скрипте обращение к `fsockopen()` оформлено в виде пользовательской функции `get_content()`, которая получает два параметра: `$hostname` — имя сервера, с которым устанавливается соединение, и `$path` — путь к странице относительно имени сервера.

> ***Замечание***
>
> Следует помнить, что при работе с функцией `fsockopen()` имя сервера не должно содержать префикса http://, указывающего на тип протокола и стандартный порт 80. При работе с сокетами реализация протокола ложится на плечи программиста, а порт указывается во втором параметре функции `fsockopen()`, поэтому такая детализация не требуется.

После установки соединения с сервером в переменной `$headers` формируется HTTP-запрос серверу методом GET. При подстановке значений всех переменных серверу будут отправлены соответствующие заголовки (листинг 5.4).

Листинг 5.4. Заголовки, отправляемые серверу www.php.net

```
GET / HTTP/1.1\r\n
Host: www.php.net\r\n
Connection: Close\r\n\r\n
```

После отправки запроса функцией `fgets()` производится чтение ответа из сокета блоками по 1 024 байта до тех пор, пока не встретится символ конца файла, определяемый функцией `feof()`. Затем результат возвращается и выводится в окно браузера.

В первой строке листинга 5.4 у сервера запрашивается индексный файл корневого каталога сервера (`/`) методом `GET` по протоколу `HTTP/1.1`. HTTP-заголовок `Host` является обязательным, он необходим HTTP-серверу для определения сайта, запрашиваемого клиентом (на тот случай, если к одному IP-адресу привязано несколько доменных имен). Последний HTTP-заголовок `Connection` требует от сервера закрыть соединение после передачи данных.

> ***Замечание***
>
> Вместо символа `/` можно передать адрес страницы на сервере, например, `/downloads.php`, в результате чего будет загружена другая страница сайта — именно так и действуют браузеры. При переходе пользователей по ссылке они извлекают часть адреса после доменного имени и передают HTTP-запрос, подставив эту часть после ключевого слова `GET`.

Эти три строки эквивалентны запросу в окне браузера **http://www.php.net/**. Результат работы скрипта из листинга 5.3 приведен на рис. 5.1.

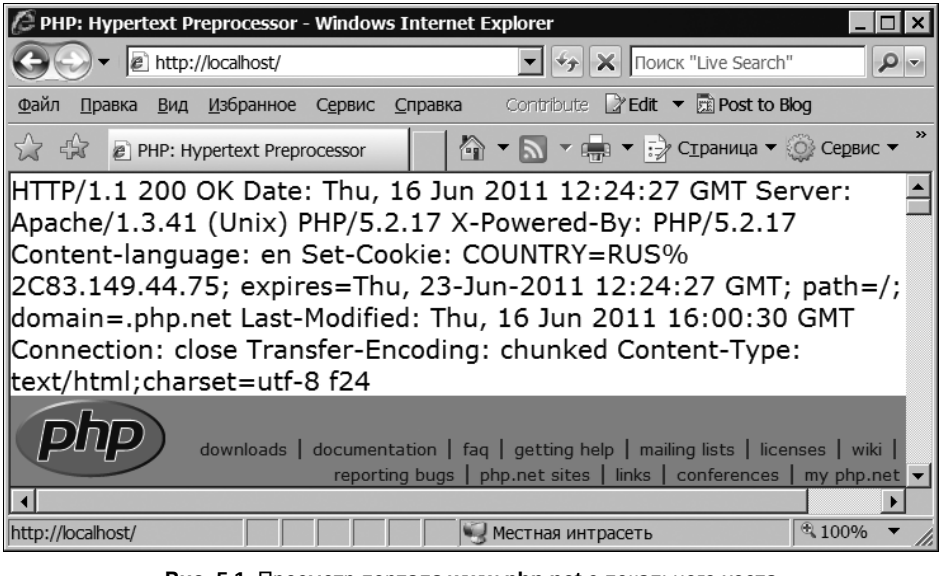


Рис. 5.1. Просмотр портала **www.php.net** с локального хоста

В начале страницы идут HTTP-заголовки, которые браузер обычно скрывает от пользователя, после чего начинается тело страницы, которое интерпретируется браузером. Избавиться от HTTP-заголовков можно при помощи функции `strstr()`, которая возвращает часть строки (первый параметр), начиная с первого вхождения подстроки (второй параметр) (листинг 5.5).

Листинг 5.5. Удаление HTTP-заголовков

```
<?php
  ...
  $hostname = "www.php.net";
  $path = "/";
  // Устанавливаем большее время работы скрипта.
  // Пока вся страница не будет загружена, она не будет отображена
  set_time_limit(180);
  // Вызываем функцию
  $content = get_content($hostname, $path);
  echo strstr($content, '<');
?>
```

Результат работы скрипта из листинга 5.5 представлен на рис. 5.2. Несмотря на то, что скрипт загружает страницу с портала **http://www.php.net**, адресная строка указывает на скрипт, расположенный на локальном хосте. Таким образом, в качестве браузера выступает PHP-скрипт.

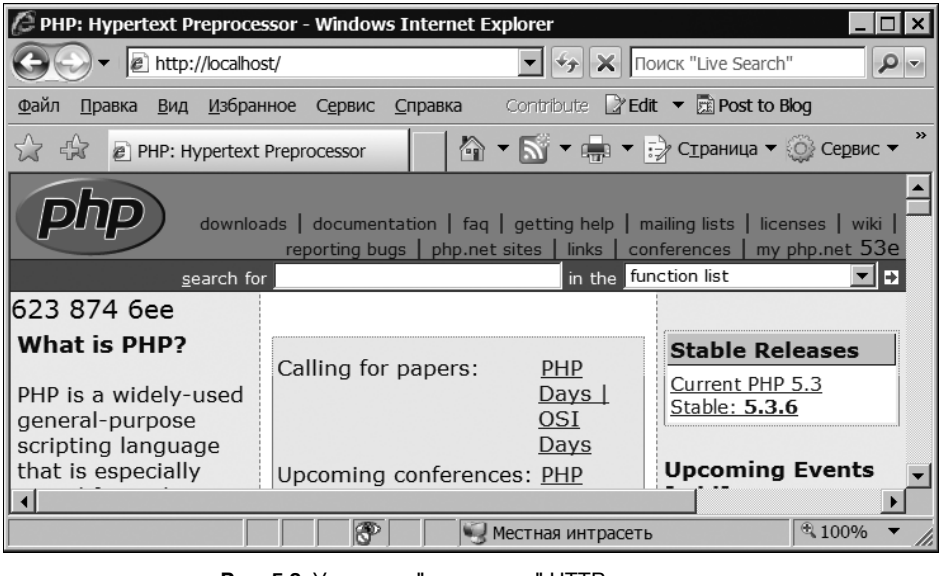


Рис. 5.2. Удаление "мешающих" HTTP-заголовков

5.3. Получение HTTP-заголовков

Если в *разд. 5.2* мы избавлялись от HTTP-заголовков, которые были получены при помощи сокетов, теперь встает обратная задача: получение HTTP-заголовков. Данная задача особенно актуальна при создании собственных интеллектуальных агентов, которые должны эмулировать работу одного из браузеров.

> ***ЗАМЕЧАНИЕ***
>
> Интеллектуальные агенты используются для содействия оператору или сбора информации, автоматизируя действия от имени пользователя.

Самым простым решением будет модификация функции из листинга 5.3 для получения лишь заголовков HTTP-ответа (листинг 5.6). HTTP-заголовки отделяются от тела HTTP-документа двойным переводом строк. Таким образом, при получении ответа достаточно обнаружить первую пустую строку, после чего прекратить прием данных с сервера.

Листинг 5.6. Загружаем только заголовки HTTP-ответа

```
<?php
  // Функция получения HTTP-заголовков
  function get_content($hostname, $path)
  {
    $line = "";
    // Устанавливаем соединение, имя которого
    // передано в параметре $hostname
    $fp = fsockopen($hostname, 80, $errno, $errstr, 30);
    // Проверяем успешность установки соединения
    if (!$fp) echo "$errstr ($errno)<br />\n";
    else
    {
      // Формируем HTTP-запрос серверу
      $headers = "GET $path HTTP/1.1\r\n";
      $headers .= "Host: $hostname\r\n";
      $headers .= "Connection: Close\r\n\r\n";
      // Отправляем HTTP-запрос серверу
      fwrite($fp, $headers);
      $end = $false;
      // Получаем ответ
      while (!$end)
      {
        $line = fgets($fp, 1024);
        if (trim($line) == "") $end = true;
        else $out[] = $line;
      }
      fclose($fp);
    }
```

```
    return $out;
  }
  $hostname = "www.php.net";
  $path = "/";
  // Устанавливаем большее время работы скрипта.
  // Пока страница не будет загружена целиком, она не будет отображена
  set_time_limit(180);
  // Вызываем функцию
  $out = get_content($hostname, $path);
  // Выводим содержимое массива
  echo "<pre>";
  print_r($out);
  echo "</pre>";
?>
```

Результат работы функции может выглядеть следующим образом:

```
Array
(
    [0] => HTTP/1.1 200 OK
    [1] => Date: Sun, 05 Jun 2011 07:19:00 GMT
    [2] => Server: Apache/1.3.41 (Unix) PHP/5.2.12RC4-dev
    [3] => X-Powered-By: PHP/5.2.12RC4-dev
    [4] => Content-language: en
    [5] => Set-Cookie: COUNTRY=RUS%2C83.149.47.252; expires=Sun, 12-Jun-2011
07:19:00 GMT; path=/; domain=.php.net
    [6] => Last-Modified: Sun, 05 Jun 2011 11:00:27 GMT
    [7] => Connection: close
    [8] => Transfer-Encoding: chunked
    [9] => Content-Type: text/html;charset=utf-8
)
```

Первая строка является стандартным ответом сервера о том, что запрос успешно обработан (код ответа 200). Если запрашиваемый ресурс не существует, то будет возвращен код ответа 404 (HTTP/1.1 404 Not Found).

Второй заголовок `Date` сообщает время формирования документа на сервере.

Третий заголовок `Server` сообщает тип и версию Web-сервера. Как можно видеть, портал **http://www.php.net** работает на сервере под управлением UNIX, используя в качестве Web-сервера Apache 1.3.41, а также PHP версии 5.2.12.

Четвертый заголовок `X-Powered-By` сообщает, что страница сгенерирована при помощи PHP версии 5.2.12.

> ***Замечание***
>
> HTTP-заголовки, начинающиеся с префикса `X-`, не являются стандартными, в них, как правило, серверы и клиенты передают дополнительную информацию от модулей сервера, антивируса, систем антиспама и т. п.

Пятый заголовок `Last-Modified` сообщает о дате последней модификации страницы; впрочем, при динамическом формировании содержимого сайта он не актуален.

Шестой заголовок `Content-language` сообщает, что браузеру передаются данные на английском языке.

Седьмой заголовок `Set-Cookie` устанавливает cookie с именем `COUNTRY` и значением `"RUS,83.149.47.252"`, содержащим страну пользователя и его IP-адрес сроком на неделю для домена `.php.net`. Данная информация необходима для раздела **downloads** сайта **http://www.php.net**, в котором посетителю предлагается ближайший к нему сервер.

Восьмой заголовок `Connection` передает клиенту просьбу закрыть соединение после получения ответа.

Девятый заголовок `Transfer-Encoding` (имеющий значение `chuncked`) указывает получателю, что ответ разбит на фрагменты.

Десятый заголовок `Content-Type` указывает тип загружаемого документа (`text/html`) и его кодировку (`charset=utf-8`).

В листинге 5.6 для получения заголовков использован запрос GET, однако для этой цели в протоколе HTTP предусмотрен специальный метод HEAD. Таким образом, скрипт из листинга 5.6 можно переписать более простым способом (листинг 5.7).

Листинг 5.7. Использование метода HEAD

```
<?php
  // Функция получения HTTP-заголовков
  function get_content($hostname, $path)
  {
    $line = "";
    // Устанавливаем соединение, имя которого
    // передано в параметре $hostname
    $fp = fsockopen($hostname, 80, $errno, $errstr, 30);
    // Проверяем успешность установки соединения
    if (!$fp) echo "$errstr ($errno)<br />\n";
    else
    {
      // Формируем HTTP-запрос серверу
      $headers = "HEAD $path HTTP/1.1\r\n";
      $headers .= "Host: $hostname\r\n";
      $headers .= "Connection: Close\r\n\r\n";
      // Отправляем HTTP-запрос серверу
      fwrite($fp, $headers);
      // Получаем ответ
      while (!feof($fp))
      {
        $out[] = fgets($fp, 1024);
      }
      fclose($fp);
    }
```

```
    return $out;
  }
  $hostname = "www.php.net";
  $path = "/";
  // Устанавливаем большее время работы скрипта.
  // Пока страница не будет загружена целиком, она не отобразится
  set_time_limit(180);
  // Вызываем функцию
  $out = get_content($hostname, $path);
  // Выводим содержимое массива
  echo "<pre>";
  print_r($out);
  echo "</pre>";
?>
```

Следует отметить, что результат работы листинга 5.6 будет отличаться от листинга 5.7 отсутствием HTTP-заголовка `Transfer-Encoding`, т. к. для пересылки одних лишь HTTP-заголовков не требуется разбивать документ на части.

> ***Замечание***
>
> Метод HEAD может быть запрещен на HTTP-сервере, поэтому иногда целесообразно воспользоваться методом GET, отсекая тело документа.

Следует отметить, что в PHP предусмотрена специальная функция `get_headers()`, которая позволяет получить HTTP-заголовки удаленного ресурса, которая также использует для получения HTTP-заголовков метод HEAD. Функция принимает второй необязательный параметр, если он принимает значение `true`, то вместо индексного массива возвращается ассоциативный массив, в качестве ключей которого выступают названия заголовков (листинг 5.8).

Листинг 5.8. Получение HTTP-заголовков функцией `get_headers()`

```
<?php
  $out = get_headers("http://www.php.net", true);
  echo "<pre>";
  print_r($out);
  echo "</pre>";
?>
```

Результат работы скрипта из листинга 5.8 может выглядеть следующим образом:

```
Array
(
    [0] => HTTP/1.1 200 OK
    [Date] => Sun, 05 Jun 2011 08:06:52 GMT
    [Server] => Apache/1.3.41 (Unix) PHP/5.2.12RC4-dev
    [X-Powered-By] => PHP/5.2.12RC4-dev
    [Content-language] => en
```

```
    [Set-Cookie] => COUNTRY=RUS%2C83.149.47.252; expires=Sun, 12-Jun-2011 08:06:52
                                                        GMT; path=/; domain=.php.net
    [Last-Modified] => Sun, 05 Jun 2011 11:00:27 GMT
    [Connection] => close
    [Content-Type] => text/html;charset=utf-8
)
```

5.4. Определение размера файла на удаленном хосте

Одной из распространенных задач является определение размера файла на удаленном хосте. Для этого воспользуемся функцией `get_content()`, с помощью которой узнаем количество байтов в файле по HTTP-заголовку `Content-Length` (листинг 5.9).

Листинг 5.9. Определение размера файла на удаленном хосте

```
<?php
  // Получаем HTTP-заголовки
  $hostname = "http://www.softtime.ru/files/configs.zip";
  $out = get_headers($hostname, true);
  foreach($out as $header => $value)
  {
    if($header === "Content-Length")
    {
      echo "Количество байтов в архиве — $value";
    }
  }
?>
```

Результат работы функции:

```
Количество байтов в архиве — 26421
```

5.5. Библиотека CURL

Помимо сокетов, обеспечивающих низкоуровневое обращение к серверу, PHP располагает специальным расширением CURL (Client URL Library). В листинге 5.10 представлен скрипт, осуществляющий загрузку удаленной страницы средствами библиотеки CURL.

> ***Замечание***
>
> Для того чтобы функции библиотеки CURL были доступны из PHP-скрипта, в конфигурационном файле php.ini необходимо подключить расширение php_curl.dll, сняв комментарий (точка с запятой `;`) с директивы `extension`. Помимо этого необходимо скопировать библиотеки ssleay32.dll и libeay32.dll из каталога, где расположен PHP, в папку, прописанную в переменной окружения `PATH`, например, в C:\Windows\system32.

В случае расширения CURL не требуется удаление HTTP-заголовков, возвращаемых сервером, т. к. библиотека их удаляет по умолчанию. Однако CURL можно настроить на выдачу HTTP-заголовков, передаваемых сервером, если установить

при помощи функции `curl_setopt()` ненулевое значение параметра `CURLOPT_HEADER` (см. табл. 5.1).

Листинг 5.10. Загрузка страницы с использованием расширения CURL

```
<?php
  // Задаем адрес удаленного сервера
  $curl = curl_init("http://www.php.net");
  // Устанавливаем параметры соединения
  curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
  // Получаем содержимое страницы
  $content = curl_exec($curl);
  // Закрываем CURL-соединение
  curl_close($curl);
  // Выводим содержимое страницы
  echo $content;
?>
```

При помощи функции `curl_init()` задается адрес удаленного сервера и путь к файлу на нем. В отличие от функции `fsockopen()`, необходимо задавать адрес полностью, включая префикс http://, т. к. расширение CURL позволяет работать с несколькими видами протоколов (HTTP, HTTPS, FTP). Если соединение с указанным сервером происходит успешно, функция `curl_init()` возвращает дескриптор соединения, который используется в качестве параметра во всех остальных функциях библиотеки.

Функция `curl_setopt()` позволяет задать параметры текущего соединения и имеет следующий синтаксис:

```
curl_setopt($curl, $option, $value)
```

Функция устанавливает для соединения с дескриптором `$curl` параметр `$option` со значением `$value`. В качестве параметра `$option` используются константы, определенные в табл. 5.1–5.5. Если параметр успешно установлен, функция возвращает `TRUE`.

Запрос выполняет функция `curl_exec()`. В листинге 5.10 содержимое запрашиваемой страницы возвращается в виде строки `$content` (такое поведение определяется константой `CURLOPT_RETURNTRANSFER`, установленной ранее при помощи функции `curl_setopt()`).

В завершение функция `curl_exec()` закрывает установленное ранее CURL-соединение.

Таблица 5.1. *Логические параметры CURL-соединения*

Параметр	Описание
`CURLOPT_AUTOREFERER`	Если осуществляется следование HTTP-заголовку `Location`, HTTP-заголовок `Referer` устанавливается автоматически

Таблица 5.1 *(продолжение)*

Параметр	Описание
CURLOPT_COOKIESESSION	CURL игнорирует все сессионные cookie (хранящиеся в оперативной памяти и лишь до момента закрытия браузера)
CURLOPT_CRLF	UNIX-переводы строк \n автоматически преобразуются к виду \r\n
CURLOPT_DNS_USE_GLOBAL_CACHE	Используется глобальный DNS-кэш
CURLOPT_FAILONERROR	Получение HTTP-кода более 300 считается ошибкой
CURLOPT_FILETIME	CURL пытается получить дату последней модификации загружаемого документа
CURLOPT_FOLLOWLOCATION	Перенаправление на указанный URL (это действие выполняется рекурсивно для каждого полученного заголовка Location)
CURLOPT_FORBID_REUSE	CURL-соединение завершается сразу после выполнения функции curl_exec(); повторное использования дескриптора соединения не допускается
CURLOPT_FRESH_CONNECT	Для повторной операции curl_exec() CURL требует создания нового соединения, не используя кэшированное
CURLOPT_FTPAPPEND	Данные будут добавляться к файлу на FTP сервере, в противном случае файл будет перезаписан
CURLOPT_FTPASCII	Данные с FTP-сервера будут передаваться не в бинарном, а в текстовом (ASCII) виде
CURLOPT_FTPLISTONLY	При установке этого параметра в TRUE будет получен список файлов в текущем каталоге FTP-сервера
CURLOPT_HEADER	Результат будет включать полученные HTTP-заголовки
CURLOPT_HTTPPROXYTUNNEL	При установке этого параметра в TRUE данные будут передаваться через прокси-сервер
CURLOPT_MUTE	При установке этого параметра в TRUE все сообщения CURL будут подавляться
CURLOPT_NETRC	При установке этого параметра в TRUE будет осуществлена попытка найти имя пользователя и пароль к удаленному серверу в файле ~/netrc (где ~ — домашний каталог)
CURLOPT_NOBODY	Результат не будет включать документ. Часто используется для того, чтобы получить только HTTP-заголовки
CURLOPT_NOPROGRESS	При установке этого параметра в TRUE не будет выводиться индикатор прогресса операции (по умолчанию установлен)
CURLOPT_POST	Отправляется POST-запрос типа application/x-www-form-urlencoded
CURLOPT_PUT	При установке этого параметра в TRUE будет производиться закачка файла методом PUT протокола HTTP. Файл задается параметрами CURLOPT_INFILE и CURLOPT_INFILESIZE. Впрочем, метод PUT на большинстве серверов запрещен к использованию
CURLOPT_RETURNTRANSFER	При установке этого параметра в TRUE CURL будет возвращать результат, а не выводить его

Таблица 5.1 *(окончание)*

Параметр	Описание
CURLOPT_SSL_VERIFYPEER	При установке параметра в FALSE запрещается проверка сертификата удаленного сервера. Дополнительные сертификаты можно задать с помощью параметра CURLOPT_CAINFO. Можно также указать путь к файлам сертификатов в параметре CURLOPT_CAPATH. Если CURLOPT_SSL_VERIFYPEER установлен в 0, возможно, потребуется установить в 1 или 0 также CURLOPT_SSL_VERIFYHOST (по умолчанию 2)
CURLOPT_UNRESTRICTED_AUTH	При установке параметра в TRUE имя пользователя и пароль сохраняются и передаются серверу при следовании HTTP-заголовку Location, даже если осуществляется переход на другой сайт
CURLOPT_UPLOAD	При установке этого параметра в TRUE производится закачка файла на удаленный сервер
CURLOPT_VERBOSE	При установке этого параметра в TRUE выводятся подробные сообщения обо всех производимых действиях

Таблица 5.2. *Целочисленные параметры CURL-соединения*

Параметр	Описание
CURLOPT_BUFFERSIZE	Размер буфера, используемого для чтения документа с удаленного сервера
CURLOPT_CONNECTTIMEOUT	Количество секунд перед попыткой соединения с сервером. По умолчанию имеет значение 0
CURLOPT_DNS_CACHE_TIMEOUT	Количество секунд, в течение которых хранится запись в DNS-кэше. По умолчанию имеет значение 120 (2 минуты)
CURLOPT_HTTP_VERSION	Версия HTTP-протокола; допустимы три значения: CURL_HTTP_VERSION_NONE *(*версия выбирается автоматически*)*, CURL_HTTP_VERSION_1_0 *(*используется HTTP *1.0)*, CURL_HTTP_VERSION_1_1 *(*используется HTTP *1.1)*
CURLOPT_HTTPAUTH	Метод (методы) HTTP-аутентификации; допустимые значения: CURLAUTH_BASIC, CURLAUTH_DIGEST, CURLAUTH_GSSNEGOTIATE, CURLAUTH_NTLM, CURLAUTH_ANY и CURLAUTH_ANYSAFE
CURLOPT_INFILESIZE	Размер файла при его загрузке на удаленный сервер
CURLOPT_LOW_SPEED_LIMIT	Минимальная скорость передачи в байтах в секунду. Если в течение периода времени, заданного параметром CURLOPT_LOW_SPEED_TIME, скорость передачи станет меньше этого значения, операция будет прервана
CURLOPT_LOW_SPEED_TIME	Время в секундах, в течение которого скорость передачи должна быть не ниже чем CURLOPT_LOW_SPEED_LIMIT, чтобы операция была признана слишком медленной и прервана
CURLOPT_MAXCONNECTS	Максимальное количество постоянных соединений
CURLOPT_MAXREDIRS	Максимальное количество переходов по заголовку Location. Параметр используется совместно с CURLOPT_FOLLOWLOCATION

Таблица 5.2 *(окончание)*

Параметр	Описание
CURLOPT_PORT	Альтернативный номер порта для соединения
CURLOPT_PROXYAUTH	Метод (методы) аутентификации на прокси-сервере; принимает те же значения, что и параметр CURLOPT_HTTPAUTH
CURLOPT_PROXYPORT	Номер порта для соединения с прокси-сервером; используется совместно с CURLOPT_PROXY
CURLOPT_PROXYTYPE	Тип прокси-сервера: устанавливать соединение через HTTP-протокол (CURLPROXY_HTTP) или через сокет (CURLPROXY_SOCKS5)
CURLOPT_RESUME_FROM	Задает в файле позицию байта, с которого начнется передача данных
CURLOPT_SSL_VERIFYHOST	Строгость проверки имени, указанного в сертификате удаленного сервера (при установке SSL-соединения): 1 — предполагает только проверку существования имени; 2 — кроме того, и проверку соответствия имени хоста
CURLOPT_SSLVERSION	Версия SSL; допустимые значения 2 и 3. По умолчанию версия SSL определяется автоматически, но в некоторых случаях требуется явное указание
CURLOPT_TIMECONDITION	Способ интерпретации значения параметра CURLOPT_TIMEVALUE. Допустимые значения: TIMECOND_IFMODSINCE и TIMECOND_ISUNMODSINCE. Применяется только для протокола HTTP
CURLOPT_TIMEOUT	Максимальное время выполнения операции в секундах
CURLOPT_TIMEVALUE	Время в секундах, прошедшее с полуночи 1 января 1970 г. Значение будет использовано в соответствии со значением параметра CURLOPT_TIMECONDITION (по умолчанию TIMECOND_IFMODSINCE)

Таблица 5.3. *Строковые параметры CURL-соединения*

Параметр	Описание
CURLOPT_CAINFO	Имя файла, содержащего один или более сертификатов, которые будут использованы при проверке подлинности удаленного сервера. Имеет значение только совместно с параметром CURLOPT_SSL_VERIFYPEER
CURLOPT_CAPATH	Путь к каталогу с сертификатами. Имеет значение только совместно с параметром CURLOPT_SSL_VERIFYPEER
CURLOPT_COOKIE	Содержимое HTTP-заголовка Cookie. Для установки нескольких значений cookie можно использовать несколько вызовов функции curl_setopt()
CURLOPT_COOKIEFILE	Имя файла, содержащего данные cookie. Данные могут быть либо в формате Netscape, либо HTTP-заголовков
CURLOPT_COOKIEJAR	Имя файла, в который сохраняются несессионные cookie, доступные при следующем сеансе соединения с сервером

Таблица 5.3 *(продолжение)*

Параметр	Описание
CURLOPT_CUSTOMREQUEST	Метод, который будет использован в HTTP-запросе вместо GET или HEAD. Используется для отправки запросов DELETE или других, редко используемых. Допустимыми значениями являются GET, POST и т. д.
CURLOPT_ENCODING	Значение HTTP-заголовка Accept-Encoding, задающего метод сжатия HTTP-ответа. Допустимые значения: identity, deflate, gzip
CURLOPT_FTPPORT	Значение IP-адреса для команды PORT протокола FTP. Допустимые значения: IP-адрес, имя хоста, имя сетевого интерфейса (под UNIX), или просто — для использования IP-адреса по умолчанию
CURLOPT_INTERFACE	Имя используемого сетевого интерфейса. Может быть именем интерфейса, IP-адресом или именем хоста
CURLOPT_KRB4LEVEL	Уровень безопасности KRB4 (Kerberos 4) для протокола FTP. Допустимы следующие значения (в порядке возрастания безопасности): clear, safe, confidential, private. Если переданное значение не входит в этот список, используется private. Установка параметра в NULL запрещает безопасность KRB4
CURLOPT_POSTFIELDS	Строка данных для POST-запроса
CURLOPT_PROXY	Имя прокси-сервера, через который будут направляться запросы
CURLOPT_PROXYUSERPWD	Строка с именем пользователя и паролем к прокси-серверу HTTP в виде [username]:[password]
CURLOPT_RANDOM_FILE	Путь к файлу, используемому генератором случайных чисел для работы протокола SSL
CURLOPT_RANGE	Координаты фрагмента загружаемого файла в формате "X-Y" (вместо X и Y указываются позиции байта в файле). Одна из координат может быть опущена, например: "X-". Протокол HTTP также поддерживает передачу нескольких фрагментов файла, это задается в виде "X-Y,N-M"
CURLOPT_REFERER	Значение HTTP-заголовка Referer
CURLOPT_SSL_CIPHER_LIST	Список шифров, используемых SSL, например: RC4-SHA, TLSv1
CURLOPT_SSLCERT	Имя файла с SSL-сертификатом в формате PEM
CURLOPT_SSLCERTPASSWD	Пароль к файлу SSL-сертификата, заданному параметром CURLOPT_SSLCERT
CURLOPT_SSLCERTTYPE	Формат сертификата. Допускаются следующие форматы: PEM (по умолчанию), DER и ENG
CURLOPT_SSLKEY	Имя файла, содержащего закрытый SSL-ключ
CURLOPT_SSLKEYPASSWD	Секретный пароль, необходимый для использования закрытого SSL-ключа
CURLOPT_SSLKEYTYPE	Формат закрытого SSL-ключа. Допускаются следующие форматы: PEM (по умолчанию), DER и ENG
CURLOPT_URL	URL, с которым будет производиться операция. Значение этого параметра также может быть задано при вызове функции curl_init()

Таблица 5.3 *(окончание)*

Параметр	Описание
CURLOPT_USERAGENT	Задает значение HTTP-заголовка User-Agent.
CURLOPT_USERPWD	Строка с именем пользователя и паролем в виде [username]:[password]

Таблица 5.4. *Параметры CURL-соединения, являющиеся массивами*

Параметр	Описание
CURLOPT_HTTP200ALIASES	Массив с HTTP-заголовками группы 200 (успешно выполненный запрос)
CURLOPT_HTTPHEADER	Массив со всеми HTTP-заголовками
CURLOPT_POSTQUOTE	Массив с FTP-командами, которые будут выполнены после выполнения основного запроса
CURLOPT_QUOTE	Массив с FTP-командами, которые будут выполнены перед выполнением основного запроса

Таблица 5.5. *Параметры CURL-соединения, использующие в качестве значений дескриптор файла, возвращаемого функцией* `fopen()`

Параметр	Данные, хранимые в файле
CURLOPT_FILE	Результат операции
CURLOPT_INFILE	Данные для передачи
CURLOPT_WRITEHEADER	Полученные HTTP-заголовки
CURLOPT_STDERR	Сообщения об ошибках

Если не задан ни один из параметров CURLOPT_RETURNTRANSFER, CURLOPT_FILE или CURLOPT_WRITEHEADER, функция curl_exec() по умолчанию выводит результат непосредственно в окно браузера. Так как результат запроса необходимо вывести непосредственно в браузер, листинг 5.10 можно переписать более компактно, не устанавливая никаких параметров (листинг 5.11).

Листинг 5.11. Альтернативное решение

```
<?php
  // Задаем адрес удаленного сервера
  $curl = curl_init("http://www.php.net");
  // Получаем содержимое страницы
  echo curl_exec($curl);
  // Закрываем CURL-соединение
  curl_close($curl);
?>
```

Так задачу по извлечению HTTP-заголовков при помощи библиотеки CURL можно решить при помощи скрипта, представленного в листинге 5.12.

Листинг 5.12. Использование CURL

```
<?php
  function get_content($hostname)
  {
    // Задаем адрес удаленного сервера
    $curl = curl_init($hostname);

    // Вернуть результат в виде строки
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
    // Включить в результат HTTP-заголовки
    curl_setopt($curl, CURLOPT_HEADER, 1);
    // Исключить тело HTTP-документа
    curl_setopt($curl, CURLOPT_NOBODY, 1);

    // Получаем HTTP-заголовки
    $content = curl_exec($curl);
    // Закрываем CURL-соединение
    curl_close($curl);

    // Преобразуем строку $content в массив
    return explode("\r\n", $content);
  }

  $hostname = "http://www.php.net";
  $out = get_content($hostname);

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

Перед отправкой запросов при помощи функции `curl_setopt()` устанавливаются параметры `CURLOPT_HEADER` и `CURLOPT_NOBODY`, первый из которых требует включения в результат HTTP-заголовков, а второй — игнорирования тела HTTP-документа.

5.6. Получение точного времени

На большинстве современных серверов проблема точного времени не стоит слишком остро: специальные средства отслеживают и корректируют время, причем делают это постепенно, чтобы предотвратить скачки в службах мониторинга (если последние используются на сервере). Тем не менее, в случае отсутствия таких средств или по другим причинам, получить точное время можно, обратившись

к одному из серверов точного времени по 13 порту (сервер отправляет точное время постоянно). В листинге 5.13 приводится пример обращения к серверу точного времени **wwv.nist.gov**.

Листинг 5.13. Получение ответа сервера точного времени

```
<?php
  $fp = fsockopen("wwv.nist.gov",
                  13,
                  $errno,
                  $errstr,
                  5);
  if (!$fp) exit("ERROR: $errno – $errstr");
  else echo fread($fp, 26);
  fclose($fp);
?>
```

Результат работы скрипта из листинга 5.13 может выглядеть следующим образом:

```
55717 11-06-05 10:17:45 5
```

5.7. Извлечение ссылок Yandex

Для того чтобы получить содержимое страницы поисковой системы Yandex, достаточно передать GET-параметр `text` с запросом обработчику **http://www.yandex.ru/yandsearch**. Например, для поиска фразы "Форум PHP" можно сформировать следующий запрос (чтобы закодировать русские символы достаточно пропустить текст запроса через функцию `urlendode()`):

```
http://www.yandex.ru/yandsearch?text=%D4%EE%F0%F3%EC+PHP
```

В листинге 5.14 приводится пример скрипта, который формирует запрос к поисковой системе Yandex и возвращает результаты в виде двумерного массива. Первое измерение содержит номер позиции на странице, второе измерение содержит два ассоциативных ключа: `url` для ссылки и `name` для названия страницы.

Листинг 5.14. Извлечение массива ссылок и текстов (Yandex)

```
<?php
  // Извлечение ссылок со страниц поисковой системы Yandex
  function yandex($url)
  {
    // Результирующий массив
    $result = array();
    // Загружаем содержимое страницы
    $contents = file_get_contents($url);
    // Регулярное выражение
    $pattern =
 "|<a.*?tabindex=\"[\d]+\"(.*?)href=\"([^\"]+)\"[^>]+>(.*?)</a>|is";
```

```
    // Выполняем поиск по регулярному выражению
    preg_match_all($pattern, $contents, $out, PREG_PATTERN_ORDER);
    // Помещаем результаты в результирующий массив
    for($i = 0; $i < count($out[2]); $i++)
    {
      $result[$i]['url'] = $out[2][$i];
      $result[$i]['name'] = $out[3][$i];
    }
    return $result;
  }
  // Поисковая страница
  $url = "http://www.yandex.ru/yandsearch?text=%D4%EE%F0%F3%EC+PHP";
  // Загружаем содержимое страницы
  $arr = yandex($url);
  echo "<pre>";
  print_r($arr);
  echo "<pre>";
?>
```

Для поиска ссылок на странице используется регулярное выражение, которое ищет ссылки, имеющие в своем составе атрибут `tabindex`.

5.8. Извлечение ссылок Google

Для того чтобы получить содержимое страницы поисковой системы Yandex, достаточно передать GET-параметр `q` с запросом обработчику **http://www.google.ru/search**. Нашей фразе "Форум PHP" соответствует запрос:

```
http://www.google.ru/search?q=%D4%EE%F0%F3%EC+PHP&complete=1&hl=ru&sa=N
```

В листинге 5.15 приводится пример скрипта, который формирует запрос к поисковой системе Google и возвращает результаты в виде двумерного массива: номер позиции на странице и два ассоциативных ключа (`url` для ссылки и `name` для названия страницы).

Листинг 5.15. Извлечение массива ссылок и текстов (Google)

```
<?php
  // Извлечение ссылок со страниц поисковой системы Google
  function google($url)
  {
    // Результирующий массив
    $result = array();
    // Загружаем содержимое страницы
    $contents = file_get_contents($url);
    // Регулярное выражение
    $pattern =
         '|<h3 class="r"><a href=\"([^\"]+)\"[^>]*>(.+)</a>|isU';
```

```
    // Выполняем поиск по регулярному выражению
    preg_match_all($pattern, $contents, $out, PREG_PATTERN_ORDER);

    // Помещаем результаты в результирующий массив
    for($i = 0; $i < count($out[1]); $i++)
    {
      $result[$i]['url'] = $out[1][$i];
      $result[$i]['name'] = $out[2][$i];
    }
    return $result;
  }
  // Поисковая страница
  $url = "http://www.google.ru/search?q=".urlencode("Форум PHP").
         "&complete=1&hl=ru&sa=N";
  // Загружаем содержимое страницы
  $arr = google($url);
  echo "<pre>";
  print_r($arr);
  echo "<pre>";
?>
```

5.9. Курс валют Центрального банка РФ

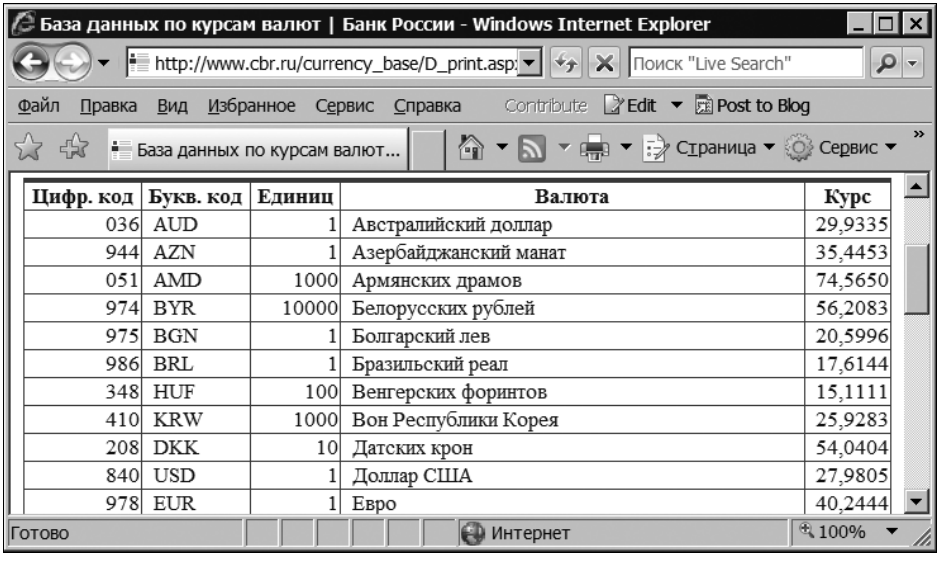
Помимо обычных HTML-страниц, в последнее время в Интернете получил распространение RSS-формат. Новости, каталоги продукции, сообщения форума представляют в виде XML-файла, который легко загрузить и разобрать, т. к. формат XML-файла не подвержен изменению, в отличие от HTML-страницы. Многие владельцы сайтов выкладывают информацию в XML-формате, позволяя владельцам других сайтов и поисковым роботам получать информацию в более удобном виде, чем HTML-страница.

Источником информации об официальном курсе валюты служит сайт Центробанка Российской Федерации. Обратившись по адресу сайта Центробанка **http://www.cbr.ru/currency_base/D_print.asp?date_req=$date**, где $date — дата в формате ДД/ММ/ГГГГ, можно узнать курс валют в запрошенный день: **http://www.cbr.ru/currency_base/D_print.asp?date_req=01/06/2011**.

> ***Замечание***
>
> Адреса скриптов могут меняться, за уточнением следует обращаться к странице **http://www.cbr.ru/scripts/Root.asp?Prtid=SXML**.

Помимо ссылок на HTML-страницу, можно обратиться к XML-представлению по адресу **http://www.cbr.ru/scripts/XML_daily.asp?date_req=$date**.

В XML-файле, который загружается с сайта Центрального банка России (рис. 5.3), каждая валюта описывается набором тегов, представленным в листинге 5.16.



Цифр. код	Букв. код	Единиц	Валюта	Курс
036	AUD	1	Австралийский доллар	29,9335
944	AZN	1	Азербайджанский манат	35,4453
051	AMD	1000	Армянских драмов	74,5650
974	BYR	10000	Белорусских рублей	56,2083
975	BGN	1	Болгарский лев	20,5996
986	BRL	1	Бразильский реал	17,6144
348	HUF	100	Венгерских форинтов	15,1111
410	KRW	1000	Вон Республики Корея	25,9283
208	DKK	10	Датских крон	54,0404
840	USD	1	Доллар США	27,9805
978	EUR	1	Евро	40,2444

Рис. 5.3. Курс валют на 1 июня 2011 г.

Листинг 5.16. Фрагмент XML-файла с курсом валюты

```
<Valute ID="R01010">
    <NumCode>036</NumCode>
    <CharCode>AUD</CharCode>
    <Nominal>1</Nominal>
    <Name>Австралийский доллар</Name>
    <Value>21,4176</Value>
</Valute>
```

Теги `<NumCode>` и `<CharCode>` характеризуют числовой и символьные коды валюты. В теге `<Name>` приводится полное название валюты. Значения тегов `<Nominal>` и `<Value>` дает курс валют.

ЗАМЕЧАНИЕ

Тег `<Nominal>`, как правило, принимает значение, кратное 10.

В первую очередь решим общую задачу разбора XML-файла при помощи регулярных выражений (листинг 5.17).

Листинг 5.17. Разбор XML-файла

```
<?php
  // Ссылка на XML-файл
  $url = "http://www.cbr.ru/scripts/XML_daily.asp?date_req=".
         date("d/m/Y");
  // Загружаем файл
  $content = file_get_contents($url);
```

```
  // Регулярное выражение
  $pattern = "|<valute
id=\"([^\"]+)\">[\s]*<NumCode>([^<]+)</NumCode>[\s]*<CharCode>([^<]+)</CharCode
>[\s]*<Nominal>([^<]+)</Nominal>[\s]*<Name>([^<]+)</Name>[\s]*<Value>([^<]+)</V
alue>[\s]*</Valute>|is";
  preg_match_all($pattern, $content, $out);
  echo "<table border=1>";
  echo "<tr>
          <td>ID</td>
          <td>Числовой код</td>
          <td>Символьный код</td>
          <td>Номинал</td>
          <td>Название</td>
          <td>Курс</td>
        </tr>";
  for($i = 0; $i < count($out[1]); $i++)
  {
    echo "<tr>
            <td>".$out[1][$i]."</td>
            <td>".$out[2][$i]."</td>
            <td>".$out[3][$i]."</td>
            <td>".$out[4][$i]."</td>
            <td>".$out[5][$i]."</td>
            <td>".$out[6][$i]."</td>
          </tr>";
  }
  echo "</table>";
?>
```

Данный скрипт последовательно разбирает содержимое XML-файла, помещая информацию о валютах в многомерный массив `$out`. Результат работы скрипта может выглядеть так, как это представлено на рис. 5.4.

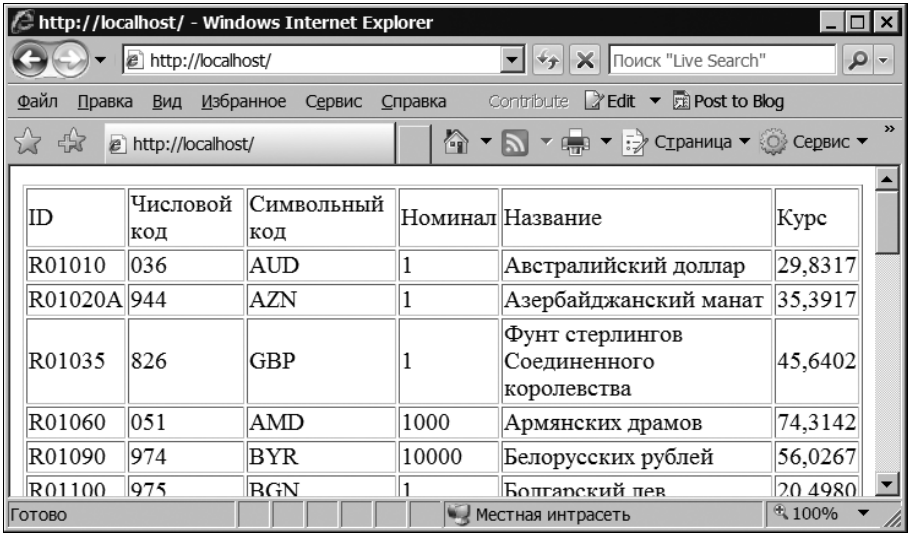
В листинге 5.18 представлен скрипт, который из всей массы доступной информации извлекает только сведения, относящиеся к доллару США и евро.

Листинг 5.18. Загрузка курса доллара и евро

```
<?php
  // Ссылка на XML-файл
  $url = "http://www.cbr.ru/scripts/XML_daily.asp?date_req=".
         date("d/m/Y");
  // Загружаем файл
  $content = file_get_contents($url);
  // Регулярное выражение
  $pattern = "|<valute
id=\"([^\"]+)\">[\s]*<NumCode>([^<]+)</NumCode>[\s]*<CharCode>
([^<]+)</CharCode>[\s]*<Nominal>([^<]+)</Nominal>[\s]*<Name>([^<]+)</Name>[\s]*
<Value>([^<]+)</Value>[\s]*</Valute>|is";
```

```
  preg_match_all($pattern, $content, $out);
  for($i = 0; $i < count($out[1]); $i++)
  {
    $out[6][$i] = str_replace(",",".",$out[6][$i]);
    if($out[3][$i] == "USD")
      echo sprintf("Доллар — %3.2f<br>", $out[6][$i]/$out[4][$i]);
    if($out[3][$i] == "EUR")
      echo sprintf("Евро — %3.2f<br>", $out[6][$i]/$out[4][$i]);
  }
?>
```

В курсе валют необходимо заменить запятую точкой при помощи функции `str_replace()`, иначе преобразование типа к числовому приведет к тому, что дробная часть будет отброшена.



ID	Числовой код	Символьный код	Номинал	Название	Курс
R01010	036	AUD	1	Австралийский доллар	29,8317
R01020A	944	AZN	1	Азербайджанский манат	35,3917
R01035	826	GBP	1	Фунт стерлингов Соединенного королевства	45,6402
R01060	051	AMD	1000	Армянских драмов	74,3142
R01090	974	BYR	10000	Белорусских рублей	56,0267
R01100	975	BGN	1	Болгарский лев	20,4980

Рис. 5.4. Результат работы скрипта из листинга 5.17

Если использование регулярных выражений для разбора XML-файлов кажется сложным, можно воспользоваться встроенными средствами разбора XML-файлов. Для получения и разбора XML-файлов традиционно используется функция `simplexml_load_file()`, которая возвращает объект, представляющий структуру XML-файла. В листинге 5.19 представлен скрипт, выводящий курсы доллара США и евро на текущий день.

Листинг 5.19. Загрузка курса доллара и евро

```
<?php
  // Ссылка на XML-файл
  $url = "http://www.cbr.ru/scripts/XML_daily.asp?date_req=".
         date("d/m/Y");
```

```
  // Загружаем файл
  $xml = simplexml_load_file($url);
  foreach($xml as $index => $value)
  {
    if($value->CharCode == "USD")
      echo sprintf("Доллар — %3.2f<br>", $value->Value);
    if($value->CharCode == "EUR")
      echo sprintf("Евро — %3.2f<br>", $value->Value);
  }
?>
```

5.10. Отправка данных методом POST

Создадим простейшую форму, состоящую из одного текстового поля `name` и кнопки для отправки данных (листинг 5.20).

Листинг 5.20. HTML-форма

```
<form action='handler.php' method='post'>
Имя : <input type='text' name='name'><br>
Пароль : <input type='text' name='pass'><br>
<input type='submit' value='Отправить'>
</form>
```

После заполнения текстового поля и нажатия на кнопку **Отправить** данные методом POST отправляются обработчику handler.php, код которого представлен в листинге 5.21.

Листинг 5.21. Обработчик handler.php

```
<?php
  echo "Имя — $_POST[name] <br>";
  echo "Пароль — $_POST[pass] <br>";
?>
```

Обработчик выводит в окно браузера текст, введенный в текстовые поля `name` и `pass` HTML-формы. HTML-форма помогает пользователю сформировать POST-запрос, который затем отсылается браузером. Такой запрос может быть сформирован и скриптом.

При обращении к серверу при помощи метода POST помимо HTTP-заголовка `POST /path HTTP/1.1` необходимо передать заголовок `Content-Length`, указав в нем количество байтов в области данных.

Метод POST, в отличие от метода GET, посылает данные не в строке запроса, а в области данных, после заголовков. Передача нескольких переменных аналогична методу GET: группы `имя=значение` объединяются при помощи символа амперсан-

да (&). Учитывая, что HTML-форма принимает параметр name=Игорь, pass=пароль, строка данных может выглядеть следующим образом:

```
name=Игорь&pass=пароль
```

Кроме этого, необходимо учитывать, что данные передаются в текстовом виде, поэтому все национальные символы следует подвергать кодированию при помощи функции urlencode().

Скрипт, отправляющий данные методом POST через сокеты, представлен в листинге 5.22.

Листинг 5.22. Отправка данных методом POST через сокеты

```
<?php
  $hostname = "localhost";
  $path = "/puzzles/handler.php";
  $line = "";
  // Устанавливаем соединение, имя которого
  // передано в параметре $hostname
  $fp = fsockopen($hostname, 80, $errno, $errstr, 30);
  // Проверяем успешность установки соединения
  if (!$fp) echo "$errstr ($errno)<br />\n";
  else
  {
    // Данные HTTP-запроса
    $data =
      "name=".urlencode("Игорь")."&pass=".urlencode("пароль")."\r\n\r\n";
    // Заголовок HTTP-запроса
    $headers = "POST $path HTTP/1.1\r\n";
    $headers .= "Host: $hostname\r\n";
    $headers .= "Content-type: application/x-www-form-urlencoded\r\n";
    $headers .= "Content-Length: ".strlen($data)."\r\n\r\n";
    // Отправляем HTTP-запрос серверу
    fwrite($fp, $headers.$data);
    // Получаем ответ
    while (!feof($fp))
    {
      $line .= fgets($fp, 1024);
    }
    fclose($fp);
  }
  echo $line;
?>
```

Результат работы скрипта может выглядеть следующим образом:

```
HTTP/1.1 200 OK
Date: Wed, 16 Nov 2005 18:00:44 GMT
```

```
Server: Apache/1.3.33 (Win32)
X-Powered-By: PHP/5.0.4
Transfer-Encoding: chunked
Content-Type: text/html

22
Имя — Игорь
Пароль — пароль
```

Остается только удалить HTTP-заголовки, и результат будет идентичен обращению к обработчику из HTML-формы.

> ***Замечание***
>
> Такого рода скрипты используются для автопостинга — автоматического размещения рекламных или провокационных сообщений в гостевых книгах и форумах в значительных количествах. Самым эффективным способом защиты от такого вида атак является автоматическая генерация изображения с кодом, который помещается в сессию. Пока пользователь не введет код в HTML-форму, сервис не срабатывает. Изображение может быть дополнено помехами, которые не помешают его прочитать "живому" посетителю, но потребуют от злоумышленника решения серьезной задачи распознавания образов.

В листинге 5.23 приводится решение рассматриваемой задачи с использованием CURL.

Листинг 5.23. Загрузка POST-данных с использованием CURL

```
<?php
  // Задаем адрес удаленного сервера
  $curl = curl_init("http://localhost/puzzles/5/handler.php");

  // Передача данных осуществляется методом POST
  curl_setopt($curl, CURLOPT_POST, 1);
  // Задаем POST-данные
  $data = "name=".urlencode("Игорь").
          "&pass=".urlencode("пароль")."\r\n\r\n";
  curl_setopt($curl, CURLOPT_POSTFIELDS, $data);

  // Выполняем запрос
  curl_exec($curl);
  // Закрываем CURL-соединение
  curl_close($curl);
?>
```

Для того чтобы сообщить CURL о том, что данные будут передаваться на сервер методом POST, необходимо задать параметр `CURLOPT_POST`. POST-данные устанавливаются при помощи параметра `CURLOPT_POSTFIELDS`.

5.11. Передача реферера

Переходы пользователей со стороннего сайта фиксируются в *реферере*, получить доступ к которому можно при помощи элемента суперглобального массива `$_SERVER['HTTP_REFERER']`. Так, если страница 1.php содержит ссылку на страницу 2.php (`<a href='2.php'>Ссылка</a>`), разместив в файле 2.php следующий код из листинга 5.24, мы получим результат, приведенный на рис. 5.5.

Листинг 5.24. Вывод реферера

```
<?php
  echo "Вы перешли на текущую страницу с ".$_SERVER['HTTP_REFERER'];
?>
```

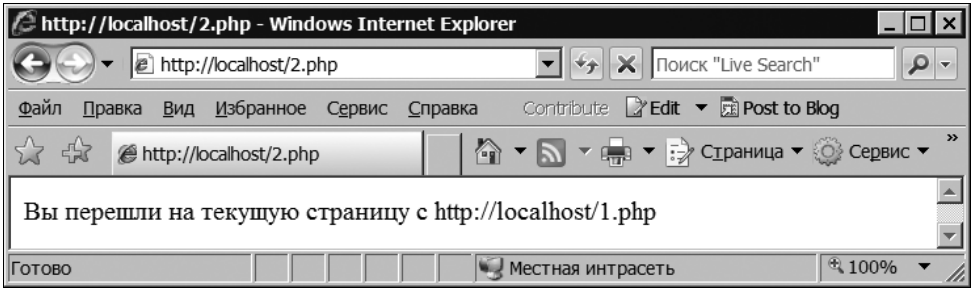


Рис. 5.5. Вывод реферера

Реферер иногда используется для простейшей защиты HTML-формы от автопостинга (заполнения HTML-формы при помощи скрипта). Рассмотрим это на примере. Пусть имеется HTML-форма авторизации в файле index.php, код которого представлен в листинге 5.25.

Листинг 5.25. HTML-форма системы авторизации

```
<table>
   <form action='handler.php' method='post'>
   <tr>
     <td>Имя:</td>
     <td><input type='text' name='name'></td>
   </tr>
   <tr>
     <td>Пароль:</td>
     <td><input type='password' name='pass'></td>
   </tr>
   <tr>
     <td> </td>
     <td><input type='submit' value='Войти'></td>
```

```
   </tr>
   </form>
</table>
```

Для защиты HTML-формы от заполнения на стороннем сайте в обработчик HTML-формы handler.php можно добавить проверку по реферeру, входит ли в его состав имя текущего сервера (листинг 5.26).

Листинг 5.26. Защита обработчика HTML-формы при помощи реферера

```
<?php
  // Формируем путь к HTML-форме на текущем сервере
  $html = "http://".$_SERVER['SERVER_NAME'];
  // Проверяем, переданы ли данные с текущего сервера
  if (substr($_SERVER['HTTP_REFERER'], 0, strlen($html)) == $html)
  {
    if ($_POST['name'] == 'admin' && $_POST['pass'] == 'admin')
    {
       // Обработчик HTML-формы
    }
  }
  else
  {
    echo "Результат из формы не может быть обработан";
  }
?>
```

Однако такой способ защиты редко используется в реальной практике. Браузер или брандмауэр могут скрывать referer, поэтому такого рода защита может приводить к некорректной работе Web-приложения для легитимных пользователей. С другой стороны, referer легко подделывается: достаточно при помощи сокетов отправить HTTP-заголовок `Referer` (листинг 5.27) или воспользоваться CURL, установив при помощи функции `curl_setopt()` параметр `CURLOPT_REFERER` (листинг 5.28).

Листинг 5.27. Отправка реферера при помощи сокетов

```
<?php
  $hostname = "localhost";
  $path = "/test/handler.php";
  $line = "";

  // Передаем методом POST имя пользователя (admin),
  // его пароль (admin), скрытое поле session_id ($SID)
  // В заголовках передаем cookie PHPSESSID
  // Устанавливаем соединение, имя которого
  // передано в параметре $hostname
  $fp = fsockopen($hostname, 80, $errno, $errstr, 30);
```

```
  // Проверяем успешность установки соединения
  if (!$fp) echo "$errstr ($errno)<br />\n";
  else
  {
    // Данные POST-запроса
    $data = "name=admin&pass=admin&\r\n\r\n";
    // Формируем HTTP-заголовки для передачи его серверу
    $headers = "POST $path HTTP/1.1\r\n";
    $headers .= "Host: $hostname\r\n";
    $headers .= "Content-type: application/x-www-form-urlencoded\r\n";
    $headers .= "Content-Length: ".strlen($data)."\r\n";
    // Подделываем реферер
    $headers .= "Referer: http://localhost/test/index.php\r\n";
    $headers .= "Connection: Close\r\n\r\n";
    // Отправляем HTTP-запрос серверу
    fwrite($fp, $headers.$data);
    // Получаем ответ
    while (!feof($fp))
    {
      $line .= fgets($fp, 1024);
    }
    fclose($fp);
  }
  echo $line;
?>
```

Листинг 5.28. Отправка реферера при помощи CURL

```
<?php
  // Задаем адрес удаленного сервера
  $curl = curl_init("http://localhost/test/handler.php");

  // Передача данных осуществляется методом POST
  curl_setopt($curl, CURLOPT_POST, 1);
  // Задаем POST-данные
  $data = "name=admin&pass=admin&\r\n\r\n";
  curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
  // Устанавливаем реферер
  curl_setopt($curl, CURLOPT_REFERER,
              "http://localhost/test/index.php");

  // Выполняем запрос
  curl_exec($curl);
  // Закрываем CURL-соединение
  curl_close($curl);
?>
```

5.12. Передача пользовательского агента

При обращении PHP-скрипта к страницам сайта при помощи файловых функций сервер делает в переменную окружения `USER_AGENT` запись вида: `PHP 5.3`. Вид этой записи определяется директивой конфигурационного файла php.ini `user_agent` *(см. разд. 2.1.11)*. В результате скрипт получает доступ к этому идентификатору через элемент суперглобального массива `$_SERVER['USER_AGENT']`. Однако этот способ проверки не может считаться универсальным, т. к. переменную окружения устанавливает клиент при помощи HTTP-заголовка `User-Agent`. Любой HTTP-заголовок, который передает клиент, может быть изменен. В листинге 5.29 приводится пример, в котором скрипт, обращаясь к другому скрипту, выдает себя за пользователя операционной системы Windows XP, использующего браузер Internet Explorer версии 6.0. Для этого в качестве HTTP-заголовка `User-Agent` передается следующая строка:

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
```

Листинг 5.29. Фальсифицируем пользовательский агент

```
<?php
  $hostname = "localhost";
  $path = "/test/index.php";
  $line = "";

  // Устанавливаем соединение, имя которого
  // передано в параметре $hostname
  $fp = fsockopen($hostname, 80, $errno, $errstr, 30);
  // Проверяем успешность установки соединения
  if (!$fp) echo "$errstr ($errno)<br />\n";
  else
  {
    // Формируем HTTP-заголовки для передачи серверу
    $headers = "GET $path HTTP/1.1\r\n";
    $headers .= "Host: $hostname\r\n";
    // Подделываем пользовательский агент, маскируясь
    // под пользователя Windows XP
    $headers .= "User-Agent: Mozilla/4.0 ".
                "(compatible; MSIE 6.0; Windows NT 5.1)\r\n";
    $headers .= "Connection: Close\r\n\r\n";
    // Отправляем HTTP-запрос серверу
    fwrite($fp, $headers);
    // Получаем ответ
    while (!feof($fp))
    {
      $line .= fgets($fp, 1024);
    }
```

```
    fclose($fp);
  }
  echo $line;
?>
```

В листинге 5.30 представлено решение той же задачи с использованием расширения CURL. Для этого при помощи функции `curl_setopt()` устанавливается параметр `CURLOPT_USERAGENT` — его значение и будет передаваться серверу в HTTP-заголовке `User-Agent`.

Листинг 5.30. Фальсификация пользовательского агента при помощи CURL

```
<?php
  // Задаем адрес удаленного сервера
  $curl = curl_init("http://localhost/test/handler.php");

  // Устанавливаем реферер
  $useragent = "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1";
  curl_setopt($curl, CURLOPT_USERAGENT, $useragent);

  // Выполняем запрос
  curl_exec($curl);
  // Закрываем CURL-соединение
  curl_close($curl);
?>
```

5.13. Передача cookie

Механизм действия cookie заключается в том, что, получив указания от сервера, браузер создает в оперативной памяти (если это сессионный cookie) или на жестком диске файл с заданными параметрами. После этого при каждом запросе браузер оправляет серверу HTTP-заголовок `Cookie` с именем и значением cookie. Так как за установку HTTP-заголовка несет ответственность клиентская сторона, cookie легко можно подделать, отправив HTTP-заголовок следующего формата:

```
Cookie: name1=value1; name2=value2; ...
```

где `name1`, `name2`, ... — имена cookie, а `value1`, `value2`, ... — их значения.

Пусть имеется скрипт, который проверяет наличие cookie, установленного у посетителя при аутентификации. В cookie `name` устанавливается имя пользователя, а установленные cookie `admin`, `editor` и `user` позволяют провести авторизацию для администратора, редактора и пользователя соответственно (листинг 5.31).

Листинг 5.31. Предоставление доступа к информации по cookie

```
<?php
  if (isset($_COOKIE['name']))
  {
```

```
    if (isset($_COOKIE['admin']))
    {
       echo "Здравствуйте, администратор $_COOKIE[name] !<br>";
    }
    if (isset($_COOKIE['editor']))
    {
       echo "Здравствуйте, редактор $_COOKIE[name] !<br>";
    }
    if (isset($_COOKIE['user']))
    {
       echo "Здравствуйте, пользователь $_COOKIE[name] !<br>";
    }
  }
?>
```

Скрипт, осуществляющий авторизацию с правами администратора, представлен в листинге 5.32.

> ***Замечание***
>
> Содержимое сессий не может быть фальсифицировано, т. к. хранящиеся в них данные не покидают пределов сервера. Однако фальсификации может быть подвергнут уникальный идентификатор сессии SID, который, как правило, передается через cookie. Таким образом скрипт может вести себя как браузер, поддерживая сессию, или, если злоумышленник получил доступ к идентификатору действующей сессии (например, в результате XSS-атаки), получать доступ к данным другого пользователя.

Листинг 5.32. Подделка cookie

```
<?php
  $hostname = "localhost";
  $path = "/1.php";

  // Устанавливаем соединение, имя которого
  // передано в параметре $hostname
  $fp = fsockopen($hostname, 80, $errno, $errstr, 30);
  // Проверяем успешность установки соединения
  if (!$fp) echo "$errstr ($errno)<br />\n";
  else
  {
    // Формируем HTTP-заголовки для передачи серверу
    $headers = "GET $path HTTP/1.1\r\n";
    $headers .= "Host: $hostname\r\n";
    // Подделываем cookie
    $headers .= "Cookie: name=cheops; admin=1;\r\n";
    $headers .= "Connection: Close\r\n\r\n";
    // Отправляем HTTP-запрос серверу
    fwrite($fp, $headers);
```

```
    // Получаем ответ
    $line = "";
    while (!feof($fp))
    {
      $line .= fgets($fp, 1024);
    }
    fclose($fp);
  }
  echo $line;
?>
```

Для того чтобы подделать cookie, скрипт отправляет HTTP-заголовок:

```
Cookie: name=cheops; admin=1;\r\n
```

В нем передается имя пользователя `cheops` и cookie `admin` со значением 1. Результат работы скрипта представлен на рис. 5.6. HTTP-заголовки ответа, которые предваряют вывод, не фильтруются, но могут быть легко удалены в случае необходимости.

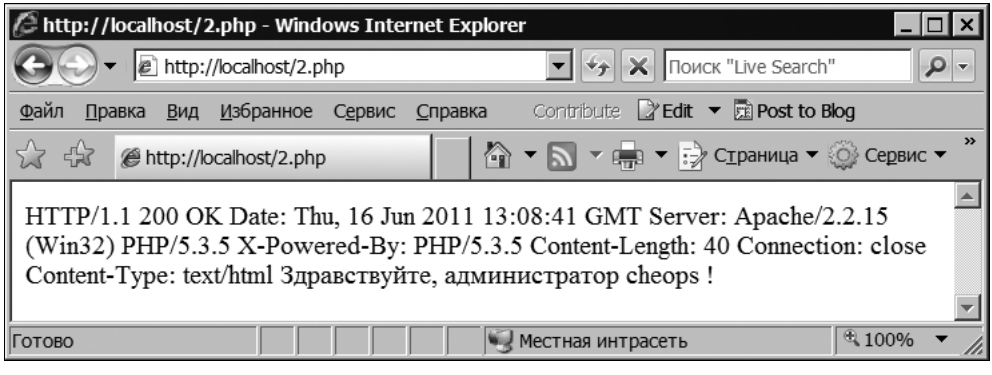


Рис. 5.6. Авторизация с правами администратора

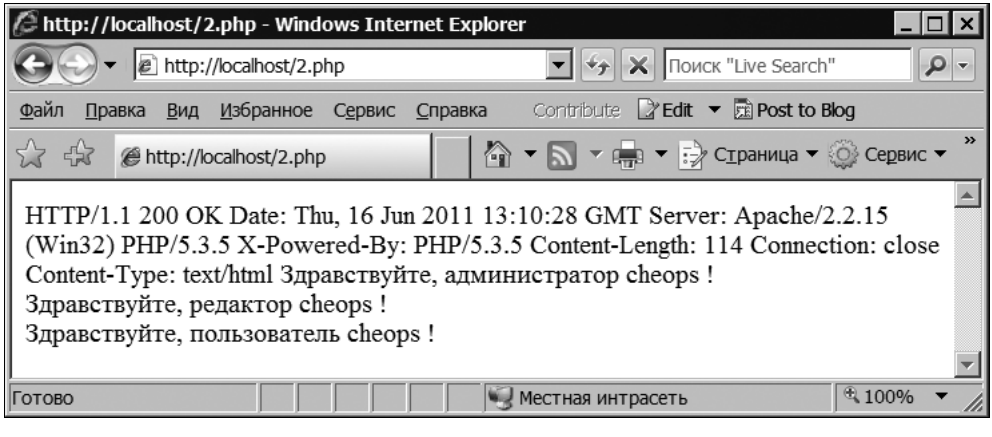


Рис. 5.7. Авторизация одновременно всех трех видов пользователей

Не составляет труда авторизоваться также с правами редактора или пользователя: достаточно заменить имя cookie `admin` на `editor` или `user`. Более того, в условиях скрипта из листинга 5.32 можно произвести авторизацию одновременно для всех трех видов пользователей. Для этого достаточно отправить HTTP-заголовок вида:

```
Cookie: name=cheops; admin=1; editor=1; user=1;\r\n
```

Результат работы такого скрипта представлен на рис. 5.7.

5.14. Определение IP-адреса по сетевому адресу

Все хосты в сети Интернет имеют уникальные IP-адреса, однако для удобства вместо них используются доменные имена, которые при помощи DNS-серверов привязываются к IP-адресам. В протоколе HTTP 1.1 появилась возможность привязывать к одному IP-адресу несколько доменных имен.

Помимо функции `fsockopen()`, рассмотренной ранее, PHP располагает несколькими функциями, предназначенными для работы с IP-адресами и доменными именами (табл. 5.6). Кроме того, при помощи сокетов можно обратиться к Whois-сервису, который предоставляет информацию об IP-адресах.

Таблица 5.6. *Функции для работы с IP-адресами и доменными именами*

Функция	Описание
`checkdnsrr($hostname [, $type])`	Находит на DNS-сервере записи ресурсов вида `$type` для хоста `$hostname`
`gethostbyname($hostname)`	Возвращает для доменного имени `$hostname` соответствующий ему IP-адрес
`gethostbynamel($hostname)`	Возвращает для доменного имени `$hostname` соответствующие ему IP-адреса
`gethostbyaddr($ip_address)`	Для IP-адреса `$ip_address` возвращает соответствующее ему доменное имя
`ip2long($ip_address)`	Преобразует IP-адрес `$ip_address` в целое число
`long2ip($proper_address)`	Преобразует целое число в IP-адрес `$proper_address`

Функция `gethostbyname()` принимает в качестве аргумента строку, представляющую доменное имя компьютера, и возвращает соответствующий IP-адрес. Функция имеет следующий синтаксис:

```
gethostbyname($hostname)
```

В листинге 5.33 приводится пример получения IP-адреса сервера сайта **http://www.yandex.ru**.

Листинг 5.33. Использование функции gethostbyname()

```
<?php
  $hostname = "www.yandex.ru";
  $ip_address = gethostbyname($hostname);
  echo "IP-адрес $hostname: $ip_address";
?>
```

Если компьютер подключен к Интернету, результатом работы скрипта из листинга 5.33 может быть строка:

```
IP-адрес www.yandex.ru: 93.158.134.203
```

Многие компьютеры имеют несколько IP-адресов; особенно типична такая ситуация для серверов. Получить полный список IP-адресов, соответствующих данному имени компьютера, можно с помощью функции `gethostbynamel()`, действующей аналогично функции `gethostbyname()`. Функция имеет следующий синтаксис:

```
gethostbynamel($hostname)
```

Другая ситуация, в которой полезно применение этой функции — если одно имя DNS соответствует нескольким компьютерам. Это случается при работе с DNS-серверами, поддерживающими механизм кругового распределения нагрузки, при котором одно имя DNS-сервера отображается на несколько компьютеров в локальной сети этого сервера.

Возвращаемый список IP-адресов функция `gethostbynamel()` помещает в массив (листинг 5.34).

Листинг 5.34. Использование функции gethostbynamel()

```
<?php
  $hostname = "www.yandex.ru";
  $ip_address = gethostbynamel($hostname);
  for($i = 0; $i < count($ip_address); $i++)
  {
    echo "IP-адрес $hostname: {$ip_address[$i]}<br>";
  }
?>
```

Если компьютер подключен к Интернету, результатом работы скрипта из листинга 5.34 может быть следующий дамп:

```
IP-адрес www.yandex.ru: 93.158.134.203
IP-адрес www.yandex.ru: 213.180.204.3
IP-адрес www.yandex.ru: 77.88.21.3
IP-адрес www.yandex.ru: 87.250.250.3
IP-адрес www.yandex.ru: 87.250.250.203
IP-адрес www.yandex.ru: 87.250.251.3
IP-адрес www.yandex.ru: 93.158.134.3
```

5.15. Определение сетевого адреса по IP-адресу

Функция `gethostbyaddr()` принимает в качестве аргумента IP-адрес `$ip_address` и возвращает соответствующее ему имя хоста. Функция имеет следующий синтаксис:

```
gethostbyaddr($ip_address)
```

В листинге 5.35 приводится пример использования функции `gethostbyaddr()` применительно к локальному хосту.

Листинг 5.35. Использование функции `gethostbyaddr()`

```
<?php
  $ip_address = "127.0.0.1";
  $hostname = gethostbyaddr($ip_address);
  echo "Имя хоста с IP-адресом $ip_address: $hostname";
?>
```

Результатом работы скрипта из листинга 5.35 будет строка:

```
Имя хоста с IP-адресом 127.0.0.1: localhost
```

5.16. Получение информации об IP-адресе

Сервис Whois позволяет определить, на кого зарегистрирован тот или иной IP-адрес. Для этого достаточно послать IP-адрес по 43 порту соответствующего Whois-сервера. Каждый из множества Whois-серверов несет ответственность за IP-адреса той или иной части света. Главным сервером является **whois.arin.net**. Если главный Whois-сервис не содержит информации о запрашиваемом IP-адресе, то в отчете обязательно присутствует строка `ReferralServer`, которая указывает адрес Whois-сервиса, ответственного за нужный диапазон IP-адресов. Поэтому обращение к Whois-сервису удобно выделить в отдельную функцию `whois()`, которая будет принимать два параметра: адрес Whois-сервиса и запрашиваемый IP-адрес. Если полученный отчет будет содержать ссылку на реферальный сервис, функция будет осуществлять рекурсивный спуск (листинг 5.36).

Листинг 5.36. Рекурсивное следование реферальному серверу

```
<center>
 <form method='post'>
 <input type='text' name='ip' size='35'
        value='<?php
 if (!empty($_POST['ip'])) echo htmlspecialchars($_POST['ip']);
 ?>'>
 <input type='submit' value='Введите IP-адрес'>
```

```
 </form>
</center>
<?php
if (!empty($_POST['ip'])) echo whois("whois.arin.net",$_POST['ip']);

function whois($url,$ip)
{
  // Соединение с сокетом TCP, ожидающим на сервере
  // "whois.arin.net" по порту 43.
  // В результате возвращается дескриптор соединения $sock.
  $sock = fsockopen($url, 43, $errno, $errstr);
  if (!$sock) exit("$errno($errstr)");
  else
  {
    echo $url."<br>";
    // Записываем строку из переменной $_POST["ip"]
    // в дескриптор сокета.
    fputs ($sock, $ip."\r\n");
    // Осуществляем чтение из дескриптора сокета
    $text = "";
    while (!feof($sock))
    {
      $text .= fgets ($sock, 128)."<br>";
    }
    // Закрываем соединение
    fclose ($sock);

    // Ищем реферальный сервер
    $pattern = "|ReferralServer: whois://([^\n<:]+)|i";
    preg_match($pattern, $text, $out);
    if (!empty($out[1])) return whois($out[1], $ip);
    else return $text;
  }
}
?>
```

Скрипт всегда начинает поиск с **whois.arin.net**, при необходимости осуществляя повторные запросы к реферальным сервисам. При этом перед отчетом выводится список Whois-сервисов, которые посещает скрипт. Так для адреса 213.115.162.82, который соответствует сайту корпорации AB MySQL, отчет может выглядеть так, как это представлено на рис. 5.8.

Как видно из рис. 5.8, скрипт посещает два Whois-сервиса: **whois.arin.net** и **whois.ripe.net**.

Для IP-адреса 192.131.90.161, который соответствует сетевому адресу **http://www.w3c.org**, отчет может выглядеть так, как это представлено на рис. 5.9. Из отчета видно, что информация по данному IP-адресу была найдена в Whois-сервисе по адресу **whois.apnic.net**.

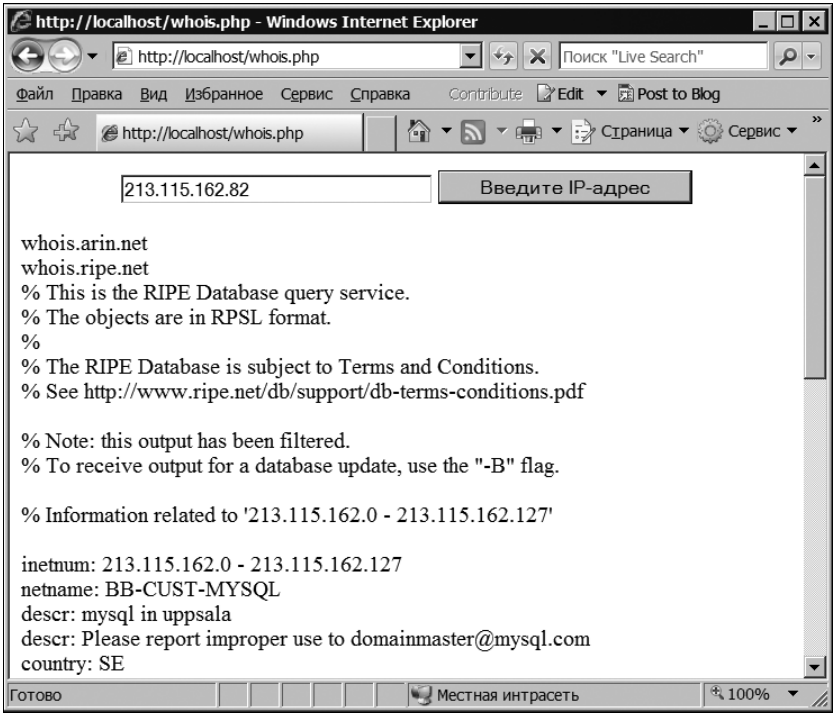


Рис. 5.8. Принадлежность IP-адреса 213.115.162.82

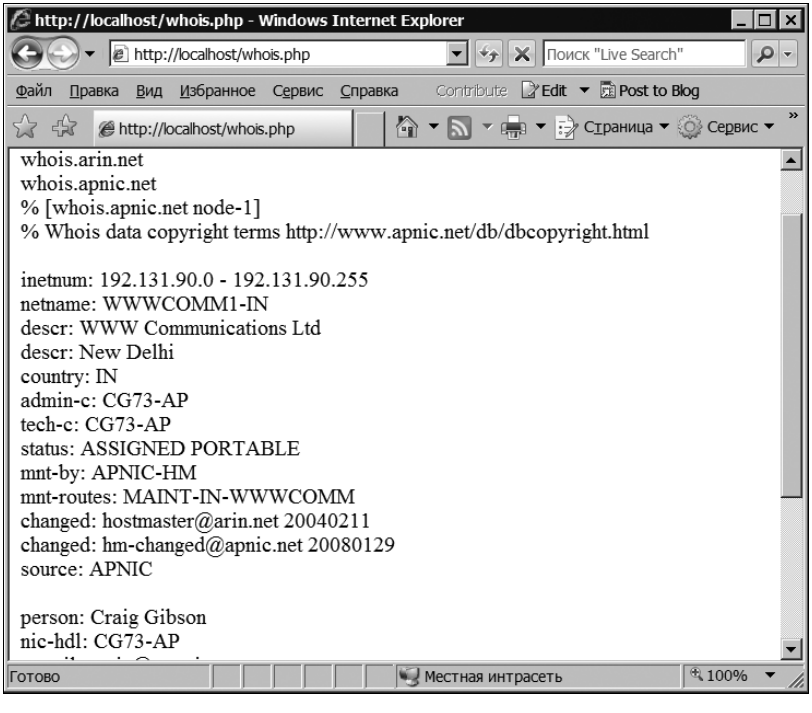


Рис. 5.9. Принадлежность IP-адреса 192.131.90.161

5.17. Отправка почтового сообщения

Отправка почты средствами PHP осуществляется при помощи функции `mail()`, которая имеет следующий синтаксис:

```
mail($to, $subject, $body [, $headers] [, $parameters])
```

Эта функция принимает следующие аргументы:

- `$to` — адрес электронной почты получателя;
- `$subject` — тема сообщения;
- `$message` — текст сообщения;
- `$headers` — дополнительные заголовки, которые можно задать в сообщении;
- `$parameters` — дополнительные параметры, которые можно задать в сообщении.

Если не указывается четвертый параметр `$headers`, письмо не снабжается никакими дополнительными почтовыми заголовками. Однако очень часто требуется изменить формат письма вместо обычного текста (`text/plain`) на HTML-формат (`text/html`) или указать кодировку сообщения. Установка формата письма и его кодировки осуществляется при помощи почтовых заголовков `Content-Type` и `charset` соответственно.

```
Content-Type: text/html; charset=UTF-8\r\n
```

> ***Замечание***
>
> Для отправки почтового сообщения в кодировке cp1251 вместо `UTF-8` следует прописать `windows-1251`.

Таким образом, скрипт, выполняющий отправку почтового сообщения, может выглядеть так, как это представлено в листинге 5.37.

Листинг 5.37. Отправка почтового сообщения при помощи функции `mail()`

```
<?php
  $theme = "Отчет с сайта";
  $message = "<html>
              <head></head>
              <body>
              Письмо отправлено — ".date("d.m.Y H:i:s")."<br>
              Размер скрипта отправителя — ".filesize($_SERVER['PHP_SELF'])."
              </body>
              </html>";
  $headers = "Content-Type: text/html; charset=UTF-8\r\n";
  if(mail($to, $subject, $message, $headers))
  {
    echo "Письмо успешно отправлено";
  }
```

```
  else
  {
    echo "Произошла ошибка — письмо не отправлено";
  }
?>
```

В качестве отправителя будет подставлен адрес сервера. В качестве обратного адреса можно назначить произвольный адрес при помощи почтового заголовка `From`:

```
From: name <e-mail>
```

Вместо *name* указывается имя, которое будет отображаться клиентским почтовым агентом как имя отправителя, а *e-mail* содержит обратный почтовый адрес. Так строки формирования переменной `$headers` могут выглядеть следующим образом (листинг 5.38).

Листинг 5.38. Формирование почтовых заголовков

```
<?php
  $headers = "Content-Type: text/html; charset=KOI8-R\r\n";
  $headers .= "From: server <someone@somewhere.ru>\r\n\r\n";
?>
```

Отправление, снабженное почтовыми заголовками, будет представлено как письмо от пользователя `server` с электронным адресом `someone@somewhere.ru`.

5.18. Отправка писем с вложением

Для того чтобы отправить почтовое сообщение с прикрепленным к нему файлом, необходимо особым способом подготовить текст письма и снабдить его соответствующими почтовыми заголовками. Для отправки почтового сообщения создадим HTML-форму, состоящую из двух текстовых полей для адреса назначения `mail_to`, темы сообщения `mail_subject`, текстовой области `mail_msg` для ввода содержимого письма, поля типа `file` для выбора файла отправки и кнопки, позволяющей отправить данные из HTML-формы обработчику (листинг 5.39).

Листинг 5.39. HTML-форма для отправки почтового сообщения

```
<?php
if(!empty($_POST))
{
  // Обработчик HTML-формы
  include "handler.php";
}
?>
```

```
<table>
<form enctype='multipart/form-data' method='post'>
<tr>
  <td width=50%>To:</td>
  <td align='right'>
    <input type='text' name='mail_to' maxlength='32'>
  </td>
</tr>
<tr>
  <td width=50%>Subject:</td>
  <td align='right'>
    <input type='text' name='mail_subject' maxlength='64'>
  </td>
</tr>
<tr>
  <td colspan='2'>
   Сообщение:<br><textarea cols='50' rows='8' name='mail_msg'></textarea>
  </td>
</tr>
<tr>
  <td width='50%'>Photo:</td>
  <td align='right'>
    <input type='file' name='mail_file' maxlength='64'>
  </td>
</tr>
<tr><td colspan='2'><input type='submit' value='Отправить'></td></tr>
</form>
</table>
```

Внешний вид HTML-формы представлен на рис. 5.10.

В качестве обработчика формы служит файл handler.php, который включается в начале формы и имеет следующее содержание (листинг 5.40).

Листинг 5.40. Обработчик HTML-формы

```
<?php
  if(empty($_POST['mail_to'])) exit("Введите адрес получателя");
  // проверяем правильность заполнения с помощью регулярного выражения
  $pattern = "/^[0-9a-z_]+@[0-9a-z_^\.]+\.[a-z]{2,6}$/i";
  if (!preg_match($pattern, $_POST['mail_to']))
  {
   exit("Введите адрес в виде somebody@server.com");
  }
  $_POST['mail_to'] = htmlspecialchars(stripslashes($_POST['mail_to']));
  $_POST['mail_subject'] =
                  htmlspecialchars(stripslashes($_POST['mail_subject']));
```

```
  $_POST['mail_msg'] =
                      htmlspecialchars(stripslashes($_POST['mail_msg']));
  $picture = "";
  // Если поле выбора вложения не пустое — закачиваем его на сервер
  if (!empty($_FILES['mail_file']['tmp_name']))
  {
    // Загружаем файл
    $path = $_FILES['mail_file']['name'];
    if (copy($_FILES['mail_file']['tmp_name'], $path)) $picture = $path;
  }
  $thm = $_POST['mail_subject'];
  $msg = $_POST['mail_msg'];
  $mail_to = $_POST['mail_to'];
  // Отправляем почтовое сообщение
  if(empty($picture)) mail($mail_to, $thm, $msg);
  else send_mail($mail_to, $thm, $msg, $picture);
  // Вспомогательная функция для отправки почтового сообщения с вложением
  function send_mail($to, $thm, $html, $path)
  {
    $fp = fopen($path,"r");
    if (!$fp)
    {
      print "Файл $path не может быть прочитан";
      exit();
    }
    $file = fread($fp, filesize($path));
    fclose($fp);

    $boundary = "--".md5(uniqid(time())); // генерируем разделитель
    $headers .= "MIME-Version: 1.0\n";
    $headers .="Content-Type: multipart/mixed; boundary=\"$boundary\"\n";
    $multipart .= "--$boundary\n";
    $kod = 'utf-8'; // или $kod = 'windows-1251';
    $multipart .= "Content-Type: text/html; charset=$kod\n";
    $multipart .= "Content-Transfer-Encoding: Quot-Printed\n\n";
    $multipart .= "$html\n\n";

    $message_part = "--$boundary\n";
    $message_part .= "Content-Type: application/octet-stream\n";
    $message_part .= "Content-Transfer-Encoding: base64\n";
    $message_part .= "Content-Disposition: attachment; filename =
\"".$path."\"\n\n";
    $message_part .= chunk_split(base64_encode($file))."\n";
    $multipart .= $message_part."--$boundary--\n";

    if(!mail($to, $thm, $multipart, $headers))
    {
```

```
      exit("К сожалению, письмо не отправлено");
    }
  }
  // Автоматический переход на главную страницу форума
  @header("Location: ".$_SERVER['PHP_SELF']);
?>
```

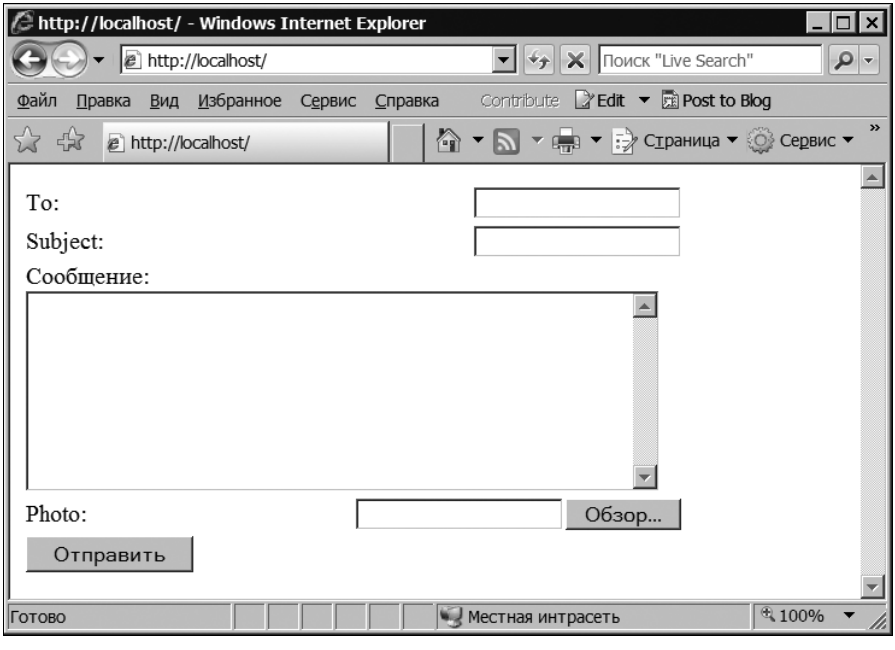


Рис. 5.10. HTML-форма для отправки почтового сообщения

После проверки корректности ввода проверяется, прикрепил ли пользователь файл к сообщению. Если вложение отсутствует, письмо отправляется при помощи стандартной функции `mail()`. В противном случае вложение загружается из временного каталога в текущий, а письмо отправляется при помощи собственной функции `send_mail()`. Первые три параметра функции `send_mail()` совпадают с параметрами функции `mail()`. В качестве четвертого параметра передается имя файла, который необходимо прикрепить к почтовому сообщению. Далее функция подготавливает тело сообщения и почтовые заголовки для передачи сообщения с прикрепленным файлом.

5.19. Отправка писем со встроенными изображениями

Задача отправки дополнительных изображений, которые встраиваются в HTML-код, не прикрепляясь в виде вложения, решается посредством функции из листинга 5.41.

Листинг 5.41. Отправка встроенных изображений

```
<?php
  function htmlimgmail($mail_to, $thema, $html, $path)
  {
    $EOL = "\n";
    $boundary     = "--".md5(uniqid(time()));
    $headers    = "MIME-Version: 1.0;$EOL";
    $headers .= "From: somebody@somewhere.ru$EOL";
    $headers   .= "Content-Type: multipart/related; ".
                  "boundary=\"$boundary\"$EOL";

    $multipart  = "--{$boundary}$EOL";
    $multipart .= "Content-Type: text/html; charset=koi8-r$EOL";
    $multipart .= "Content-Transfer-Encoding: 8bit$EOL";
    $multipart .= $EOL;
    if($EOL == "\n")  $multipart .= str_replace("\r\n", $EOL, $html);
    $multipart .= $EOL;

    if (!empty($path))
    {
      for($i = 0; $i < count($path); $i++)
      {
        $file = file_get_contents($path[$i]);
        $name = basename($path[$i]);
        $multipart .=  "$EOL--$boundary$EOL";
        $multipart .= "Content-Type: image/jpeg; name=\"$name\"$EOL";
        $multipart .= "Content-Transfer-Encoding: base64$EOL";
        $multipart .= "Content-ID: <".md5($name).">$EOL";
        $multipart .= $EOL;
        $multipart .= chunk_split(base64_encode($file), 76, $EOL);
      }
    }
    $multipart .= "$EOL--$boundary--$EOL";
    if(!is_array($mail_to))
    {
      // Письмо отправляется одному адресату
      if(!mail($mail_to, $thema, $multipart, $headers))
        return False;
      else
        return True;
      exit;
    }
    else
    {
      // Письмо отправляется на несколько адресов
      foreach($mail_to as $mail)
```

```
      {
        mail($mail, $thema, $multipart, $headers);
      }
    }
  }
?>
```

В листинге 5.42 приводится пример использования функции `htmlimgmail()`. Функция принимает в качестве первого параметра `$mail_to` электронный адрес, на который необходимо направить письмо (или массив электронных адресов, если их несколько), через второй параметр `$thema` передается тема сообщения, третий параметр содержит тело письма в HTML-формате. Последний параметр содержит массив с путями файлов. Путь должен указываться от каталога, из которого скрипт вызывается, т. к. функции `htmlimgmail()` потребуется содержимое каждого из изображений. Из HTML-кода ссылаться на встроенные изображения можно при помощи префикса cid, после которого следует уникальный хэш изображения, задаваемый в почтовом заголовке `Content-ID`. Данный хэш вычисляется из названия файла при помощи функции `md5()`.

Листинг 5.42. Использование функции `htmlimgmail()`

```
<?php
  // Отправляем почтовое сообщение
  $picture[0] = "s_20040815135808.jpg";
  $picture[1] = "s_20040815135939.jpg";
  $mail_to = "sombody@somewhere.ru";
  $thm     = convert_cyr_string("Тема сообщения", "w", "k");
  $html = "<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML ".
          "4.01 Transitional//EN\">
           <html>
             <head><title>Почтовая рассылка</title></head>
             <body><img src='cid:".md5($picture[0])."' border='0'>Тело
сообщения<br><br><img src='cid:".md5($picture[1])."' border='0'></body>
           </html>";
  $html = convert_cyr_string($html, "w", "k");
  if(send_mail($mail_to, $thm, $html, $picture))
    echo "Успех ".date("d.m.Y H:i");
  else
    echo "Не отправлено";
?>
```

ГЛАВА 6

Введение в MySQL

Интерфейс файловой системы предоставляет слишком бедные возможности для манипулирования информацией внутри файлов в условиях конкурентного доступа к нему множества пользователей. Ранее разработчикам приходилось реализовывать собственные минибазы данных, которые организовывали запросы в очередь, осуществляли кэширование информации в оперативной памяти, создавали индексы по искомой информации и т. п. Со временем такие комплексы выделялись в отдельный класс приложений, стандартизировался язык запросов к ним (SQL). Эти функции на себя взяли хорошо спроектированные, разрабатываемые в течение десятка лет СУБД. С PHP традиционно используется бесплатная СУБД MySQL. Хотя для PHP разработаны расширения, позволяющие взаимодействовать со всеми широко распространенными СУБД, будь то Oracle, MS SQL, Sybase и т. д.

6.1. Что такое SQL

Системы управления базами данных (СУБД) предназначены для управления большими объемами данных. По сути, база данных — это те же файлы, в которых хранится информация. Сами по себе базы данных не представляли бы никакого интереса, если бы не было систем управления базами данных (СУБД). СУБД — это программный комплекс, который выполняет все низкоуровневые операции по работе с файлами базы данных, оставляя программисту оперировать логическими конструкциями при помощи языка программирования.

> ***Замечание***
>
> Рассмотреть полностью СУБД в рамках одной или нескольких глав невозможно. Дополнительную информацию можно почерпнуть на нашем форуме **http://softtime.ru/forum/index.php?id_forum=3**.

Язык программирования, с помощью которого пользователь осуществляет запросы к базе данных, называется SQL (Structured Query Language, структурированный язык запросов).

Несмотря на то, что SQL называется "языком запросов", в настоящее время этот язык представляет собой нечто большее, чем просто инструмент для создания за-

просов. С помощью SQL осуществляется реализация всех возможностей, которые представляются пользователям разработчиками СУБД, а именно:

- выборка данных (извлечение из базы данных содержащейся в ней информации);
- организация данных (определение структуры базы данных и установление отношений между ее элементами);
- обработка данных (добавление/изменение/удаление);
- управление доступом (ограничение возможностей ряда пользователей на доступ к некоторым категориям данных, защита данных от несанкционированного доступа);
- обеспечение целостности данных (защита базы данных от разрушения);
- управление состоянием СУБД.

SQL является специализированным языком программирования, и в отличие от языков высокого уровня (PHP, C++, Pascal и т. д.) с его помощью нельзя создать полноценную программу. Все запросы выполняются либо в специализированных программах, либо из прикладных программ при помощи специальных библиотек.

Несмотря на то, что язык запросов SQL строго стандартизирован, существует множество его диалектов: по сути, каждая база данных реализует собственный диалект со своими особенностями и ключевыми словами, недоступными в других базах данных. Такая ситуация связана с тем, что стандарты SQL появились достаточно поздно, в то время как компании-поставщики баз данных существуют давно и обслуживают большое число клиентов, для которых требуется обеспечить обратную совместимость со старыми версиями программного обеспечения. Поэтому, когда дело доходит до принятия стандартов, базы данных уже имеют реализацию той или иной особенности, и комиссии по стандартам в условиях жесткого давления приходится выбирать в качестве стандарта решение одной из конкурирующих фирм.

6.2. Создание, редактирование и удаление базы данных

В MySQL базы данных представляют собой подкаталоги каталога данных data. В каждой из баз данных могут быть созданы таблицы, которые в свою очередь разбиваются на столбцы и строки. Таким образом, базы данных можно рассматривать как каталоги, в которых хранятся таблицы, каждая из которых хранит данные в ячейках, образуемых столбцами и строками.

В табл. 6.1 представлены операторы управления базами данных.

Таблица 6.1. *Операторы управления базами данных*

Оператор	Описание
`CREATE DATABASE db;`	Создать базу данных с именем db
`DROP DATABASE db;`	Удалить базу данных с именем db

Таблица 6.1 *(окончание)*

Оператор	Описание
`SHOW DATABASES;`	Показать список баз данных
`USE db;`	Выбрать базу данных с именем db
`RENAME DATABASE db TO new;`	Переименовать базу данных с именем db в new
`ALTER DATABASE db;`	Изменить параметры базы данных db

ЗАМЕЧАНИЕ

Оператор `USE` не является полноценным оператором языка SQL, эта команда используется для выбора текущей базы данных в консольном клиенте mysql. В PHP-расширении для работы с базой данных MySQL для этих задач предназначена функция `mysql_select_db()`.

При создании базы данных при помощи оператора `CREATE DATABASE` можно указать кодировку по умолчанию при помощи ключевого слова `DEFAULT CHARACTER SET`. Данная кодировка будет использоваться в таблицах и текстовых столбцах базы данных в случае, если она не будет указана явно. В листинге 6.1 приводится пример создания базы данных wet с кодировкой по умолчанию Windows-1251.

ЗАМЕЧАНИЕ

Список кодировок, которые поддерживает MySQL, можно получить при помощи оператора `SHOW CHARACTER SET`.

Листинг 6.1. Создание базы данных с кодировкой по умолчанию Windows-1251

```
CREATE DATABASE wet DEFAULT CHARACTER SET cp1251;
```

Если кодировка базы данных не указана или в процессе проектирования базы данных принято решение изменить ее, можно воспользоваться оператором `ALTER DATABASE` (листинг 6.2).

Листинг 6.2. Изменение кодировки базы данных на UTF-8

```
ALTER DATABASE wet DEFAULT CHARACTER SET utf8;
```

Для переименования базы данных предназначен оператор `RENAME DATABASE`. В листинге 6.3 база данных `wet` переименовывается в `db`.

Листинг 6.3. Переименование базы данных

```
RENAME DATABASE wet TO db;
```

Для удаления таблицы предназначен оператор `DROP DATABASE`. Если удаляемая база данных отсутствует, возвращается ошибка, поэтому данный оператор часто снабжается ключевым словом `IF EXISTS`, которое задействует оператор только в том

случае, если база данных реально существует (листинг 6.4). Ключевое слово `IF EXISTS` приобретает большое значение для пакетных заданий, где ошибка выполнения одного оператора приводит к остановке выполнения всего пакета операторов.

Листинг 6.4. Удаление базы данных

```
DROP DATABASE IF EXISTS db;
```

Кстати, при создании базы данных при помощи оператора `CREATE DATABASE` можно использовать похожую функциональность, используя ключевое слово `IF NOT EXISTS`, которое позволяет задействовать оператор только в том случае, если базы данных с таким именем не существует (листинг 6.5).

Листинг 6.5. Создание базы данных только в случае ее отсутствия

```
CREATE DATABASE IF NOT EXISTS wet;
```

6.3. Создание, редактирование и удаление таблиц

После того как база данных создана, в ней можно создавать таблицы. Есть два пути для обращения к таблицам: либо используя расширенную запись, включающую имя базы данных `db.tbl`, либо просто обращаясь к имени таблицы `tbl`. Последнее возможно, если текущая база данных выбрана при помощи оператора `USE` или функции `mysql_select_db()`. Дальнейшее повествование будет вестись в предположении, что база данных по умолчанию выбрана.

В табл. 6.2 представлен список операторов управления таблицами.

***Таблица 6.2.** Операторы управления таблицами*

Оператор	Описание
`CREATE TABLE tbl;`	Создать таблицу с именем tbl
`DROP TABLE tbl;`	Удалить таблицу с именем tbl
`ALTER TABLE tbl;`	Изменить структуру таблицы tbl
`RENAME TABLE tbl TO new;`	Переименовать таблицу tbl в new
`DESCRIBE TABLE tbl;`	Вывести структуру таблицы tbl
`TRUNCATE TABLE tbl;`	Удалить все записи из таблицы tbl
`SHOW TABLES;`	Показать список таблиц текущей базы данных

В листинге 6.6 приводится пример создания простейшей таблицы `tbl`, содержащей два столбца: целочисленный столбец `id` и текстовый столбец `name`.

Листинг 6.6. Создание таблицы с двумя столбцами

```
CREATE TABLE tbl (
  id INT,
  name TEXT
);
```

Столбцы могут иметь разные типы, которые условно делятся на числовые, строковые и календарные. В табл. 6.3 приводится список числовых типов.

ЗАМЕЧАНИЕ

Целые типы данных могут быть объявлены положительными. Для этого после объявления типа следует использовать ключевое слово UNSIGNED. В этом случае элементам данного столбца нельзя будет присвоить отрицательные значения, а допустимый диапазон, который может принимать тип, удваивается.

Таблица 6.3. *Числовые типы столбцов*

Тип	Описание
TINYINT	От –128 до 127, от 0 до 255
SMALLINT	От –32 768 до 32 767, от 0 до 65535
MEDIUMINT	От –8 388 608 до 8 388 608, от 0 до 16 777 215
INT, INTEGER	От –2 147 683 648 до 2 147 683 648, от 0 до 4 294 967 295
BIGINT	От -2^{63} до $2^{63}-1$, от 0 до 2^{64}
BIT	От 1 до 64 битов
BOOL, BOOLEAN	Либо 0, либо 1
DECIMAL, DEC, NUMERIC	Число повышенной точности, диапазон задается параметрами: например, DECIMAL(5,2) отводит под число 5 символов с двумя знаками после запятой, следовательно, позволяет хранить цифры от –99,99 до 999,99
FLOAT	Минимальное значение $\pm 1{,}175494351 \times 10^{-39}$, максимальное значение $\pm 3{,}402823466 \times 10^{38}$
DOUBLE, REAL, DOUBLE PRECISION	Минимальное значение $\pm 2{,}2250738585072014 \times 10^{-308}$, максимальное значение $\pm 1{,}797693134862315 \times 10^{308}$

СУБД MySQL имеет пять целых типов: TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT. Различие между ними заключается в диапазоне величин, которые можно хранить в столбцах такого типа. Чем больше диапазон значений у типа данных, тем больше памяти для него требуется.

Тип DECIMAL и его синонимы NUMERIC и DEC предназначены для величин повышенной точности, например, денежных данных. Этот тип данных медленнее, чем остальные типы с плавающей точкой (FLOAT и DOUBLE), однако при вычислении в значениях типа DECIMAL не накапливаются ошибки вычисления, связанные с приблизительной природой модели чисел в FLOAT и DOUBLE.

ЗАМЕЧАНИЕ

При работе с СУБД MySQL часто можно наблюдать указание точности, например, `INT(11)`. Это значение не отражается на фактическом состоянии числа в таблице и предназначено для указания количества цифр, отводимых под число в консольных программах, вроде mysql.

В табл. 6.4 представлены строковые типы данных, традиционно к этому типу данных относится BLOB-тип, позволяющий хранить бинарные данные.

Таблица 6.4. *Строковые типы столбцов*

Тип	Описание
`CHAR`	Строка фиксированной длины, длина строки указывается в скобках, например, `CHAR(255)` могут хранить строки длиной до 255 символов (максимальное значение 65 535). Если в таблице присутствует хотя бы один столбец переменной длины, этот тип столбца автоматически превращается в `VARCHAR`
`VARCHAR`	Строка переменной длины, аналогичная `CHAR`, но занимающая столько места в базе данных, сколько символов в строке, плюс один дополнительный
`TINYBLOB`, `TINYTEXT`	Строка переменной длины, хранящая до 255 символов
`BLOB`, `TEXT`	Строка переменной длины, хранящая до 65 535 символов
`MEDIUMBLOB`, `MEDIUMTEXT`	Строка переменной длины, хранящая до 2^{24}–1 символов
`LONGBLOB`, `LONGTEXT`	Строка переменной длины, хранящая до 2^{32}–1 символов
`ENUM('value1','value2',...)`	Хранит одно строковое значение из указанного в круглых скобках набора
`SET('value1','value2',...)`	Хранит набор значений в круглых скобках: все или ни одного

Календарные типы столбцов предназначены для хранения даты и времени (табл. 6.5).

Таблица 6.5. *Календарные типы столбцов*

Тип	Описание
`DATE`	Предназначен для хранения даты, принимает значения от '1000-01-01' до '9999-12-31'
`TIME`	Предназначен для хранения времени, принимает значения от '–828:59:59' до '828:59:59'
`DATATIME`	Предназначен для хранения даты и времени, принимает значения от '1000-01-01 00:00:00' до '9999-12-31 00:00:00'
`TIMESTAMP`	Предназначен для хранения даты и времени, принимает значения от '1970-01-01 00:00:00' до '2038-12-31 59:59:59'. Столбы данного типа автоматически обновляются при выполнении операторов `INSERT` и `UPDATE`
`YEAR`	От 1901 до 2155 для `YEAR(4)`, от 1970 до 2069 для `YEAR(2)`

В конце оператора `CREATE TABLE`, после указания столбцов могут следовать параметры таблицы. Например, можно указать тип таблицы при помощи ключевого

слова ENGINE. В листинге 6.7 приводится пример создания MyISAM-таблицы comments, для которой при помощи ключевого слова DEFAULT CHARSET задается кодировка по умолчанию UTF-8.

> ***Замечание***
>
> Если для работы с MySQL необходима поддержка транзакций, следует использовать тип таблиц InnoDB. В том случае, если в операторе CREATE TABLE не указывается тип таблицы, устанавливается тип по умолчанию (это значение задается директивой default-storage-engine конфигурационного файла my.ini).

Листинг 6.7. Использование параметров таблицы

```
CREATE TABLE comments (
  id_position INT,
  name TINYTEXT,
  comment TEXT,
  putdate DATETIME,
  hide ENUM('show','hide')
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Для удаления таблиц предназначен оператор DROP TABLE, использование которого продемонстрировано в листинге 6.8. По аналогии с оператором DROP DATABASE данный оператор при помощи ключевого слова IF EXISTS позволяет удалять только реально существующие таблицы.

Листинг 6.8. Удаление таблицы

```
DROP TABLE IF EXISTS comments;
```

Для редактирования таблицы предназначен оператор ALTER TABLE, который позволяет вставлять и удалять столбцы, изменять их тип и порядок следования. В листинге 6.9 приводится пример добавления текстового столбца author после столбца id_position таблицы comments.

> ***Замечание***
>
> Если ключевое слово AFTER id_position не указано, столбец будет добавлен в конец таблицы. Для вставки нового столбца в начало таблицы используется ключевое слово FIRST.

Листинг 6.9. Вставка нового столбца в таблицу comments

```
ALTER TABLE comments ADD COLUMN author TEXT AFTER id_position;
```

Для просмотра структуры таблицы удобно воспользоваться оператором DESCRIBE, который возвращает в результирующей таблице описание полей запрошенной таблицы (листинг 6.10).

Листинг 6.10. Просмотр структуры таблицы

```
DESCRIBE comments;
+-------------+---------------------+------+-----+---------+-------+
| Field       | Type                | Null | Key | Default | Extra |
+-------------+---------------------+------+-----+---------+-------+
| id_position | int(11)             | NO   |     | NULL    |       |
| author      | text                | NO   |     | NULL    |       |
| name        | tinytext            | NO   |     | NULL    |       |
| comment     | text                | NO   |     | NULL    |       |
| putdate     | datetime            | NO   |     | NULL    |       |
| hide        | enum('show','hide') | NO   |     | NULL    |       |
+-------------+---------------------+------+-----+---------+-------+
```

Для удаления столбца из таблицы используются ключевые слова `DROP COLUMN` (листинг 6.11).

Листинг 6.11. Удаление столбца из таблицы `comments`

```
ALTER TABLE comments DROP COLUMN author;
```

Для изменения типа существующего столбца используется ключевое слово `MODIFY`. В листинге 6.12 тип столбца `id_position` меняется с `INT` на `BIGING`.

Листинг 6.12. Изменение типа столбца

```
ALTER TABLE comments MODIFY id_position BIGINT;
```

Для изменения имени столбца используется ключевое слово `CHANGE`, оно также позволяет задать новый тип. В листинге 6.13 столбец `id_position` переименовывается в `id_comment`.

Листинг 6.13. Изменение названия столбца

```
ALTER TABLE comments CHANGE id_position id_comment INT;
```

6.4. Вставка данных в таблицу. Оператор *INSERT*

Для вставки записи в таблицу используется оператор `INSERT INTO tbl VALUES()`. В скобках через запятую приводятся значения полей. При создании таблицы оператором `CREATE TABLE` для столбцов таблицы можно указать значения параметров по умолчанию.

Создадим таблицу `tbl`, состоящую из двух числовых столбцов, один из которых `id` по умолчанию будет иметь значение 5, а второй `id_cat` будет принимать по умолчанию значение `NULL` (листинг 6.14).

Листинг 6.14. Таблица `tbl`

```
CREATE TABLE tbl (
  id INT(11) NOT NULL DEFAULT '5',
  id_cat INT(11) DEFAULT NULL
);
```

Существует несколько вариантов использования оператора `INSERT`, каждый из которых приводит к вставке новой записи (листинг 6.15).

Листинг 6.15. Вставка записей при помощи оператора `INSERT`

```
INSERT INTO tbl VALUES (10, 20);
INSERT INTO tbl (id_cat, id) VALUES (10, 20);
INSERT INTO tbl (id) VALUES (30);
INSERT INTO tbl () VALUES ();
```

Если значение для столбца отсутствует, оператор `INSERT` подставит в него значение по умолчанию. Проверить результат вставки новых записей в таблицу можно при помощи оператора `SELECT` (листинг 6.16), синтаксис которого подробно разбирается в *разд. 6.6*.

Листинг 6.16. Просмотр содержимого таблицы

```
SELECT * FROM tbl;
+----+--------+
| id | id_cat |
+----+--------+
| 10 |     20 |
| 20 |     10 |
| 30 |   NULL |
|  5 |   NULL |
+----+--------+
```

При вставке строковых и календарных значений их необходимо заключать в двойные или одиночные кавычки. Оператор `INSERT` для таблицы `comments`, представленной в листинге 6.7, может выглядеть следующим образом (листинг 6.17).

Листинг 6.17. Вставка строковых значений

```
INSERT INTO comments
VALUES(1, 'Название', 'Содержимое комментария', NOW(), 'show');
```

Здесь использовалась MySQL-функция `NOW()`, которая возвращает текущее время в подходящем для вставки формате.

Оператор `INSERT` позволяет за один раз вставлять сразу несколько строк, для этого после ключевого слова `VALUES` добавляется несколько списков в круглых скобках, разделенных запятыми (листинг 6.18).

Листинг 6.18. Вставка нескольких строковых значений

```
INSERT INTO comments VALUES
(1, 'Сообщение', 'Содержимое комментария', NOW(), 'show'),
(2, 'Ответ', 'Еще один комментарий комментария', NOW(), 'show');
```

6.5. Вставка уникальных значений

Для записей в базах данных часто требуются уникальные значения, чтобы связывать записи из разных таблиц друг с другом, однозначно идентифицировать запись при отображении в приложении и т. д.

Для назначения таких уникальных в пределах таблицы значений в MySQL предусмотрен специальный механизм генерации. Для его подключения достаточно снабдить столбец признаком первичного ключа `PRIMARY KEY` с атрибутом `AUTO_INCREMENT`. Тогда при создании новой записи в качестве значения достаточно передать `NULL` или 0 — поле автоматически получит значение, равное максимальному значению столбца, плюс единица. В листинге 6.19 создается таблица `tbl`, состоящая из первичного ключа `id` и текстового поля `name`. Первичный ключ `id` снабжен атрибутом `AUTO_INCREMENT`.

Листинг 6.19. Действие механизма `AUTO_INCREMENT`

```
CREATE TABLE tbl (
  id INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT,
  PRIMARY KEY (id)
) DEFAULT CHARSET=cp1251;
INSERT INTO tbl VALUES (NULL, 'Процессоры');
INSERT INTO tbl VALUES (NULL, 'Материнские платы');
INSERT INTO tbl VALUES (NULL, 'Видеоадаптеры');
SELECT * FROM tbl;
+----+-------------------+
| id | name              |
+----+-------------------+
|  1 | Процессоры        |
|  2 | Материнские платы |
|  3 | Видеоадаптеры     |
+----+-------------------+
```

Часто в прикладных программах требуется узнать значение, присвоенное столбцу и снабженное атрибутом `AUTO_INCREMENT` (чтобы использовать сгенерированное зна-

чение первичного ключа в качестве значения в другой таблице и связать данные из разных таблиц). Только что сгенерированное значение возвращает встроенная функция MySQL `LAST_INSERT_ID()`, как это показано в листинге 6.20.

Замечание

В PHP получить значение автоматически назначенного первичного ключа можно при помощи функции `mysql_insert_id()`.

Листинг 6.20. Получение только что добавленного значения

```
SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                3 |
+------------------+
```

6.6. Извлечение данных. Оператор *SELECT*

Для извлечения данных и таблицы предназначен оператор `SELECT`. Простейший пример оператора был представлен в листинге 6.19. После ключевого слова `SELECT` следует символ `*`, который сообщает СУБД о необходимости вернуть все столбцы таблицы, после ключевого слова `FROM` указывается имя таблицы. При необходимости вернуть только часть столбцов или изменить их порядок вместо символа `*` следует указать список столбцов. Применительно к таблице `tbl`, создание которой описывается в листинге 6.19, полный оператор `SELECT`, с перечислением всех столбцов может выглядеть следующим образом (листинг 6.21).

Листинг 6.21. Использование оператора `SELECT`

```
SELECT id, name FROM tbl;
+----+------------------+
| id | name             |
+----+------------------+
|  1 | Процессоры       |
|  2 | Материнские платы |
|  3 | Видеоадаптеры    |
+----+------------------+
```

6.6.1. Переименование столбцов. Ключевое слово *AS*

Помимо названий столбцов в списке после ключевого слова `SELECT` могут находиться вычисляемые столбцы (арифметические выражения и вызовы функций или даже просто значения). В листинге 6.22 в `SELECT`-операторе к значению столбца `id`

прибавляется 100, а в качестве третьего столбца выступает функция NOW(), которая возвращает текущее время.

Листинг 6.22. Использование вычисляемых столбцов

```
SELECT id + 100, name, NOW() FROM tbl;
+----------+------------------+---------------------+
| id + 100 | name             | NOW()               |
+----------+------------------+---------------------+
|      101 | Процессоры       | 2011-06-18 13:26:46 |
|      102 | Материнские платы | 2011-06-18 13:26:46 |
|      103 | Видеоадаптеры    | 2011-06-18 13:26:46 |
+----------+------------------+---------------------+
```

Имена вычисляемых столбцов принимают значения выражений из SELECT-списка. Это может быть неудобно, поскольку имена столбцов могут выступать в качестве ключей ассоциативного массива в клиентской программе. Поэтому MySQL позволяет переименовать их, используя ключевое слово AS (листинг 6.23).

Листинг 6.23. Переименование столбцов

```
SELECT
  id + 100 AS id,
  name AS catalog,
  NOW() AS time
FROM tbl;
+-----+------------------+---------------------+
| id  | catalog          | time                |
+-----+------------------+---------------------+
| 101 | Процессоры       | 2011-06-18 13:35:16 |
| 102 | Материнские платы | 2011-06-18 13:35:16 |
| 103 | Видеоадаптеры    | 2011-06-18 13:35:16 |
+-----+------------------+---------------------+
```

6.6.2. Условная выборка. Ключевое слово *WHERE*

Ситуация, когда требуется изменить количество выводимых строк, встречается гораздо чаще, чем ситуация, когда требуется изменить число и порядок выводимых столбцов. Для ввода в SQL-запрос такого рода ограничений в операторе SELECT предназначено специальное ключевое слово WHERE, после которого следует логическое условие. Если запись удовлетворяет такому условию, она попадает в результат выборки, в противном случае такая запись отбрасывается.

В листинге 6.24 приводится пример выборки из таблицы tbl строк, значение поля id которых превышает 1.

Замечание

Операторы сравнения SQL совпадают с операторами сравнения PHP, за исключением того факта, что для оператора "не равно" != в SQL имеется синоним <>. Для обозначения логического И (ИЛИ) в SQL используются операторы AND (OR).

Листинг 6.24. Условная выборка

```
SELECT * FROM tbl
WHERE id > 1;
+----+-------------------+
| id | name              |
+----+-------------------+
|  2 | Материнские платы |
|  3 | Видеоадаптеры     |
+----+-------------------+
```

Условия могут состоять из простых условных выражений, связанных логическими операторами OR и AND. В листинге 6.25 приводится пример вывода строк со значением поля id больше 1 и количеством символов в поле name меньше 15.

Замечание

Для подсчета количества значений в поле name используется MySQL-функция LENGTH().

Листинг 6.25. Использование сложных условий выборки

```
SELECT * FROM tbl
WHERE
  id > 1 AND
  LENGTH(name) < 15;
+----+---------------+
| id | name          |
+----+---------------+
|  3 | Видеоадаптеры |
+----+---------------+
```

6.6.3. Сортировка записей. Ключевое слово *ORDER BY*

Одной из особенностей СУБД является тот факт, что записи в них хранятся в произвольном порядке. Несмотря на то, что примерах они могут выводиться именно в том порядке, в котором они были вставлены, такое положение дел не только не гарантируется, но и быстро нарушается при обновлении и удалении записей. Дело в том, что при удалении записей, чтобы ускорить эту довольно требовательную к ресурсам операцию, место, которое занималось записью, не возвращается в файловую систему, а остается свободным в таблице. Далее, при добавлении новых записей или обновлении уже существующих, это свободное место может быть использова-

но. Поэтому в таблицах записи перемешиваются, и полагаться на их естественный порядок не стоит. Для того чтобы вывести записи в требуемом порядке, используется конструкция `ORDER BY`. В листинге 6.26 приводится пример извлечения записей из таблицы `tbl` с сортировкой их по столбцу `name`.

> ***Замечание***
>
> Оператор `ALTER TABLE` позволяет физически отсортировать таблицу по одному из столбцов, для этого используется оператор вида `ALTER TABLE tbl ORDER BY id`.

Листинг 6.26. Сортировка записей при помощи конструкции `ORDER BY`

```
SELECT * FROM tbl
ORDER BY name;
+----+-------------------+
| id | name              |
+----+-------------------+
|  3 | Видеоадаптеры     |
|  2 | Материнские платы |
|  1 | Процессоры        |
+----+-------------------+
```

Если после имени столбца следует ключевое слово `DESC`, сортировка данных производится в обратном порядке.

6.6.4. Вывод записей в случайном порядке

Еще одной часто возникающей задачей является вывод записей в случайном порядке. Для этого конструкции `ORDER BY` вместо имени столбца передается функция `RAND()` (листинг 6.27).

Листинг 6.27. Вывод записей в случайном порядке

```
SELECT * FROM tbl
ORDER BY RAND();
+----+-------------------+
| id | name              |
+----+-------------------+
|  2 | Материнские платы |
|  3 | Видеоадаптеры     |
|  1 | Процессоры        |
+----+-------------------+
```

6.6.5. Ограничение выборки. Ключевое слово *LIMIT*

Результат выборки может содержать сотни и тысячи записей. Их вывод и обработка занимают значительное время и серьезно загружают сервер базы данных, поэтому

информацию часто разбивают на страницы и предоставляют ее пользователю порциями. Извлечение только части запроса требует меньше времени и вычислений, кроме того, пользователю часто бывает достаточно просмотреть первые несколько записей. Для ограничения выборки в операторе SELECT служит ключевое слово LIMIT. В листинге 6.28 приводится пример выборки из таблицы tbl первых двух записей.

Листинг 6.28. Ограничение количества выбираемых записей при помощи LIMIT

```
SELECT * FROM tbl
ORDER BY id
LIMIT 2;
+----+------------------+
| id | name             |
+----+------------------+
|  1 | Процессоры       |
|  2 | Материнские платы |
+----+------------------+
```

Ключевое слово LIMIT имеет альтернативную запись, которая включает два числа, разделенных запятой. В этом случае первое число указывает номер записи, начиная с которой следует производить выборку, а второе число задает количество выбираемых записей (листинг 6.28).

Листинг 6.29. Альтернативная форма записи ключевого слова LIMIT

```
SELECT * FROM tbl
ORDER BY id
LIMIT 2, 2;
+----+---------------+
| id | name          |
+----+---------------+
|  3 | Видеоадаптеры |
+----+---------------+
```

6.7. Обновление данных. Оператор *UPDATE*

Операция обновления позволяет менять значения полей в уже существующих записях. В листинге 6.30 приводится пример обновления поля name таблицы tbl для строки, чье значение поле id равняется 1.

Замечание

Если в запросе из листинга 6.30 не использовать WHERE-условие, новое значение получат все поля name таблицы tbl.

Листинг 6.30. Обновление поля таблицы при помощи оператора UPDATE

```
UPDATE tbl SET name = 'Процессоры1'
WHERE id = 1;
SELECT * FROM tbl;
+----+------------------+
| id | name             |
+----+------------------+
|  1 | Процессоры1      |
|  2 | Материнские платы |
|  3 | Видеоадаптеры    |
+----+------------------+
```

Один оператор UPDATE может изменять значения в нескольких столбцах одновременно. Для этого достаточно перечислить их через запятую после ключевого слова SET (листинг 6.31).

Замечание

В операторе UPDATE допускается использование ключевых слов ORDER BY и LIMIT для ограничения количества обновляемых записей.

Листинг 6.31. Обновление нескольких столбцов при помощи оператора UPDATE

```
UPDATE tbl
SET
  name = 'Процессоры',
  id = 100
WHERE id = 1;
SELECT * FROM tbl;
+-----+------------------+
| id  | name             |
+-----+------------------+
| 100 | Процессоры       |
|   2 | Материнские платы |
|   3 | Видеоадаптеры    |
+-----+------------------+
```

6.8. Удаление записей. Оператор *DELETE*

Для удаления записей из таблицы предназначен оператор DELETE. В листинге 6.32 удаляется запись со значением 3 поля id.

Замечание

В операторе DELETE допускается использование ключевых слов ORDER BY и LIMIT для ограничения количества обновляемых записей.

Листинг 6.32. Удаление записей при помощи оператора `DELETE`

```
DELETE FROM tbl
WHERE id = 3;
SELECT * FROM tbl;
+-----+------------------+
| id  | name             |
+-----+------------------+
| 100 | Процессоры       |
|   2 | Материнские платы |
+-----+------------------+
```

ГЛАВА 7

Сложные вопросы MySQL

СУБД MySQL является довольно сложным продуктом, который зачастую позволяет решать задачи управления данными быстрее и эффективнее PHP.

7.1. Индексы и оценка производительности

Индексы играют большую роль в базах данных, т. к. это основной способ ускорения их работы. Обычно записи в таблице располагаются в хаотическом порядке. Для того чтобы найти нужную запись, необходимо сканировать всю таблицу, на что уходит большое количество времени. Идея индексов состоит в том, чтобы создать для столбца копию, которая постоянно будет поддерживаться в отсортированном состоянии. Это позволяет очень быстро осуществлять поиск по такому столбцу, т. к. заранее известно, где необходимо искать значение.

В MySQL различают четыре вида индексов:

- *первичный ключ* — главный индекс таблицы. В таблице может быть только один первичный ключ, и все значения такого индекса должны отличаться друг от друга, т. е. являться уникальными;
- *обычный индекс* — таких индексов может быть несколько;
- *уникальный индекс* — уникальных индексов также может быть несколько, но значения индекса не должны повторяться;
- *полнотекстовый индекс* — специальный вид индекса для столбцов типа `TEXT`, позволяющий производить полнотекстовый поиск.

Индексы могут иметь как собственные имена, так и имена индексируемых столбцов. Один индекс может охватывать один или несколько столбцов, причем тип столбца может быть любым, за исключением `ENUM` и `SET`. Один столбец может участвовать в нескольких индексах. При построении индекса на нескольких столбцах следует помнить, что индекс может влиять на запросы, в которых участвуют либо все столбцы, либо первые столбцы/столбец. Индекс на трех столбцах `id_position`, `id_catalog`, `number` будет ускорять выборку для следующих запросов:

```
SELECT * FROM tbl
WHERE id_position = 3 AND id_catalog = 2 AND number > 15;
SELECT * FROM tbl
WHERE id_position = 3 AND id_catalog = 2;
SELECT * FROM tbl
WHERE id_position = 3;
```

Однако если потребуется ускорить следующий запрос, нужно будет построить дополнительный индекс:

```
SELECT * FROM tbl
WHERE id_catalog = 2 AND number > 15;
```

Создавая индексы, необходимо иметь в виду, что они ускоряют `SELECT`-запросы, а запросы `INSERT`, `DELETE` и `UPDATE` замедляются (необходимы дополнительные усилия на перестроение индекса).

Первичный ключ является главным индексом таблицы и объявляется при помощи ключевого слова `PRIMARY KEY`, у таблицы может быть только один первичный ключ. Значение первичного ключа должно быть уникально и не повторяться в пределах таблицы. Кроме того, столбцы, помеченные атрибутом `PRIMARY KEY`, не могут принимать значение `NULL`. Для пометки поля таблицы в качестве первичного ключа достаточно поместить это ключевое слово `PRIMARY KEY` в определение столбца. В листинге 7.1 приведено определение таблицы `tbl`, где в качестве первичного ключа назначен столбец `id`.

> ***Замечание***
>
> Первичный ключ не является обязательной конструкцией, допускается создание таблиц, не содержащих первичных ключей и индексов.

Листинг 7.1. Создание первичного ключа

```
CREATE TABLE tbl (
  id INT(11) NOT NULL PRIMARY KEY,
  name TINYTEXT NOT NULL
)
```

Существует альтернативный способ объявления первичного ключа, представленный в листинге 7.2.

Листинг 7.2. Альтернативный способ объявления первичного ключа

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  name TINYTEXT NOT NULL,
  PRIMARY KEY(id)
)
```

В отличие от первичного ключа таблица может содержать несколько обычных и уникальных индексов. Для того чтобы их различать, индексы могут иметь собственные имена. Часто имена индексов совпадают с именами столбцов, которые они индексируют, но для индекса можно назначить и совершенно другое имя. Объявление индекса производится при помощи ключевого слова `INDEX` или `KEY`. Для уникальных индексов, которые не позволяют добавлять в таблицу записи с уже существующими значениями, вводится дополнительное ключевое слово `UNIQUE`, т. е. пишется `UNIQUE INDEX` и `UNIQUE KEY`.

В листинге 7.3 приводится оператор `CREATE TABLE`, создающий таблицу `tbl`, снабженную первичным ключом `id` и двумя индексами `id_tbl` и `id_cat`.

Листинг 7.3. Создание таблицы, снабженной несколькими индексами

```
CREATE TABLE tbl (
  id INT(11) NOT NULL AUTO_INCREMENT,
  id_tbl INT(11) NOT NULL DEFAULT '0',
  id_cat INT(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (id),
  KEY id_tbl (id_tbl),
  KEY id_cat (id_cat)
);
```

Имена индексов совпадают с именами индексируемых столбцов, но вполне допустимо присвоить им другие имена, например:

```
KEY first (id),
KEY second (id_cat)
```

> ***Замечание***
>
> В отличие от первичного ключа, для обычных индексов недопустимо их определение вместе со столбцом. Индекс объявляется отдельно в конце таблицы при помощи ключевого слова `INDEX` или `KEY`.

7.2. Кодировки

СУБД MySQL поддерживает несколько кодировок, одна кодировка может поддерживать несколько языков. Поэтому каждой кодировке обычно указывается сопоставление — порядок следования символов друг за другом, позволяющий сортировать записи по алфавиту.

Для вывода всех поддерживаемых кодировок используется оператор `SHOW CHARACTER SET` (листинг 7.4).

Листинг 7.4. Вывод списка поддерживаемых кодировок

```
SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
...
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
...
| utf8     | UTF-8 Unicode               | utf8_general_ci     |      3 |
| cp866    | DOS Russian                 | cp866_general_ci    |      1 |
| cp1251   | Windows Cyrillic            | cp1251_general_ci   |      1 |
...
| eucjpms  | UJIS for Windows Japanese   | eucjpms_japanese_ci |      3 |
+----------+-----------------------------+---------------------+--------+
```

Первый столбец результирующей таблицы выводит название кодировки, второй — описание, третий — сопоставление, которое используется по умолчанию, четвертый столбец сообщает о максимальном количестве байтов, которое отводится под один символ кодировки.

При помощи оператора `SHOW COLLATION` можно определить полный список сопоставлений. По умолчанию оператор выводит большое количество информации, поэтому уменьшим его при помощи ключевого слова `LIKE`, ограничив вывод только кодировкой `cp1251`, которая соответствует русской Windows-кодировке (листинг 7.5).

Листинг 7.5. Сортировки, применяемые совместно с кодировкой Windows-1251

```
mysql> SHOW COLLATION LIKE 'cp1251%';
+--------------------+---------+----+---------+----------+---------+
| Collation          | Charset | Id | Default | Compiled | Sortlen |
+--------------------+---------+----+---------+----------+---------+
| cp1251_bulgarian_ci | cp1251  | 14 |         |          |       0 |
| cp1251_ukrainian_ci | cp1251  | 23 |         |          |       0 |
| cp1251_bin          | cp1251  | 50 |         |          |       0 |
| cp1251_general_ci   | cp1251  | 51 | Yes     |          |       0 |
| cp1251_general_cs   | cp1251  | 52 |         |          |       0 |
+--------------------+---------+----+---------+----------+---------+
```

Как видно из листинга 7.5, кодировка `cp1251` обладает пятью сопоставлениями, которые имеют следующие значения:

- `cp1251_bulgarian_ci` — болгарский язык (регистронезависимый);
- `cp1251_ukrainian_ci` — украинский язык (регистронезависимый);
- `cp1251_bin` — бинарная сортировка, сравнение по ASCII-кодам символов;

- `cp1251_general_ci` — многоязыковая сортировка (регистронезависимая);
- `cp1251_general_cs` — многоязыковая сортировка (зависимая от регистра).

Последние два сопоставления можно использовать для корректной работы с русским языком. В четвертом столбце результирующей таблицы оператора `SHOW COLLATION` отметкой `Yes` указываются сопоставления, использующиеся в кодировке, по умолчанию (см. листинг 7.5). Так если указывается только одна кодировка `cp1251`, без уточнения сопоставления, то будет выбрано сопоставление `cp1251_general_ci`. Это означает, что если в таблице (базе данных, столбце) планируется хранить русский текст в кодировке `cp1251`, то вполне можно использовать оператор `CREATE TABLE`, представленный в листинге 7.6.

Листинг 7.6. Создание таблицы, для хранения русского текста в кодировке `cp1251`

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL
) ENGINE=MyISAM CHARACTER SET cp1251;
```

Если в таблице планируется хранить текст на украинском языке, потребуется явно указать сортировку, как показано в листинге 7.7, т. к. в противном случае будет выставлена кодировка по умолчанию.

Листинг 7.7. Создание таблицы для хранения украинского текста в кодировке `cp1251`

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL
) ENGINE=MyISAM CHARACTER SET cp1251 COLLATION cp1251_ukrainian_ci;
```

Сразу после установки, если не проводились дополнительные настройки сервера, на уровне сервера устанавливается кодировка `latin1` со шведским сопоставлением `latin1_swedish_ci`. Это означает, что при создании баз данных и таблиц, если явно не указывать кодировку и сортировку при помощи ключевых слов `CHARACTER SET` и `COLLATION`, то по умолчанию будет выставляться кодировка `latin1` со шведским сопоставлением. Для того чтобы переопределить кодировку на уровне сервера, необходимо при запуске серверу передать параметр `--default-character-set` с указанием кодировки. При необходимости при помощи параметра `--default-collation` можно указать сортировку. В листинге 7.8 приводится пример запуска сервера с кодировкой `cp1251` с сортировкой по умолчанию (для русского текста) и с сортировкой `cp1251_ukrainian_ci` (для украинского текста).

Листинг 7.8. Запуск MySQL-сервера с кодировкой `cp1251` по умолчанию

```
mysqld --default-character-set=cp1251
mysqld --default-character-set=cp1251
       --default-collation=cp1251_ukrainian_ci
```

Можно избежать передачи параметров всякий раз при запуске сервера. Для этого их можно прописать в конфигурационном файле `my.cnf` (`my.ini`) в секции `[mysqld]` (листинг 7.9).

Листинг 7.9. Установка кодировки и сортировки в конфигурационном файле my.ini

```
[mysqld]
default-character-set=cp1251
default-collation=cp1251_general_ci
```

При указании сопоставления следует следить, чтобы оно соответствовало кодировке. Если совместно с кодировкой будет указано сопоставление из другого кодового набора, север не запустится.

Помимо серверного уровня настройки кодировок существует уровень соединения (клиента с сервером). При установке соединения клиента с сервером последний ожидает от клиента запросы в кодировке `character_set_client`. После того как запрос получен, сервер транслирует его в кодировку, заданную в системной переменной `character_set_connection`. Сортировка задается при помощи системной переменной `collation_connection`.

Переменная `character_set_results` задает кодировку, в которой сервер возвращает результаты клиенту. Если клиент (или клиентский код) запрашивает результирующие таблицы или сам вставляет записи с русским текстом, то перед работой необходимо выставить системные переменные так, как это представлено в листинге 7.10.

Листинг 7.10. Установка системных переменных для работы с русским текстом

```
SET character_set_client='cp1251';
SET character_set_results='cp1251';
SET character_set_connection='cp1251';
```

Вместо трех операторов, представленных в листинге 7.10, можно воспользоваться специальным оператором `SET NAMES 'имя_кодировки'` (листинг 7.11).

Листинг 7.11. Использование оператора SET NAMES

```
SET NAMES 'cp1251';
```

7.3. Функции MySQL

Рассмотрим часто используемые функции MySQL.

7.3.1. Версия MySQL

Получить текущую версию MySQL-сервера можно при помощи информационной функции `VERSION()` (листинг 7.12).

Листинг 7.12. Получение версии сервера

```
SELECT VERSION();
+------------------+
| VERSION()        |
+------------------+
| 5.1.56-community |
+------------------+
```

7.3.2. Количество записей в таблице

Подсчет количества записей в таблице осуществляется при помощи функции `COUNT()` со следующим синтаксисом:

```
COUNT(expr)
COUNT(*)
COUNT(DISTINCT expr1, expr2, ...)
```

Первая форма возвращает количество записей в таблице, поле *expr* которых не равно `NULL`. В листинге 7.13 представлен дамп таблицы `tbl`.

Листинг 7.13. Дамп таблицы `tbl`

```
CREATE TABLE tbl (
  id int(11) NOT NULL,
  value int(11) default NULL
);
INSERT INTO tbl VALUES (1, 230);
INSERT INTO tbl VALUES (2, NULL);
INSERT INTO tbl VALUES (3, 405);
INSERT INTO tbl VALUES (4, NULL);
```

В листинге 7.14 представлен запрос, возвращающий количество записей в таблице `tbl`.

Листинг 7.14. Использование функции `COUNT()`

```
SELECT * FROM tbl;
+----+-------+
| id | value |
+----+-------+
|  1 |   230 |
|  2 |  NULL |
|  3 |   405 |
|  4 |  NULL |
+----+-------+
SELECT COUNT(id), COUNT(value) FROM tbl;
```

```
+-----------+--------------+
| COUNT(id) | COUNT(value) |
+-----------+--------------+
|         4 |            2 |
+-----------+--------------+
```

Как видно из листинга 7.14, для полей `id` и `value` возвращаются различные значения. Это связано с тем, что количество `NULL`-полей в столбцах различается.

Форма функции `COUNT(*)` возвращает общее количество строк в таблице, независимо от того, принимает какое-либо поле значение `NULL` или нет (листинг 7.15). Запись учитывается в результате, даже если все ее поля равны `NULL`.

Листинг 7.15. Использование функции `COUNT(*)`

```
SELECT COUNT(*) FROM tbl;
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
```

Функция `COUNT(*)` оптимизирована для очень быстрого возврата результата при условии, что команда `SELECT` извлекает данные из одной таблицы, никакие другие столбцы не обрабатываются и запрос не содержит условия `WHERE`.

7.3.3. Максимальное и минимальное значение в таблице

Для поиска максимального и минимального значения в столбце предназначены функции `MIN()` и `MAX()`(листинг 7.16).

Замечание

В листинге 7.16 используется таблица, определение которой дано в листинге 7.13.

Листинг 7.16. Использование функций `MIN()` и `MAX()`

```
SELECT
  MIN(id) AS min,
  MAX(id) AS max
FROM tbl;
+------+------+
| min  | max  |
+------+------+
|    1 |    4 |
+------+------+
```

Однако использование функций MIN() и MAX() в выражении WHERE приведет к ошибке. В листинге 7.17 показана попытка извлечения записи из таблицы tbl с максимальным значением поля id.

Листинг 7.17. Использование функций MIN() и MAX() в WHERE недопустимо

```
SELECT * FROM tbl WHERE id = MAX(id);
ERROR 1111: Invalid use of group function
```

Решение поставленной ранее задачи следует искать с привлечением выражения ORDER BY. В листинге 7.18 первый запрос извлекает запись с наименьшим значением поля id, а второй — с наибольшим.

Листинг 7.18. Использование функций MIN() и MAX() совместно с ORDER BY

```
SELECT * FROM tbl ORDER BY id LIMIT 1;
+----+-------+
| id | value |
+----+-------+
|  1 | 230   |
+----+-------+
SELECT * FROM tbl ORDER BY id DESC LIMIT 1;
+----+-------+
| id | value |
+----+-------+
|  5 | NULL  |
+----+-------+
```

7.3.4. Сумма значений столбца

Сумму элементов столбца позволяет подсчитать функция SUM() (листинг 7.19).

Листинг 7.19. Подсчет суммы значений столбца

```
SELECT SUM(id) FROM tbl;
+---------+
| SUM(id) |
+---------+
|     102 |
+---------+
```

7.3.5. Форматирование даты

Функция DATE_FORMAT(*date*, *format*) форматирует время *date* в соответствии со строкой *format*. В строке *format* могут использоваться определители, представленные в табл. 7.1.

Таблица 7.1. *Определители строки форматирования функции* `DATE_FORMAT()`

Определитель	Описание
`%a`	Сокращенное наименование дня недели (Sun–Sat)
`%b`	Сокращенное наименование месяца (Jan–Dec)
`%c`	Месяц в числовой форме (1–12)
`%D`	День месяца с английским суффиксом (1st, 2nd, 3rd и т. д.)
`%d`	День месяца в числовой форме с ведущим нулем (01–31)
`%e`	День месяца в числовой форме (1–31)
`%f`	Микросекунды (000000–999999)
`%H`	Час с ведущим нулем (00–23)
`%h`	Час с ведущим нулем (01–12)
`%I`	Час с ведущим нулем (01–12)
`%i`	Минуты с ведущим нулем (00–59)
`%j`	День года (001–366)
`%k`	Час с ведущим нулем (0–23)
`%l`	Час без ведущего нуля (1–12)
`%M`	Название месяца (January–December)
`%m`	Месяц в числовой форме с ведущим нулем (01–12)
`%p`	AM или PM (для 12-часового формата)
`%r`	Время, 12-часовой формат (hh:mm:ss AM или hh:mm:ss `PM`)
`%S`	Секунды (00–59)
`%s`	Секунды (00–59)
`%T`	Время, 24-часовой формат (hh:mm:ss)
`%U`	Неделя (00–52), где воскресенье считается первым днем недели
`%u`	Неделя (00–52), где понедельник считается первым днем недели
`%V`	Неделя (01–53), где воскресенье считается первым днем недели. Используется с `%X`
`%v`	Неделя (01–53), где понедельник считается первым днем недели. Используется с `%x`
`%W`	Название дня недели (Sunday–Saturday)
`%w`	День недели (0–6), где 0 — воскресенье
`%X`	Год, в который началась неделя, где воскресенье считается первым днем недели, 4 разряда, используется с `%V`
`%x`	Год, в который началась неделя, где воскресенье считается первым днем недели, число, 4 разряда, используется с `%v`
`%Y`	Год, 4 разряда (YYYY)
`%y`	Год, 2 разряда (YY)
`%%`	Литерал `%`

Все другие символы, которые не указаны в табл. 7.1, выводятся без изменений.

Рассмотрим наиболее часто встречающуюся задачу преобразования календарного значения из MySQL-формата '2011-01-12 11:55:17' в привычный формат '12.01.2011 11:55'. Осуществить такое преобразование можно при помощи запроса, представленного в листинге 7.20.

Листинг 7.20. Преобразование даты при помощи функции DATE_FORMAT()

```
SELECT DATE_FORMAT(putdatetime,'%d.%m.%Y %H:%i') AS putdatetime FROM tbl;
+------------------+
| putdatetime      |
+------------------+
| 12.01.2011 11:55 |
+------------------+
```

7.3.6. Вычисление возраста человека

Пусть имеется таблица tbl, содержащая сведения о сотрудниках и состоящая из трех столбцов:

- id — первичный ключ таблицы;
- fio — фамилия, имя, отчество сотрудника;
- putdate — дата рождения.

В листинге 7.21 представлен запрос CREATE TABLE, создающий таблицу tbl.

Листинг 7.21. Создание таблицы tbl

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  fio TINYTEXT NOT NULL,
  putdate DATE NOT NULL
);
INSERT INTO tbl VALUES (1, 'Тимирязев В.К.', '1972-12-02');
INSERT INTO tbl VALUES (2, 'Малышев Ю.Г.', '1976-06-24');
INSERT INTO tbl VALUES (3, 'Абрамов К.Т.', '1968-10-24');
INSERT INTO tbl VALUES (4, 'Ганюшкин В.В.', '1986-12-07');
```

Необходимо вычислить возраст сотрудника на текущую дату. Один из вариантов состоит в преобразовании даты рождения и текущей даты в дни при помощи функции TO_DAYS() и делению на число 365,25 (листинг 7.22): в году 365 дней; дробное число 0,25 призвано компенсировать високосные года, которые случаются раз в четыре года.

Листинг 7.22. Вычисление возраста сотрудников

```
SELECT
  fio,
  (TO_DAYS(NOW()) - TO_DAYS(putdate))/365.25 AS putdate
```

```
FROM tbl;
+----------------+---------+
| fio      | putdate |
+----------------+---------+
| Тимирязев В.К. | 38.5462 |
| Малышев Ю.Г.   | 34.9870 |
| Абрамов К.Т.   | 42.6530 |
| Ганюшкин В.В. | 24.5339 |
+----------------+---------+
```

Для того чтобы избавиться от дробной части, можно воспользоваться функцией `FLOOR()` (листинг 7.23).

Листинг 7.23. Отбрасывание дробной части

```
SELECT
  fio,
  FLOOR((TO_DAYS(NOW()) — TO_DAYS(putdate))/365.25) AS putdate
FROM tbl;
+----------------+---------+
| fio            | putdate |
+----------------+---------+
| Тимирязев В.К. |      38 |
| Малышев Ю.Г.   |      34 |
| Абрамов К.Т.   |      42 |
| Ганюшкин В.В.  |      24 |
+----------------+---------+
```

7.3.7. Преобразование IP-адреса в число

Функция `INET_ATON(address)` принимает IP-адрес *address* и представляет его в виде целого числа (листинг 7.24). Для числа `XXX.YYY.ZZZ.WWW` результат функции вычисляется по формуле:

$$XXX \times 256^3 + YYY \times 256^2 + ZZZ \times 256 + WWW.$$

ЗАМЕЧАНИЕ

Для того чтобы можно было поместить в целочисленное поле весь диапазон IP-адресов, следует использовать тип `BIGINT`.

Листинг 7.24. Преобразование IP-адреса в число при помощи функции `INET_ATON()`

```
SELECT INET_ATON('62.145.69.10'), INET_ATON('127.0.0.1');
+--------------------------+-----------------------+
| INET_ATON('62.145.69.10') | INET_ATON('127.0.0.1') |
+--------------------------+-----------------------+
|                1049707786 |             2130706433 |
+--------------------------+-----------------------+
```

Функция INET_ATON() способна принимать IP-адреса в сокращенной форме, как показано в листинге 7.25.

Листинг 7.25. Работа с сокращенной формой IP-адреса

```
SELECT INET_ATON('127.0.01'), INET_ATON('127.1');
+----------------------+--------------------+
| INET_ATON('127.0.01') | INET_ATON('127.1') |
+----------------------+--------------------+
|            2130706433 |         2130706433 |
+----------------------+--------------------+
```

Функция INET_NTOA(*address*) принимает IP-адрес в виде числа (результат выполнения функции INET_ATON()) и возвращает адрес в виде строки, состоящей из четырех чисел, разделенных точкой (листинг 7.26).

Листинг 7.26. Преобразование числа в IP-адрес при помощи функции INET_NTOA()

```
SELECT INET_NTOA(1049707786), INET_NTOA(2130706433);
+----------------------+----------------------+
| INET_NTOA(1049707786) | INET_NTOA(2130706433) |
+----------------------+----------------------+
| 62.145.69.10          | 127.0.0.1            |
+----------------------+----------------------+
```

7.4. Получение уникальных значений

Очень часто встает задача выбора из таблицы уникальных значений. Таблица products содержит поле id_catalog, предназначенное для связи с другой таблицей catalogs. Пусть требуется вывести все значения поля id_catalog; для этого можно воспользоваться запросом, представленным в листинге 7.27.

Листинг 7.27. Выборка значений поля id_catalog из таблицы products

```
SELECT id_catalog FROM products ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
|          1 |
|          1 |
|          2 |
|          2 |
|          3 |
|          4 |
|          5 |
|          5 |
+------------+
```

Как видно из листинга 7.27, результат не совсем удобен для восприятия. Было бы лучше, если бы запрос вернул уникальные значения столбца id_catalog. Для этого перед именем столбца можно использовать ключевое слово DISTINCT (листинг 7.28).

Листинг 7.28. Выборка уникальных значений

```
SELECT DISTINCT id_catalog FROM products ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
```

Как показано в листинге 7.28, результат запроса не содержит ни одного повторяющегося значения.

Для извлечения уникальных записей прибегают также к конструкции GROUP BY, содержащей имя столбца, по которому группируется результат (листинг 7.29).

Замечание

Конструкция GROUP BY располагается в SELECT-запросе перед конструкциями ORDER BY и LIMIT.

Листинг 7.29. Использование конструкции GROUP BY

```
SELECT id_catalog
FROM products
GROUP BY id_catalog
ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
```

7.5. Вложенные запросы

Вложенный запрос — это запрос, который является составной частью другого запроса. В листинге 7.30 представлены две таблицы — catalogs и products, каждая запись таблицы products содержит внешний ключ id_catalog, который устанавливает связь между таблицами так, как это представлено на рис. 7.1.

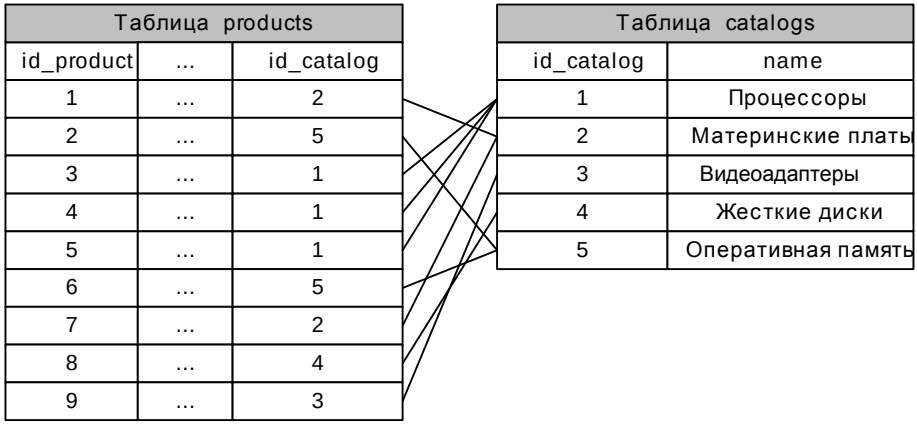


Таблица products		
id_product	...	id_catalog
1	...	2
2	...	5
3	...	1
4	...	1
5	...	1
6	...	5
7	...	2
8	...	4
9	...	3

Таблица catalogs	
id_catalog	name
1	Процессоры
2	Материнские платы
3	Видеоадаптеры
4	Жесткие диски
5	Оперативная память

Рис. 7.1. Связанные друг с другом таблицы

Листинг 7.30. Таблицы `catalogs` и `products`

```
CREATE TABLE catalogs (
  id_catalog int(11) NOT NULL auto_increment,
  name tinytext NOT NULL,
  PRIMARY KEY  (id_catalog)
);

INSERT INTO catalogs VALUES (1, 'Процессоры'), (2, 'Материнские платы'),
(3, 'Видеоадаптеры'), (4, 'Жесткие диски'), (5, 'Оперативная память');

CREATE TABLE products (
  id_product int(11) NOT NULL auto_increment,
  name tinytext NOT NULL,
  id_catalog int(11) NOT NULL,
  PRIMARY KEY  (id_product),
  KEY id_catalog (id_catalog)
);

INSERT INTO products VALUES (1, 'Материнская плата N 1', 2),
(2, 'Оперативная память N 1', 5), (3, 'Процессор N 1', 1),
(4, 'Процессор N 2', 1), (5, 'Процессор N 3', 1),
(6, 'Оперативная память N 2', 5), (7, 'Материнская плата N 2', 2),
(8, 'Жесткий диск N 1', 4), (9, 'Видеоадаптер N 1', 3);
```

Пусть необходимо вывести все позиции из таблицы `products`, которые соответствуют элементу каталога "Процессоры". Решить данную проблему можно при помощи вложенного запроса, представленного в листинге 7.31.

Листинг 7.31. Применение вложенного запроса

```
SELECT * FROM products
WHERE id_catalog = (SELECT id_catalog FROM catalogs
                    WHERE name = 'Процессоры');
+------------+---------------+------------+
| id_product | name          | id_catalog |
+------------+---------------+------------+
|          3 | Процессор N 1 |          1 |
|          4 | Процессор N 2 |          1 |
|          5 | Процессор N 3 |          1 |
+------------+---------------+------------+
```

Как видно из листинга 7.31, здесь в качестве внешнего запроса выступает выражение:

```
SELECT * FROM products
```

В качестве вложенного используется запрос:

```
SELECT id_catalog FROM catalogs WHERE name = 'Процессоры'
```

Таким образом, результат вложенного запроса (`id_catalog`), минуя промежуточные переменные в MySQL или внешней программе, применяется в качестве элемента внешнего запроса. Вложенный запрос помещается в круглые скобки.

Степень вложения запросов не ограничена вторым уровнем, могут использоваться трехуровневые и многоуровневые вложенные запросы. В листинге 7.32 извлекается название каталога из таблицы `catalogs`, которому принадлежит последняя позиция в таблице `products`.

Листинг 7.32. Каталог, которому принадлежит последняя позиция

```
SELECT name FROM catalogs
WHERE id_catalog = (SELECT id_catalog FROM products
                    WHERE id_product = (SELECT MAX(id_product)
                                        FROM products));
+---------------+
| name          |
+---------------+
| Видеоадаптеры |
+---------------+
```

Сначала извлекается последний первичный ключ `id_product` из таблицы `products`:

```
SELECT MAX(id_product) FROM products
```

Затем извлекается вторичный ключ `id_catalog`, соответствующий полученному значению `id_product` первого вложенного запроса:

```
SELECT id_catalog FROM products
WHERE id_product = (SELECT MAX(id_product)
                    FROM products)
```

Полученное значение id_catalog используется для извлечения названия во внешнем запросе (см. листинг 7.32).

> ***Замечание***
>
> На практике редко прибегают к вложенным запросам со степенью вложения больше трех, т. к. высокая степень вложенности быстро приводит к увеличению времени выполнения запроса.

Вложенный запрос, возвращающий единственное значение, просто подставляет результат на место своего выполнения (листинг 7.33).

Листинг 7.33. Использование вложенного запроса в списке столбцов

```
SELECT id_catalog, (SELECT MAX(id_product) FROM products)
FROM catalogs;
+------------+----------------------------------------+
| id_catalog | (SELECT MAX(id_product) FROM products) |
+------------+----------------------------------------+
|          1 |                                      9 |
|          2 |                                      9 |
|          3 |                                      9 |
|          4 |                                      9 |
|          5 |                                      9 |
+------------+----------------------------------------+
```

В листинге 7.33 вложенный запрос SELECT MAX(id_product) FROM products возвращает максимальное значение столбца id_product (9) из таблицы products и подставляет это значение на место запроса.

Наиболее часто вложенные запросы применяются в операциях сравнения в условиях, которые задаются ключевыми словами WHERE, HAVING или ON. Для этого задействуются шесть операторов сравнения (=, <>, <, <=, >, >=). Вложенный запрос, участвующий в операции сравнения, должен возвращать в качестве результата единичное значение. В листинге 7.34 приводится пример запроса, который выводит список элементов каталога, чей первичный ключ больше, чем первичный ключ каталога, которому принадлежит позиция с ключом id_product, равным пяти.

Листинг 7.34. Использование вложенных запросов в операциях сравнения

```
SELECT name FROM catalogs
WHERE id_catalog > (SELECT id_catalog FROM products
                    WHERE id_product = 5);
+--------------------+
| name               |
+--------------------+
| Материнские платы  |
| Видеоадаптеры      |
| Жесткие диски      |
| Оперативная память |
+--------------------+
```

```
SELECT name FROM catalogs
WHERE (SELECT id_catalog FROM products
       WHERE id_product = 5) < id_catalog;
+--------------------+
| name               |
+--------------------+
| Видеоадаптеры      |
| Жесткие диски      |
| Оперативная память |
+--------------------+
```

7.6. Вложенные запросы, возвращающие несколько строк

В предыдущем разделе рассмотрены запросы, где вложенный запрос возвращает единственное значение. Если вложенный запрос возвращает несколько строк, СУБД MySQL генерирует ошибку 1242: "Вложенный запрос возвращает более одной строки" (листинг 7.35).

Листинг 7.35. Вложенный запрос возвращает более одной строки

```
SELECT name FROM catalogs
WHERE id_catalog = (SELECT id_catalog FROM products);
ERROR 1242: Subquery returns more than 1 row
```

Для того чтобы предотвратить такую ошибку, там, где во вложенном запросе ожидается лишь одно значение, разумно использовать конструкцию `LIMIT 1` (листинг 7.36).

Листинг 7.36. Использование конструкции `LIMIT`

```
SELECT name FROM catalogs
WHERE id_catalog = (SELECT id_catalog FROM products LIMIT 1);
+------------+
| name       |
+------------+
| Процессоры |
+------------+
```

7.6.1. Ключевое слово *IN*

Для того чтобы выбрать строки за таблицы `catalogs`, у которых первичный ключ `id_catalog` совпадает с одним из значений, возвращаемых вложенным запросом, следует воспользоваться конструкцией `IN` (листинг 7.36).

Листинг 7.36. Использование конструкции IN

```
SELECT name FROM catalogs
WHERE id_catalog IN (SELECT id_catalog FROM products
                     GROUP BY id_catalog)
ORDER BY name;
+--------------------+
| name               |
+--------------------+
| Видеоадаптеры      |
| Жесткие диски      |
| Материнские платы  |
| Оперативная память |
| Процессоры         |
+--------------------+
```

Запрос, представленный в листинге 7.36, аналогичен запросу, показанному в листинге 7.37.

Листинг 7.37. Использование конструкции IN совместно со списком скалярных величин

```
SELECT name FROM catalogs
WHERE id_catalog IN (1,2,3,4,5)
ORDER BY name;
+--------------------+
| name               |
+--------------------+
| Видеоадаптеры      |
| Жесткие диски      |
| Материнские платы  |
| Оперативная память |
| Процессоры         |
+--------------------+
```

Для того чтобы возвратить строки, которые отсутствуют в результирующей таблице, возвращаемой вложенным запросом, следует воспользоваться оператором `NOT IN`, представленным в листинге 7.38.

Листинг 7.38. Использование оператора NOT IN

```
SELECT name FROM catalogs
WHERE id_catalog NOT IN (SELECT DISTINCT id_catalog
                         FROM products WHERE id_catalog < 3)
ORDER BY name;
```

```
+--------------------+
| name               |
+--------------------+
| Видеоадаптеры      |
| Жесткие диски      |
| Оперативная память |
+--------------------+
```

7.6.2. Ключевое слово *ANY*

Конструкция IN позволяет осуществить поиск величины в списке и выражает логику оператора "равно" (=). Однако на месте оператора = в этом запросе может стоять другой оператор сравнения (листинг 7.39).

Листинг 7.39. Вложенный запрос возвращает более одной строки

```
SELECT name FROM catalogs
WHERE id_catalog > (SELECT id_catalog FROM products);
ERROR 1242: Subquery returns more than 1 row
```

Данный запрос также завершается неудачей, но сформулировать его с использованием конструкции IN уже не получится. К счастью, язык запросов SQL обладает средствами решения подобных задач. Для этого применяется ключевое слово ANY (листинг 7.40).

Листинг 7.40. Использование ключевого слова ANY

```
SELECT id_catalog, name FROM catalogs
WHERE id_catalog > ANY (SELECT id_catalog FROM products);
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
```

Ключевое слово ANY применяется для сравнения значений с использованием одного из шести операторов сравнения (=, <>, <, <=, >, >=). Проверяемое значение id_catalog поочередно сравнивается с каждым элементом, который возвращает вложенный запрос. Если хотя бы одно из сравнений возвращает 1 (истина), строка выводится запросом. В листинге 7.40 происходит сравнение значений первичного ключа id_catalog (1, 2, 3, 4, 5), которые присутствуют в таблице catalogs, со значениями поля id_catalog (1, 2, 3, 4, 5) из таблицы products. Значение id_catalog = 1 не удовлетворяет ни одному условию:

```
1 > 1 -- 0 (ложь)
1 > 2 -- 0 (ложь)
1 > 3 -- 0 (ложь)
1 > 4 -- 0 (ложь)
1 > 5 -- 0 (ложь)
```

Поэтому в результаты эта строка не попадает, в то же время все остальные цифры удовлетворяют хотя бы одному условию и попадают в результирующую таблицу:

```
3 > 1 -- 1 (истина)
3 > 2 -- 1 (истина)
3 > 3 -- 0 (ложь)
3 > 4 -- 0 (ложь)
3 > 5 -- 0 (ложь)
```

То есть запрос вида

```
"WHERE X > ANY (SELECT Y...)"
```

можно интерпретировать как "где X больше хотя бы одного выбранного Y", а запрос вида

```
"WHERE X < ANY (SELECT Y...)"
```

следует читать "где X меньше хотя бы одного Y...".

7.6.3. Ключевое слово *ALL*

Вместо ключевого слова ANY может быть использовано ключевое слово ALL, которое точно так же применяется совместно с одним из шести операторов сравнения (=, <>, <, <=, >, >=). В этом случае проверяемое значение также поочередно сравнивается с каждым элементом, который возвращает вложенный запрос, но строка возвращается только тогда, когда все сравнения дают 1 (истина).

> ***Замечание***
>
> Если в выражениях с ключевым словом ANY используется логика ИЛИ, т. е. достаточно, чтобы срабатывало хотя бы одно из многих условий, то в случае ALL используется логика И — должны срабатывать все условия.

В листинге 7.41 представлен запрос, возвращающий все товарные позиции из таблицы products базы данных shop, цена которых превышает среднюю цену каждого из элементов каталога.

Листинг 7.41. Использование ключевого слова ALL

```
SELECT * FROM catalogs
WHERE id_catalog >= ALL (SELECT id_catalog FROM products
                         GROUP BY id_catalog);
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          5 | Оперативная память |
+------------+--------------------+
```

Для того чтобы понять логику запроса, представленного в листинге 7.41, полезно выполнить вложенный запрос отдельно (листинг 7.42).

Листинг 7.42. Вложенный запрос

```
SELECT id_catalog FROM products GROUP BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
```

Условие ALL в листинге 7.41 эквивалентно требованию вывести все каталоги, чей первичный ключ больше или равен каждого значения из результирующей таблицы запроса, представленного в листинге 7.42.

7.7. Групповые условия. Ключевое слово *HAVING*

SQL-запросы с дополнительными условиями после ключевого слова HAVING требуют более подробного рассмотрения конструкции GROUP BY. Применение агрегатных функций (таких как COUNT(), SUM(), MIN(), MAX() и др.) к этой конструкции приводит к тому, что вычисляется значение функции внутри группы, а не всей таблицы в целом. В листинге 7.43 поля таблицы группируются по полю id_catalog, использование функции COUNT() позволяет вычислить, сколько записей таблицы products имеют одинаковые значения данного поля.

Листинг 7.43. Совместное использование GROUP BY и функции COUNT()

```
SELECT id_catalog, COUNT(id_catalog)
FROM products
GROUP BY id_catalog
ORDER BY id_catalog;
+------------+-------------------+
| id_catalog | COUNT(id_catalog) |
+------------+-------------------+
|          1 |                 3 |
|          2 |                 2 |
|          3 |                 1 |
|          4 |                 1 |
|          5 |                 2 |
+------------+-------------------+
```

В рамках групповых условий возможно применение условия WHERE (листинг 7.44).

Листинг 7.44. Использование ключевого слова WHERE совместно с GROUP BY

```
SELECT id_catalog, COUNT(id_catalog)
FROM products
WHERE id_catalog > 2
GROUP BY id_catalog
ORDER BY id_catalog;
+------------+------------------+
| id_catalog | COUNT(id_catalog) |
+------------+------------------+
|          3 |                1 |
|          4 |                1 |
|          5 |                2 |
+------------+------------------+
```

Часто при составлении условий требуется ограничить выборку по результату агрегатной функции, например, выбрать группы, где количество записей больше двух или равно двум. Использование для этих целей конструкции WHERE приводит к ошибке.

Для решения этой проблемы вместо ключевого слова WHERE используется ключевое слово HAVING, которое располагается вслед за конструкцией GROUP BY (листинг 7.45).

Листинг 7.45. Использование ключевого слова GROUP BY

```
SELECT id_catalog, COUNT(id_catalog) AS total
FROM products
GROUP BY id_catalog
HAVING total > 1
ORDER BY id_catalog;
+------------+-------+
| id_catalog | total |
+------------+-------+
|          1 |     3 |
|          2 |     2 |
|          5 |     2 |
+------------+-------+
```

В условии HAVING можно использовать все столбцы результирующей таблицы, не только вычисляемые.

7.8. Многотабличные запросы SELECT

Многотабличные запросы позволяют выводить в результирующей таблице информацию из нескольких таблиц. Для обсуждения многотабличных запросов воспользуемся уже известными таблицами `catalogs` и `products`.

Для осуществления запроса сразу к нескольким таблицам их имена перечисляются после ключевого слова `FROM` через запятую или оператор `JOIN`. Условие `USING` позволяет указать поле, связывающее таблицы, при этом упоминание таблиц в других частях запроса должно предваряться названием таблицы, которому это поле принадлежит (листинг 7.46).

Листинг 7.46. Многотабличный запрос

```
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
SELECT * FROM products;
+------------+-----------------------+------------+
| id_product | name                  | id_catalog |
+------------+-----------------------+------------+
|          1 | Материнская плата N 1 |          2 |
|          2 | Оперативная память N 1 |          5 |
|          3 | Процессор N 1         |          1 |
|          4 | Процессор N 2         |          1 |
|          5 | Процессор N 3         |          1 |
|          6 | Оперативная память N 2 |          5 |
|          7 | Материнская плата N 2 |          2 |
|          8 | Жесткий диск N 1      |          4 |
|          9 | Видеоадаптер N 1      |          3 |
+------------+-----------------------+------------+
SELECT
  products.id_product AS id_product,
  products.name AS product,
  catalogs.name AS catalog
FROM
  catalogs JOIN products
USING(id_catalog);
```

```
+------------+-----------------------+--------------------+
| id_product | product               | catalog            |
+------------+-----------------------+--------------------+
|          1 | Материнская плата N 1 | Материнские платы  |
|          2 | Оперативная память N 1| Оперативная память |
|          3 | Процессор N 1         | Процессоры         |
|          4 | Процессор N 2         | Процессоры         |
|          5 | Процессор N 3         | Процессоры         |
|          6 | Оперативная память N 2| Оперативная память |
|          7 | Материнская плата N 2 | Материнские платы  |
|          8 | Жесткий диск N 1      | Жесткие диски      |
|          9 | Видеоадаптер N 1      | Видеоадаптеры      |
+------------+-----------------------+--------------------+
```

Когда имена столбцов, через которые связываются таблицы, не совпадают или условия связи более сложные, вместо ключевого слова `USING` используется ключевое слово `ON` (листинг 7.47).

Листинг 7.47. Использование ключевого слова ON

```
SELECT
  products.id_product AS id_product,
  products.name AS product,
  catalogs.name AS catalog
FROM
  catalogs JOIN products
ON catalogs.id_catalog = products.id_catalog;
```

Иногда необходимо ограничить выборку записями лишь одной таблицы. Например, для вывода списка каталогов из таблицы `catalogs` и количества соответствующих им товарных позиций из таблицы `products` можно воспользоваться ключевым словом `GROUP BY` по полю `id_catalog` таблицы `catalogs` (листинг 7.48).

Листинг 7.48. Группировка значений

```
SELECT
  catalogs.id_catalog AS id_catalog,
  catalogs.name AS catalog,
  COUNT(products.id_product) AS total
FROM
  catalogs JOIN products
USING(id_catalog)
GROUP BY catalogs.id_catalog;
+------------+--------------------+-------+
| id_catalog | catalog            | total |
+------------+--------------------+-------+
|          1 | Процессоры         |     3 |
|          2 | Материнские платы  |     2 |
```

```
|          3 | Видеоадаптеры      |     1 |
|          4 | Жесткие диски      |     1 |
|          5 | Оперативная память |     2 |
+------------+--------------------+-------+
```

Чтобы получить список соответствующих строке значений из второй таблицы, подойдет функция `GROUP_CONCAT()`. В листинге 7.49 выводится список каталогов из таблицы `catalogs`, причем в последнем столбце результирующей таблицы приводится список первичных ключей `id_product` таблицы `products`, связанных с данным каталогом.

Листинг 7.49. Использование функции `GROUP_CONCAT()`

```
SELECT
  catalogs.id_catalog AS id_catalog,
  catalogs.name AS catalog,
  GROUP_CONCAT(id_product ORDER BY id_product SEPARATOR ', ') AS list
FROM
  catalogs JOIN products
USING(id_catalog)
GROUP BY catalogs.id_catalog;
+------------+--------------------+---------+
| id_catalog | catalog            | list    |
+------------+--------------------+---------+
|          1 | Процессоры         | 3, 4, 5 |
|          2 | Материнские платы  | 1, 7    |
|          3 | Видеоадаптеры      | 9       |
|          4 | Жесткие диски      | 8       |
|          5 | Оперативная память | 2, 6    |
+------------+--------------------+---------+
```

7.9. Выбор случайных точек из таблицы

Имеется таблица `points`, хранящая значения координат в первом поле `value` и принадлежность координат оси абсцисс `x` или оси ординат `y` во втором поле `axis` (листинг 7.50). Нам необходимо выбрать случайную координату `x` и случайную координату `y` одним запросом.

Листинг 7.50. Таблица координат `points`

```
CREATE TABLE points (
  value INT(11) NOT NULL,
  axis ENUM('x','y') NOT NULL
);
INSERT INTO points VALUES(3, 'x');
INSERT INTO points VALUES(5, 'x');
```

```
INSERT INTO points VALUES(4, 'x');
INSERT INTO points VALUES(1, 'x');
INSERT INTO points VALUES(10, 'x');
INSERT INTO points VALUES(8, 'y');
INSERT INTO points VALUES(1, 'y');
INSERT INTO points VALUES(4, 'y');
INSERT INTO points VALUES(9, 'y');
INSERT INTO points VALUES(2, 'y');
```

Для решения данной задачи необходимо таблицу `points` перебрать дважды, выбрав для нее два различных псевдонима (листинг 7.51).

Листинг 7.51. Выбор случайной точки

```
SELECT a.axis, a.value, b.axis, b.value
FROM
  points AS a,
  points AS b
WHERE
  a.axis = 'x' AND
  b.axis = 'y'
ORDER BY RAND()
LIMIT 1;
+------+-------+------+-------+
| axis | value | axis | value |
+------+-------+------+-------+
| x    |     3 | y    |     4 |
+------+-------+------+-------+
```

7.10. Многотабличный запрос *DELETE*

Оператор `DELETE` можно использовать в многотабличных запросах. Для этого сразу после ключевого слова `FROM` имена таблиц перечисляются через запятую. Пусть требуется удалить из таблицы `catalogs` все записи, чей первичный ключ `id_catalog` больше 3. Для этого можно использовать запрос, представленный в листинге 7.52.

Замечание

В многотабличных запросах с участием SQL-оператора `DELETE` не допускается использование конструкций `ORDER BY` и `LIMIT`.

Листинг 7.52. Удаление записей из таблицы `catalogs`

```
DELETE FROM catalogs WHERE id_catalog > 3;
SELECT * FROM catalogs;
```

```
+------------+-------------------+
| id_catalog | name              |
+------------+-------------------+
|          1 | Процессоры        |
|          2 | Материнские платы |
|          3 | Видеоадаптеры     |
+------------+-------------------+
```

Запрос из листинга 7.52 уничтожит записи в таблице `catalogs` для элементов каталога "Жесткие диски" и "Оперативная память" с первичными ключами `id_catalog`, равными 4 и 5, соответственно. Однако записи в таблице `products` останутся нетронутыми и будут ссылаться на несуществующие позиции каталога. В этом можно убедиться, запросив число товарных позиций для каждого из каталогов при помощи SQL-запроса, представленного в листинге 7.53.

Листинг 7.53. Товарные позиции не удалились из таблицы `products`

```
SELECT id_catalog, COUNT(*) FROM products GROUP BY id_catalog;
+------------+----------+
| id_catalog | COUNT(*) |
+------------+----------+
|          1 |        9 |
|          2 |        6 |
|          3 |        4 |
|          4 |        5 |
|          5 |        6 |
+------------+----------+
```

В листинге 7.54 из таблицы `products` удаляются все записи, для которых имеется соответствие в таблице `catalogs`.

Листинг 7.54. Многотабличное удаление из таблицы `products`

```
DELETE FROM products USING products, catalogs
WHERE products.id_catalog = catalogs.id_catalog;
```

Следует отметить, что записи касаются только таблицы `products` — удаление производится лишь из таблиц, перечисленных после ключевого слова `FROM`. Запрос, представленный в листинге 7.55, затронет уже обе таблицы.

Листинг 7.55. Многотабличное удаление из таблиц `products` и `catalogs`

```
DELETE FROM products, catalogs USING products, catalogs
WHERE products.id_catalog = catalogs.id_catalog;
```

Таким образом, возвращаясь к задаче удаления позиций каталога с первичным ключом больше 3, можно сформировать следующий SQL-запрос, который удалит записи как из таблицы `catalogs`, так и из таблицы `products` (листинг 7.56).

Листинг 7.56. Удаление позиций каталога с первичным ключом больше 3

```
DELETE products, catalogs FROM products, catalogs
WHERE products.id_catalog = catalogs.id_catalog AND
catalogs.id_catalog > 3;
SELECT * FROM catalogs;
+------------+-------------------+
| id_catalog | name              |
+------------+-------------------+
|          1 | Процессоры        |
|          2 | Материнские платы |
|          3 | Видеоадаптеры     |
+------------+-------------------+
SELECT id_catalog, COUNT(*) FROM products GROUP BY id_catalog;
+------------+----------+
| id_catalog | COUNT(*) |
+------------+----------+
|          1 |        9 |
|          2 |        6 |
|          3 |        4 |
+------------+----------+
```

Как видно из листинга 7.56, многотабличное удаление не нарушает ссылочной целостности базы данных.

7.11. Удаление повторяющихся записей

Чтобы предотвратить появление повторяющихся значений, лучше всего воспользоваться уникальным ключом, однако если повторяющиеся записи все же возникли, встает вопрос очистки базы данных. Для демонстрации приемов удаления создадим таблицу `users`, содержащую две одинаковые записи информации о пользователях (листинг 7.57).

Листинг 7.57. Таблица `users`

```
CREATE TABLE users (
  id_position INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  pass TINYTEXT NOT NULL,
  email TINYTEXT NOT NULL,
  PRIMARY KEY (id_position)
);
INSERT INTO users VALUES(1, 'cheops', 'pass', 'igor@softtime.ru');
INSERT INTO users VALUES(2, 'makkuz', 'pass', 'kuznetsov@softtime.ru');
INSERT INTO users VALUES(3, 'cheops', 'pass', 'igor@softtime.ru');
```

Прием удаления повторяющихся записей через вложенные запросы и функцию `MAX()`, часто использующийся в альтернативных СУБД, в MySQL, не работает (листинг 7.58).

Листинг 7.58. Попытка удаления повторяющихся записей

```
DELETE FROM users
WHERE users.id_position < (SELECT MAX(snd.id_position)
                           FROM `users` AS snd
                           WHERE users.name = snd.name);
ERROR 1093 (HY000): You can't specify target table 'users' for update in FROM
clause
```

Выходом из ситуации может послужить многотабличное удаление из таблицы `users`, построенное на самообъединении (листинг 7.59).

Листинг 7.59. Удаление повторяющихся записей из таблицы `users`

```
DELETE fst
FROM
  users fst
JOIN
 (SELECT name, MAX(id_position) AS id_max
  FROM users
  GROUP BY name
  HAVING COUNT(*) > 1) AS snd
ON
  fst.name = snd.name AND
  fst.id_position < snd.id_max;
SELECT * FROM users;
+-------------+--------+-------+----------------------+
| id_position | name   | pass  | email                |
+-------------+--------+-------+----------------------+
|           2 | makkuz | pass  | kuznetsov@softtime.ru |
|           3 | cheops | passw | igor@softtime.ru      |
+-------------+--------+-------+----------------------+
```

ГЛАВА 8

PHP и MySQL

Для взаимодействия PHP и MySQL предназначено расширение php_mysql, которое позволяет устанавливать соединение с сервером MySQL, выполнять запросы и получать доступ к результирующей таблице.

8.1. Установка соединения с базой данных

В табл. 8.1 приведен список функций, ответственных за управление соединением с базой данных. Любой сеанс с базой данных начинается с открытия соединения с последующим его закрытием либо явно при помощи функции `mysql_close()`, либо в результате завершения работы скрипта.

Таблица 8.1. *Функции управления соединением с сервером*

Функция	Описание
`mysql_connect()`	Открывает соединение с сервером
`mysql_close()`	Закрывает соединение с сервером
`mysql_ping()`	Проверяет соединение с сервером и повторно соединяется при необходимости

Соединение с базой данных MySQL осуществляется при помощи функции `mysql_connect()`, которая имеет следующий синтаксис:

```
mysql_connect ([$server [,
                $username [,
                $password [,
                $new_link [,
                $client_flags]]]]])
```

Эта функция устанавливает соединение с сервером MySQL, сетевой адрес которого задается параметром *`$server`*. Вторым и третьим аргументами этой функции являются имя пользователя базы данных *`$username`* и его пароль *`$password`*, соответственно.

По умолчанию повторный вызов функции `mysql_connect()` с теми же аргументами не приводит к установлению нового соединения, вместо этого функция возвращает дескриптор уже существующего соединения. Если четвертому параметру *`$new_link`* присвоить значение `TRUE`, будет открыто новое соединение с сервером. Параметр *`$client_flags`* должен быть комбинацией из следующих констант:

- `MYSQL_CLIENT_COMPRESS` — предписывает использование протокола сжатия при обмене информацией между сервером и клиентом;
- `MYSQL_CLIENT_IGNORE_SPACE` — в SQL-запросах после имен функций разрешается использование пробелов;
- `MYSQL_CLIENT_INTERACTIVE` — ждать `interactive_timeout` секунд до закрытия соединения, если между сервером и клиентом не происходит обмен данными.

> ***ЗАМЕЧАНИЕ***
>
> Все аргументы функции являются необязательными. В случае их отсутствия, по умолчанию, для этой функции устанавливаются следующие параметры: `server = 'localhost:3306'`, `username` принимает значение владельца процесса сервера, а `password` принимает пустую строку.

В случае успеха функция возвращает дескриптор соединения с сервером, при неудаче возвращает значение `FALSE`.

В листинге 8.1 приведен пример использования функции `mysql_connect()`.

Листинг 8.1. Функция `mysql_connect()`

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";

  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx) // Если дескриптор равен 0, соединение не установлено
  {
    exit ("<P>В настоящий момент сервер базы данных недоступен, поэтому
             корректное отображение страницы невозможно.</P>");
  }
  else
  {
    echo "<P>Соединение установлено.</P>";
  }
?>
```

> ***ЗАМЕЧАНИЕ***
>
> Для подавления вывода сообщений об ошибках, генерируемых PHP, в окне браузера, в листинге 8.1 перед функцией `mysql_connect()` размещен символ `@`.

Переменные $dblocation, $dbuser и $dbpasswd хранят имя сервера, имя пользователя и пароль и, как правило, прописываются в отдельном файле (к примеру, config.php), который потом вставляется в каждый PHP-файл. В последнем имеется код для работы с MySQL.

Для корректной обработки строк с русским текстом перед началом выполнения запросов необходимо выставить кодировку соединения при помощи запроса SET NAMES, представленного в листинге 8.2. Иначе русский текст будет попадать в базу данных в виде знаков вопроса.

ЗАМЕЧАНИЕ

Функция mysql_query() рассматривается более подробно в *разд. 8.3*.

Листинг 8.2. Установка кодировки соединения Windows-1251

```
<?php
  mysql_query ("SET NAMES 'cp1251'");
?>
```

Закрытие соединения осуществляется при помощи функции mysql_close(), которая имеет следующий синтаксис:

```
mysql_close([$link_identifier])
```

В качестве необязательного параметра функция принимает дескриптор открытого соединения $link_identifier. Если этот параметр не указан, закрывается последнее открытое соединение. Функция возвращает TRUE в случае успеха и FALSE при возникновении ошибки.

В листинге 8.3 приведен пример использования функции.

Листинг 8.3. Закрытие соединения при помощи функции mysql_close()

```
<?php
  // Устанавливаем соединение с базой данных
  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx)
  {
    // Выводим предупреждение
    exit ("<P>В настоящий момент сервер базы данных недоступен, поэтому
    корректное отображение страницы невозможно.</P>");
  }
  if (mysql_close($dbcnx)) // закрываем соединение
  {
    echo "Соединение с базой данных прекращено";
  }
  else
  {
    echo "Не удалось завершить соединение";
  }
?>
```

Если скрипт выполняется непродолжительное время, явное закрытие соединения при помощи функции mysql_close() не обязательно, т. к. скрипт закрывает все соединения после своего завершения. Явное закрытие соединения требуется в том случае, если после выполнения всех запросов выполняется какое-то длительное действие, не дающее скрипту завершить свою работу, например, загрузка объемного файла, отдаваемого клиенту PHP-скриптом. В этом случае соединение будет оставаться занятым и бездействовать все время, пока клиент загружает скрипт.

8.2. Выбор базы данных

После того как соединение установлено, необходимо выбрать базу данных по умолчанию, с которой будет производиться работа (в консольном клиенте mysql для это служит оператор USE). Это позволит не указывать расширенные имена таблиц в запросах. Для этого используется функция mysql_select_db(), вызов которой эквивалентен вызову команды USE в SQL-запросе. Функция имеет следующий синтаксис:

```
mysql_select_db($database_name [, $link_identifier])
```

Функция принимает в качестве аргументов название выбираемой базы данных *$database_name* и дескриптор соединения *$link_identifier*. Функция возвращает TRUE при успешном выполнении операции и FALSE в противном случае. В листинге 8.4 приводится пример выбора базы данных с именем $dbname.

Листинг 8.4. Выбор базы данных при помощи функции mysql_select_db()

```
<?php
  ...
  // Код соединения с базой данных
  if (!@mysql_select_db($dbname, $dbcnx))
  {
    echo "<P>В настоящий момент база данных не доступна, поэтому
              корректное отображение страницы невозможно.</P>";
    exit();
  }
?>
```

Имеет смысл помещать функции для соединения и выбора базы данных в тот же файл (config.php), где объявлены переменные с именами сервера, пользователя и паролем (листинг 8.5), и подключать данный файл при помощи конструкции include_once() или require_once(), если требуется обращение к базе данных MySQL.

Листинг 8.5. Универсальный файл config.php

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
```

```
  // Имя базы данных на хостинге или локальной машине
  $dbname = "test";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";

  // Устанавливаем соединение с базой данных
  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx) {
   exit( "<P>В настоящий момент сервер базы данных не доступен,
            поэтому корректное отображение страницы невозможно.</P>" );
  }
  // Выбираем базу данных
  if (! @mysql_select_db($dbname, $dbcnx) ) {
    exit( "<P>В настоящий момент база данных не доступна,
             поэтому корректное отображение страницы невозможно.</P>" );
  }

  @mysql_query("SET NAMES 'cp1251'");
?>
```

8.3. Выполнение SQL-запросов

Выполнение SQL-запросов осуществляется при помощи функции `mysql_query()`, которая имеет следующий синтаксис:

```
mysql_query($query [, $link_identifier])
```

Первый аргумент функции представляет собой строку с запросом *$query*, второй, *$link_identifier* — дескриптор соединения, возвращаемый функцией `mysql_connect()`.

> ***Замечание***
>
> Если открытые соединения отсутствуют, функция пытается соединиться с СУБД, аналогично функции `mysql_connect()` без параметров.

> ***Замечание***
>
> При передаче запроса функции `mysql_query()` точку с запятой в конце запроса, обязательную при работе с клиентом `mysql`, можно не ставить. Кроме того, следует помнить, что функция `mysql_query()` выполняет лишь один запрос, поэтому не допускается передача ей нескольких SQL-запросов.

Функция возвращает дескриптор результирующей таблицы в случае успеха и `FALSE` в случае неудачного выполнения запроса. Если запрос не возвращает результирующей таблицы, можно не сохранять дескриптор в отдельной переменной. В листинге 8.6 представлен скрипт, создающий новую таблицу `catalogs`.

Листинг 8.6. Использование функции `mysql_query()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";
  // Формируем и выполняем SQL-запрос
  $query = "CREATE TABLE catalogs (
              id_catalog INT(11) NOT NULL AUTO_INCREMENT,
              name TINYTEXT NOT NULL,
              PRIMARY KEY (id_catalog))";
  if (mysql_query($query))
  {
     echo "Таблица создана успешно";
  }
  else
  {
     exit("Ошибка выполнения запроса — ".mysql_error());
  }
?>
```

В листинге 8.6 при неудачном выполнении функции `mysql_query()` функции `exit()`, останавливающей работу скрипта, в качестве параметра передается значение ошибки, возвращаемое функцией `mysql_erorr()`.

Дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`, используется далее для получения значений возвращаемых СУБД. Обычно это осуществляется при помощи одной из пяти функций: `mysql_result()`, `mysql_fetch_row()`, `mysql_fetch_assoc()`, `mysql_fetch_array()` и `mysql_fetch_object()`.

В примерах следующих разделов будет использоваться таблица `catalogs`, дамп которой представлен в листинге 8.7.

Листинг 8.7. Дамп таблицы `catalogs`

```
CREATE TABLE catalogs (
  id_catalog int(11) NOT NULL auto_increment,
  name tinytext NOT NULL,
  PRIMARY KEY  (id_catalog),
  FULLTEXT KEY name (name)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
INSERT INTO catalogs VALUES (1,'Процессоры');
INSERT INTO catalogs VALUES (2,'Материнские платы');
INSERT INTO catalogs VALUES (3,'Видеоадаптеры');
INSERT INTO catalogs VALUES (4,'Жесткие диски');
INSERT INTO catalogs VALUES (5,'Оперативная память');
```

8.4. Получение результатов запроса

Функция `mysql_query()` сама по себе не возвращает результатов запроса, она лишь возвращает дескриптор, при помощи которого можно получить доступ к результирующей таблице посредством одной из функций, представленных в табл. 8.2.

Таблица 8.2. *Таблицы доступа к результирующей таблице*

Функция	Описание
`mysql_result()`	Доступ к единичному значению из результирующей таблицы
`mysql_fetch_row()`	Доступ к строке результирующей таблицы и представление результатов в виде числового массива
`mysql_fetch_assoc()`	Доступ к строке результирующей таблицы и представление результатов в виде ассоциативного массива
`mysql_fetch_array()`	Доступ к строке результирующей таблицы и представление результатов в виде смешанного массива
`mysql_fetch_object()`	Доступ к строке результирующей таблицы и представление результатов в виде массива

Первая функция `mysql_result()` возвращает результат запроса, выполненного функцией `mysql_query()`. С ее помощью можно получить доступ к отдельному полю записи. Функция имеет следующий синтаксис:

```
mysql_result($result, $row [, $field])
```

В качестве первого аргумента *`$result`* функция принимает дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`. Второй аргумент *`$row`* представляет собой номер столбца, который необходимо вернуть. Третий необязательный параметр *`$field`* — это имя поля таблицы. В листинге 8.8 при помощи данной функции выводится количество записей таблицы `catalogs`.

Листинг 8.8. Извлечение результатов запроса функцией `mysql_result()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT COUNT(*) FROM catalogs";
  // Выполняем SQL-запрос
  $cat = mysql_query($query);
  // Обрабатываем результаты запроса
  if ($cat) echo "Количество записей — ".mysql_result($cat, 0);
  else exit("Ошибка выполнения запроса — ".mysql_error());
?>
```

Результат работы скрипта из листинга 8.8 представлен на рис. 8.1.

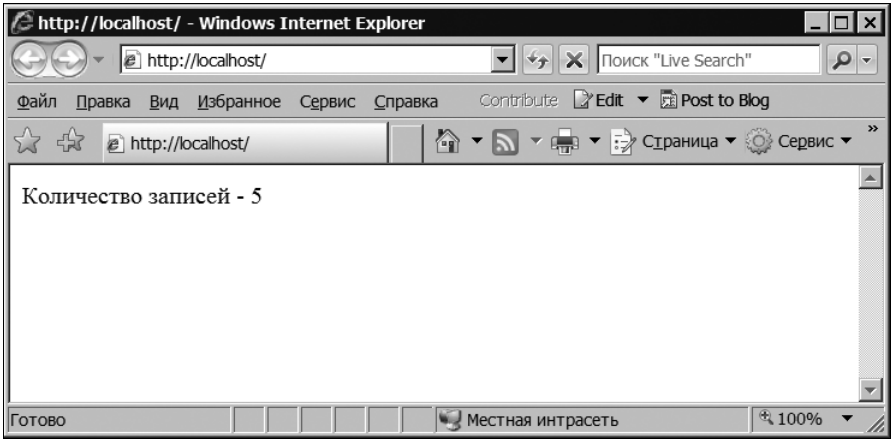


Рис. 8.1. Результат работы скрипта из листинга 8.8

Функция `mysql_fetch_row()` возвращает текущую запись в результирующей таблице в виде неассоциативного массива. Повторный вызов функции переводит курсор в результирующей таблице на следующую. При достижении конца таблицы функция возвращает `FALSE`. Функция `mysql_fetch_row()` имеет следующий синтаксис:

```
mysql_fetch_row($result)
```

В качестве единственного аргумента *`$result`* функция принимает дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`. Функция возвращает массив, каждый элемент которого соответствует одному полю в записи, или `FALSE`, если курсор достиг конца результирующей таблицы. При достижении конца результирующей таблицы, вызов функции приводит к возвращению `FALSE`. При работе с массивом, который возвращает функция `mysql_fetch_row()`, удобно использовать конструкцию `list()`, преобразующую элементы массива в переменные. В листинг 8.9 выводится HTML-таблица, в которой размещаются данные из таблицы `catalogs`.

Листинг 8.9. Извлечение результатов запроса функцией `mysql_fetch_row()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT * FROM catalogs";
  // Выполняем SQL-запрос
  $cat = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$cat) exit("Ошибка выполнения запроса — ".mysql_error());
  // Так как запрос может возвращать несколько строк, применяем цикл
  echo "<table border='1'>";
```

```
  while(list($id_author, $name) = mysql_fetch_row($cat))
  {
    echo "<tr>
            <td>$id_author</td>
            <td>$name</td>
          </tr>";
  }
  echo "</table>";
?>
```

Результат работы скрипта из листинга 8.9 представлен на рис. 8.2.

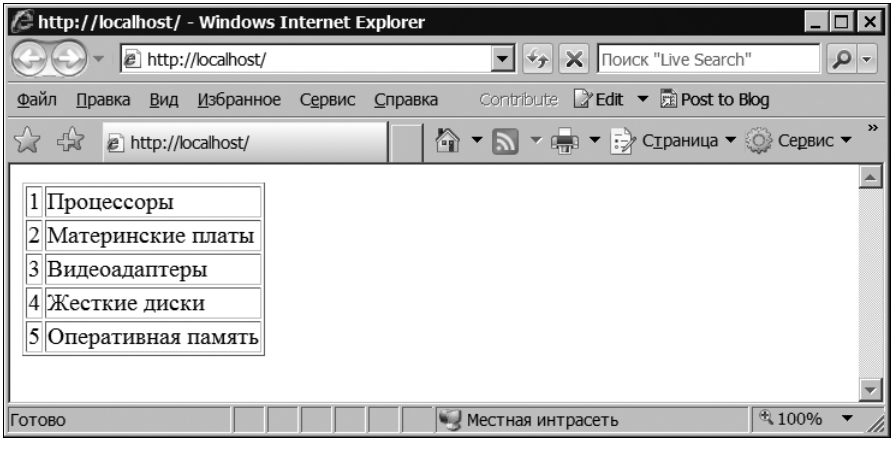


Рис. 8.2. Результат работы скрипта из листинга 8.9

Функция `mysql_fetch_assoc()` возвращает текущую запись в результирующей таблице в виде ассоциативного массива. Повторный вызов функции переводит курсор в результирующей таблице на следующую запись. Функция `mysql_fetch_assoc()` имеет следующий синтаксис:

```
mysql_fetch_assoc($result)
```

В качестве единственного аргумента *`$result`* функция принимает дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`. Функция возвращает массив, каждый элемент которого соответствует одному полю в записи, или `FALSE`, если курсор достиг конца результирующей таблицы. Причем в качестве ключей массива выступают имена полей в результирующей таблице. В листинге 8.10 выводится HTML-таблица, в которой размещаются данные из таблицы `catalogs`.

Листинг 8.10. Использование функции `mysql_fetch_assoc()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";
```

```
  // Формируем SQL-запрос
  $query = "SELECT * FROM catalogs";
  // Выполняем SQL-запрос
  $cat = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$cat) exit("Ошибка выполнения запроса — ".mysql_error());
  // Формируем таблицу
  echo "<table border='1'>";
  while($catalogs = mysql_fetch_assoc($cat))
  {
    echo "<tr>
            <td>".$catalogs['id_catalog']."</td>
            <td>".$catalogs['name']."</td>
          </tr>";
  }
  echo "</table>";
?>
```

Результат работы скрипта из листинга 8.10 выглядит точно так же, как результат работы скрипта из листинга 8.9 (см. рис. 8.2). Следует обратить внимание, что на каждой итерации цикла `while` формируется массив `$catalogs`, в качестве ключей которого выступают имена столбцов из таблицы `catalogs`. Если столбец является вычисляемым, то, для того чтобы обратиться к нему, потребуется полное обращение к имени столбца. Так в листинге 8.11 при помощи встроенной функции MySQL `VERSION()` извлекается версия сервера MySQL. При этом в качестве ключа массива выступает строка `'VERSION()'`.

Листинг 8.11. Получение версии сервера с использованием `mysql_fetch_assoc()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT VERSION()";
  // Выполняем SQL-запрос
  $ver = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$ver) exit("Ошибка выполнения запроса — ".mysql_error());
  // Определяем таблицу и заголовок
  $version = mysql_fetch_assoc($ver);
  echo $version['VERSION()'];
?>
```

Использование в качестве ключей полной сигнатуры вычисляемого столбца не всегда удобно, т. к. ключ может получиться громоздким или динамически изменяться.

Для упрощения имени столбца в этом случае прибегают к назначению псевдонима при помощи ключевого слова AS. Поэтому скрипт из листинга 8.11 можно переписать так, как показано в листинге 8.12.

Листинг 8.12. Использование псевдонима, назначаемого при помощи оператора AS

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT VERSION() AS ver";
  // Выполняем SQL-запрос
  $ver = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$ver) exit("Ошибка выполнения запроса — ".mysql_error());
  // Определяем таблицу и заголовок
  $version = mysql_fetch_assoc($ver);
  echo $version['ver'];
?>
```

Функция mysql_fetch_array() возвращает текущую запись в результирующей таблице в виде массива. Причем тип массива (численный или ассоциативный) может быть выбран. Повторный вызов функции переводит курсор в результирующей таблице на следующую запись. Функция mysql_fetch_array() имеет такой синтаксис:

```
mysql_fetch_array($result [, $result_type])
```

В качестве первого аргумента $result функция принимает дескриптор результирующей таблицы, возвращаемый функцией mysql_query(). Второй необязательный параметр может принимать три значения:

- MYSQL_ASSOC — возврат результата работы в виде ассоциативного массива;
- MYSQL_NUM — возврат результата работы в виде численного массива;
- MYSQL_BOTH — возврат результата работы в виде массива, содержащего как численные, так и ассоциативные индексы.

> ***Замечание***
>
> По умолчанию второй аргумент принимает значение MYSQL_BOTH.

> ***Замечание***
>
> Режим работы функции mysql_fetch_array(), принимающей в качестве второго аргумента константы MYSQL_ASSOC и MYSQL_NUM, аналогичен функциям mysql_fetch_assoc() и mysql_fetch_row() соответственно.

При возврате ассоциативного массива в качестве индексов выступают имена столбцов результирующей таблицы, из которой производится выборка. Наиболее интересен режим MYSQL_BOTH. В листинге 8.13 извлекается первая запись таблицы catalogs и выводится дамп массива, возвращаемого функцией mysql_fetch_array().

Замечание

Для вывода дампа массива применяется функция `print_r()`. Результат функции заключается в теги `<pre>` и `</pre>`, чтобы сохранить форматирование, т. к. браузер воспринимает символы перевода строки как пробельные символы.

Листинг 8.13. Использование функции `mysql_fetch_array()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT * FROM catalogs";
  // Выполняем SQL-запрос
  $prd = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$prd) exit("Ошибка выполнения запроса — ".mysql_error());
  // Возвращаем результат в виде массива
  $product = mysql_fetch_array($prd);
  echo "<pre>";
  print_r($product);
  echo "</pre>";
?>
```

Использовать функцию `mysql_fetch_array()` можно и для получения записей результирующей таблицы в цикле по аналогии с функциями `mysql_fetch_assoc()` и `mysql_fetch_row()` (листинг 8.14).

Листинг 8.14. Извлечение содержимого таблицы `catalogs`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT * FROM catalogs";
  // Выполняем SQL-запрос
  $cat = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$cat) exit("Ошибка выполнения запроса — ".mysql_error());
  // Формируем таблицу
  echo "<table border='1'>";
  while($catalog = mysql_fetch_array($cat))
  {
    echo "<tr>
          <td>".$catalogs['id_catalog']."</td>
```

```
            <td>".$catalogs['name']."</td>
          </tr>";
  }
  echo "</table>";
?>
```

Функция `mysql_fetch_object()` возвращает текущую запись в виде объекта, имена членов которого совпадают с именами полей в результирующей таблице. Повторный вызов функции переводит курсор в результирующей таблице на следующую запись. Функция `mysql_fetch_object()` имеет такой синтаксис:

```
mysql_fetch_object($result)
```

В качестве единственного аргумента *`$result`* функция принимает дескриптор запроса, возвращаемый функцией `mysql_query()`. Возвращает объект с членами, соответствующими столбцам в результирующей таблице, или `FALSE`, если рядов больше нет.

> ***ЗАМЕЧАНИЕ***
>
> Имена полей, возвращаемых этой функцией, не зависят от регистра.

В листинге 8.15 приведен пример, в котором с помощью этой функции из таблицы `catalogs` выводятся первичный ключ таблицы `id_catalog` и имя элемента каталога `name`.

Листинг 8.15. Использование функции `mysql_fetch_object()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT * FROM catalogs";
  // Выполняем SQL-запрос
  $cat = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$cat) exit("Ошибка выполнения запроса — ".mysql_error());
  while($catalog = mysql_fetch_object($cat))
  {
    echo "id_catalog: ".$catalog->id_catalog."<br />";
    echo "name: ".$catalog->name."<br />";
  }
?>
```

Результат работы скрипта из листинга 8.15 представлен на рис. 8.3.

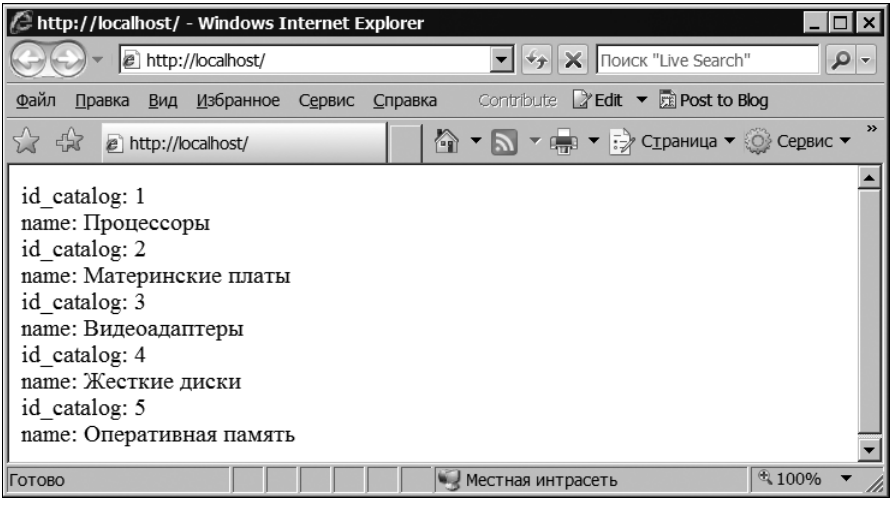


Рис. 8.3. Результат работы скрипта из листинга 8.15

8.5. Количество строк в таблице

Одной из распространенных задач при выборке из базы данных является определение числа записей, которые возвращает функция `mysql_query()`. Для этого предназначена функция `mysql_num_rows()`, которая имеет следующий синтаксис:

```
mysql_num_rows($result)
```

В качестве единственного аргумента `$result` функция принимает дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`.

> ***Замечание***
>
> Эта функция работает только с запросами `SELECT`. Чтобы получить количество рядов, обработанных функциями `INSERT`, `UPDATE`, `DELETE`, следует использовать функцию `mysql_affected_rows()`.

> ***Замечание***
>
> Функция `mysql_num_fields()` позволяет определить число столбцов в результате.

Создадим таблицу `test`, содержащую два поля: первичный ключ `id_test` и текстовое поле `name` (листинг 8.16).

Листинг 8.16. Таблица `test`

```
CREATE TABLE test (
  id_test int(11) NOT NULL auto_increment,
  name tinytext NOT NULL,
  PRIMARY KEY  (id_test)
);
```

В настоящий момент в таблице нет ни одной записи, поэтому попытка извлечь результирующую таблицу будет завершаться неудачей (листинг 8.17).

Листинг 8.17. Попытка извлечь несуществующую запись

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT name FROM test LIMIT 1";
  // Выполняем SQL-запрос
  $tst = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$tst) exit("Ошибка выполнения запроса — ".mysql_error());
  // Выводим результат
  echo mysql_result($tst,0);
?>
```

В результате работы скрипта из листинга 8.17 будет возникать ошибка (рис. 8.4).

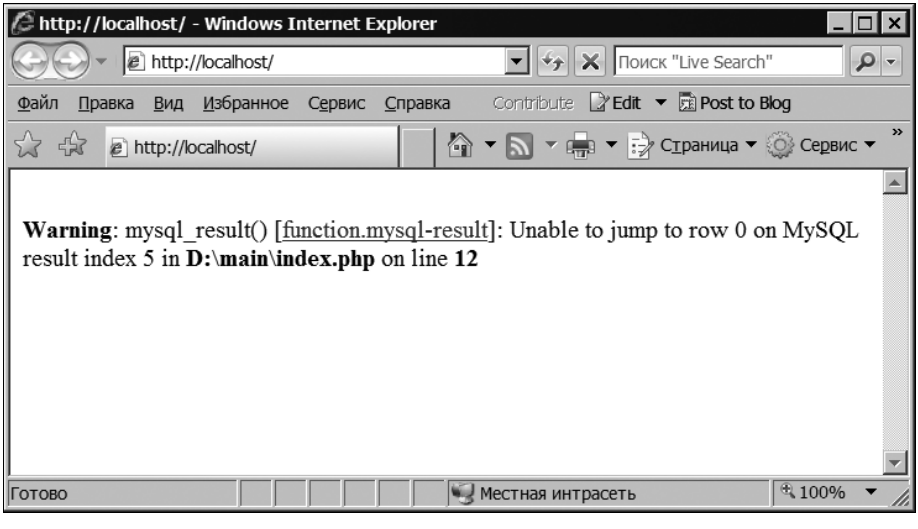


Рис. 8.4. Попытка извлечь несуществующую запись

Для предотвращения возникновения такой ошибки удобно воспользоваться функцией `mysql_num_rows()` (листинг 8.18).

Листинг 8.18. Использование функции `mysql_num_rows()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";
```

```
  // Формируем SQL-запрос
  $query = "SELECT name FROM test LIMIT 1";
  // Выполняем SQL-запрос
  $tst = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$tst) exit("Ошибка выполнения запроса — ".mysql_error());
  // Выводим результат, если в результирующей
  // таблице имеется хотя бы одна запись
  if (mysql_num_rows($tst) > 0)
  {
    echo mysql_result($tst,0);
  }
?>
```

8.6. Экранирование данных. SQL-инъекции

При работе с СУБД часто возникает задача экранирования данных, которые поступают извне. Пусть имеется таблица `userslist` с данными пользователей (листинг 8.19).

Листинг 8.19. Таблица `userslist`

```
CREATE TABLE userslist (
  id_user INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  pass TINYTEXT NOT NULL,
  email TINYTEXT NOT NULL,
  url TINYTEXT NOT NULL,
  PRIMARY KEY (id_user)
);
INSERT INTO userslist VALUES
(1, 'cheops', 'пароль', 'igor@softtime.ru', 'http://www.softtime.ru'),
(2, 'barton', 'пароль', 'barton@mail.ru', '');
```

Каждая запись таблицы `userslist` состоит из пяти полей:

- `id_user` — первичный ключ таблицы, обладающий атрибутом `AUTO_INCREMENT`;
- `name` — имя пользователя;
- `pass` — его пароль;
- `email` — адрес электронной почты пользователя;
- `url` — адрес домашней страницы пользователя.

Список пользователей можно вывести при помощи скрипта из листинга 8.20. Имя каждого пользователя подсвечивается:

```
<a href='user.php?id_user=1'>name</a>
```

Здесь `1` — первичный ключ записи в таблице `userslist`, а `name` — имя пользователя.

Листинг 8.20. Список пользователей (index.php)

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
  // Запрашиваем список всех пользователей
  $query = "SELECT * FROM userslist ORDER BY name";
  $usr = mysql_query($query);
  if (!$usr) exit("Ошибка выполнения запроса — ".mysql_error());
  while($user = mysql_fetch_array($usr))
  {
    echo "<a href='user.php?id_user=$user[id_user]'>$user[name]</a><br>";
  }
?>
```

На странице user.php происходит предоставление информации о каждом из посетителей. Код скрипта из файла user.php представлен в листинге 8.21.

Листинг 8.21. Подробная информация о пользователе (user.php)

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
  // Запрашиваем список всех пользователей
  $query = "SELECT * FROM userslist WHERE id_user = $_GET[id_user]";
  $usr = mysql_query($query);
  if (!$usr) exit("Ошибка выполнения запроса — ".mysql_error());
  $user = mysql_fetch_array($usr);
  echo "Имя пользователя — $user[name]<br>";
  if (!empty($user['email'])) echo "e-mail — $user[email]<br>";
  if (!empty($user['url'])) echo "URL — $user[url]<br>";
?>
```

Как видно из листинга 8.21, GET-параметр `id_user` вставляется в SQL-запрос без всякой дополнительной проверки. С одной стороны, это может приводить к ошибке запроса, если вместо числа параметр будет содержать строковые данные, с другой стороны, это может приводить к уязвимости типа SQL-инъекции, когда злоумышленник получает возможность выполнять на сервере собственный SQL-код.

Вместо числа в SQL-запрос

```
SELECT * FROM userslist WHERE id_user = 1
```

может быть внедрена произвольная строка. Так, если в строку запроса подставить значение `'какой-то%20текст'`, то скрипт выдаст сообщение об ошибке (рис. 8.5), т. к. будет осуществлена попытка выполнения запроса:

```
SELECT * FROM userslist WHERE id_user = 'какой-то текст'
```

Рис. 8.5. Передача через строку запроса произвольного текста

ЗАМЕЧАНИЕ

Символ `%20` обозначает пробел. Запоминать коды недопустимых символов необязательно, корректную для URL строку всегда можно получить, пропустив текст через функцию `urlencode()`.

Воспользовавшись конструкцией `UNION`, доступной, начиная с MySQL 4.0, можно добавить еще один запрос того же формата, что и первый (листинг 8.22).

Листинг 8.22. Использование конструкции `UNION`

```
SELECT * FROM userslist WHERE id_user = 1
UNION
SELECT * FROM userslist WHERE id_user = 2
```

Код SQL-инъекции выделен жирным шрифтом. Результирующая таблица будет содержать записи как из первого, так и из второго запроса. URL-запрос в этом случае может выглядеть следующим образом:

```
user.php?id_user=1%20UNION%20SELECT%20*%20FROM%20userslist%20WHERE%20id_
user%20=%202
```

Тем не менее, на странице по-прежнему выводится информация о посетителе с первичным ключом 1 (рис. 8.6).

Запрос из листинга 8.23 возвращает две записи, которые соответствуют пользователю с первичным ключом 1 и пользователю с первичным ключом 2.

Листинг 8.23. Записи, возвращаемые запросом из листинга 8.22

```
1, 'cheops', ******, 'cheops@mail.ru', 'http://www.softtime.ru'
2, 'barton', ******, 'barton@mail.ru', ''
```

Однако результаты выводятся только для самого первого запроса, т. к. в коде присутствует только один вызов функции `mysql_fetch_array()` и, следовательно, все

последующие записи игнорируются. Первой задачей злоумышленника является вывод в окно браузера результатов второго запроса из конструкции UNION, т. к. первый запрос изменить уже нельзя. Для реализации этой задачи можно воспользоваться двумя способами. Первый из них заключается в формировании такого условия первого запроса, который заведомо невыполним (листинг 8.24). Для этого переменной id_user можно присвоить отрицательное значение. Поскольку первичный ключ принимает только положительные значения, можно гарантировать, что первый запрос не вернет ни одной записи.

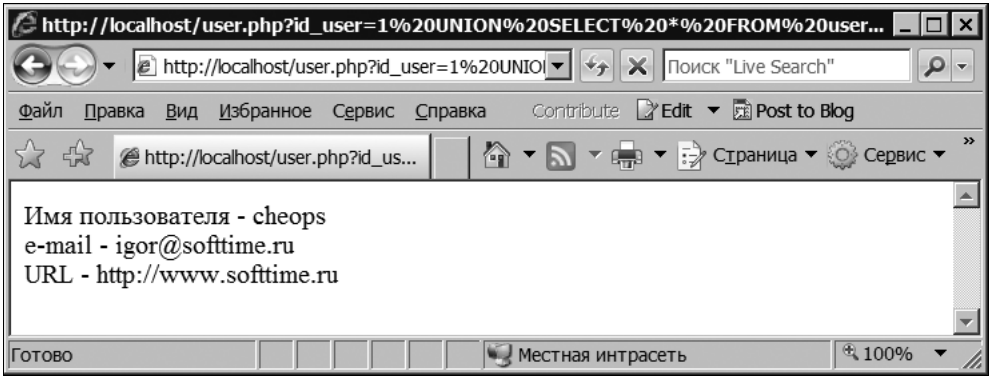


Рис. 8.6. Вывод информации о первом пользователе с первичным ключом 1

Листинг 8.24. Подавление вывода первого запроса

```
SELECT * FROM userslist WHERE id_user = -1
UNION
SELECT * FROM userslist WHERE id_user = 2
```

URL-запрос для SQL-инъекции из листинга 28.24 может выглядеть так, как это представлено ниже:

```
user.php?id_user=-1%20UNION%20SELECT%20*%20FROM%20userslist%20WHERE%20
id_user%20=%202
```

В результате такой инъекции в действие вступает второй SELECT-запрос из конструкции UNION, благодаря которому выводится информация о пользователе с первичным ключом 2 (рис. 8.7).

Иногда передать управление второму SELECT-запросу описанным приемом сложно, поэтому прибегают к конструкции ORDER BY, которая сортирует результаты запросов так, чтобы результаты из инъекционного запроса оказались в начале. В нашем случае можно подвергнуть результаты обратной сортировке по параметру id_user. Для осуществления такой операции необходимо использовать конструкцию ORDER BY id_user DESC, которая размещается в конце конструкции UNION (листинг 8.25).

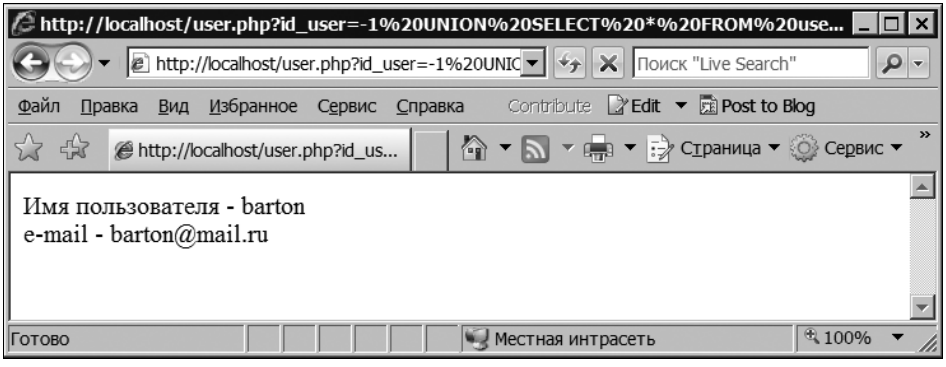


Рис. 8.7. Передача управления второму SELECT-запросу из конструкции UNION

Листинг 8.25. Сортировка результатов

```
SELECT * FROM userslist WHERE id_user = 1
UNION
SELECT * FROM userslist WHERE id_user = 2
ORDER BY id_user DESC
```

Добившись вывода результатов инъекционного SELECT-запроса, можно приступать к выводу пароля в окно браузера. Для этого следует расшифровать символ * во втором запросе (листинг 8.26).

Листинг 8.26. Расшифровка символа * в инъекционном запросе

```
SELECT * FROM userslist WHERE id_user = 1
UNION
SELECT id_user, name, pass, email, url FROM userslist WHERE id_user = 2
ORDER BY id_user DESC
```

Их количество должно совпадать с количеством столбцов из первого SELECT-запроса, иначе СУБД отклонит запрос как ошибочный. Однако столбцы name и pass являются текстовыми, поэтому ничто не мешает нам поменять их местами (листинг 8.27).

Листинг 8.27. Перемена местами полей name и pass

```
SELECT * FROM userslist WHERE id_user = 1
UNION
SELECT id_user, pass, name, email, url FROM userslist WHERE id_user = 2
ORDER BY id_user DESC
```

Так как имена столбцов формируются по первому запросу, вместо имени пользователя (name) подставляется его пароль (pass) (рис. 8.8).

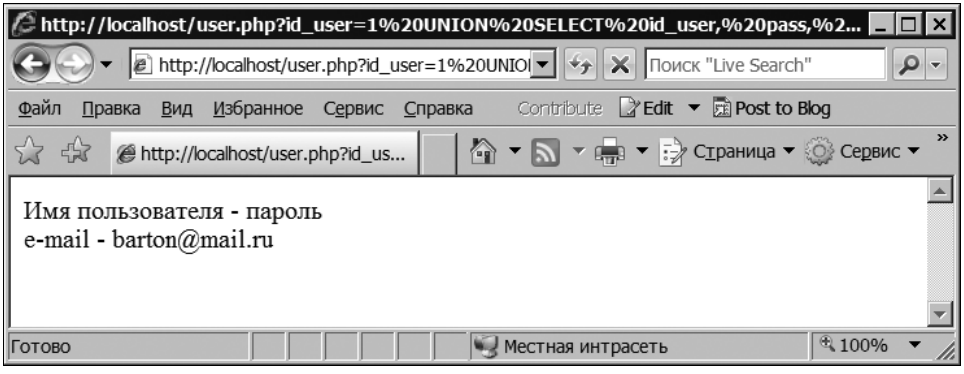


Рис. 8.8. Вывод пароля пользователя в окно браузера

ЗАМЕЧАНИЕ

Конструкция `UNION` применяется только совместно с `SELECT`-запросами, поэтому деструктивных действий такая SQL-инъекция не несет, если пароли не предоставляют доступ к панели управления, при помощи которой эти действия можно осуществить.

Для того чтобы избежать взлома по SQL-инъекции, следует проверять все числовые параметры при помощи регулярных выражений (листинг 8.28) или явно приводить их к числовой форме при помощи функции `intval()` (листинг 8.29).

Листинг 8.28. Предотвращаем SQL-инъекцию при помощи регулярных выражений

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Проверяем GET-параметр id_user, который должен
  // содержать только числа
  if (!preg_match("|^[\d]+$|",$_GET['id_user']))
  {
    exit("Недопустимый формат URL-запроса");
  }

  // Запрашиваем список всех пользователей
  $query = "SELECT * FROM userslist WHERE id_user = $_GET[id_user]";
  $usr = mysql_query($query);
  if (!$usr) exit("Ошибка выполнения запроса — ".mysql_error());
  $user = mysql_fetch_array($usr);
  echo "Имя пользователя — $user[name]<br>";
  if (!empty($user['email'])) echo "e-mail — $user[email]<br>";
  if (!empty($user['url'])) echo "URL — $user[url]<br>";
?>
```

Листинг 8.29. Явное преобразование GET-параметра к числу

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Проверяем GET-параметр id_user, который должен
  // содержать только числа
  $_GET['id_user'] = intval($_GET['id_user']);

  // Запрашиваем список всех пользователей
  $query = "SELECT * FROM userslist WHERE id_user = $_GET[id_user]";
  $usr = mysql_query($query);
  if (!$usr) exit("Ошибка выполнения запроса — ".mysql_error());
  $user = mysql_fetch_array($usr);
  echo "Имя пользователя — $user[name]<br>";
  if (!empty($user['email'])) echo "e-mail — $user[email]<br>";
  if (!empty($user['url'])) echo "URL — $user[url]<br>";
?>
```

Если вместо числового параметра используется текстовый параметр, то необходимо экранировать кавычки, которые могут нарушить синтаксис или послужить причиной SQL-инъекции. Ситуация усложняется тем, что на сервере может быть включен режим "магических кавычек", и он может самостоятельно пытаться произвести экранирование. В листинге 8.30 приводится типичный обработчик строковых параметров, который применяется перед тем, как использовать внешние данные в SQL-запросах.

ЗАМЕЧАНИЕ

Для экранирования кавычек используется функция `mysql_escape_string()`.

Листинг 8.30. Обработка строковых параметров

```
<?php
  ...
  if (!get_magic_quotes_gpc())
  {
    // Режим "магических кавычек" отключен
    $_POST['name'] = mysql_escape_string($_POST['name']);
    $_POST['pass'] = mysql_escape_string($_POST['pass']);
  }
  ...
?>
```

ГЛАВА 9

PHP и AJAX

В последнее время посещаемость сайтов только увеличивается, что в немалой степени связано с увеличением числа интернет-пользователей. Другой тенденцией является укрупнение проектов, успешные сервисы предоставляют довольно ресурсоемкие услуги. Все это привело к тому, что основной тенденцией последних лет стала попытка переложить часть вычислений с перегруженных серверов на клиента, чья машина большей частью простаивает. Клиентские технологии (CSS, JavaScript, Flash) получили мощный толчок в развитии.

В данной главе будет подробно рассмотрено взаимодействие PHP и JavaScript/jQuery, а также AJAX-приложения, работающие без перезагрузки страницы.

9.1. Что такое AJAX

Перезагрузка страниц для обновления содержимого раньше была одной из главных особенностей Web-приложений. Серверные и клиентские скрипты разделены в пространстве и времени, и к тому времени, когда начинает выполняться клиентский скрипт, серверный скрипт уже давно закончил работу и отправил результаты клиенту. Поэтому когда вся бизнес-логика расположена на сервере, какое бы действие ни требовалось совершить приложению, необходимо было отправить запрос серверу, который неминуемо сопровождался перезагрузкой страницы.

Традиционной клиентской технологией является язык программирования JavaScript, выполняющийся на стороне клиента после загрузки HTML-страницы. Обслуживая DOM-структуру HTML-документа, JavaScript позволяет добиться интерактивности и реакции на разнообразные события. Данная технология получила название DHTML. Сложность использования JavaScript заключается в том, что структура DOM у разных браузеров разная, поэтому в скриптах следует учитывать эти различия и использовать приемы, позволяющие скриптам выполняться одинаково во всех браузерах. Иногда это просто невозможно, в силу ограниченности того или иного браузера (часто устаревшей, но широко распространенной версии). Когда же подавляющее большинство браузеров получает возможность использовать новые возможности, совершается небольшая взрывообразная революция. Так вышло с

асинхронной отправкой запросов на сервер. Получив возможность отправлять запросы на сервер, получать и разбирать ответ, в фоновом режиме средствами JavaScript без перезагрузки страницы, сообщество Web-разработчиков окрестило новую возможность AJAX (Asynchronous JavaScript and XML). Следует отметить, что речь идет об обычных возможностях JavaScript, просто браузеры достигли такого уровня развития и поддержки DHTML, что очередной инструмент языка стало возможным использовать в массовом порядке. Упоминание языка разметки XML в расшифровке аббревиатуры также не должно смущать: несмотря на то, что JavaScript позволяет обмениваться с сервером в XML-формате, в подавляющем большинстве случаев эта возможность остается невостребованной.

9.2. Что такое jQuery

Приходится прикладывать немалые усилия для того, чтобы скрипты в разных браузерах вели себя одинаково. Синтаксис, обеспечивающий асинхронные запросы, различается настолько, что приводит к довольно объемному коду, который приходится оформлять в виде библиотеки.

Вопрос заключается в том, разрабатывать свою собственную библиотеку или воспользоваться готовой.

К плюсам сторонних профессиональных библиотек относится их ориентация на разработчика. Современные большие библиотеки предоставляют более удобный, по сравнению с классическим JavaScript, интерфейс, устраняют разночтения браузеров и предоставляют возможности, еще не реализованные в данной версии браузера (например, CSS 3).

Минусом сторонних библиотек является их довольно большой объем.

Плюсом собственной библиотеки является полный ее контроль, в свою собственную библиотеку можно включить только те функциональные возможности, которые будут реально применяться в приложении. В любой момент можно отредактировать библиотеку, что-то добавив или изменив ее поведение.

Минусом собственной разработки является потребность в довольно глубоком анализе особенностей JavaScript всех браузеров и всех их версий. Это может быть неприятным открытием для разработчика, специализирующегося на серверной компоненте и привыкшего к стабильному поведению языка программирования. Асинхронные запросы — это не единственная потребность современных приложений. При создании анимационных эффектов изменении оформления компонентов страницы могут потребоваться другие библиотеки. Создавать свою собственную мощную JavaScript-библиотеку может позволить себе не каждый. По крайней мере, мы себе этого позволить не можем, хотя бы потому, что книга посвящена главным образом PHP.

Если принимается решение об использовании сторонней JavaScript-библиотеки, вероятно, лучшим выбором будет библиотека jQuery, де-факто являющаяся стандартом. Получить последнюю версию библиотеки можно по адресу **http://docs.jquery.com/Downloading_jQuery**.

Библиотека jQuery позволяет выбирать из HTML-документа элементы по их селекторам, атрибутам, а также используя свойства каскадных таблиц стилей. В листинге 9.1 приводится пример окрашивания в синий цвет текста, заключенного в тег `<p>`, текста в тегах с классом `green` — в зеленый цвет, а текста с идентификатором `id` — в красный цвет.

ЗАМЕЧАНИЕ

Если использование библиотеки jQuery или JavaScript-скриптов вызывает у вас затруднения или вопросы, вы можете обратиться на наш форум **http://softtime.ru/forum/index.php?id_forum=4**.

Листинг 9.1. Использование библиотеки jQuery

```
<html>
  <head>
    <title>jQuery</title>
    <script type="text/javascript" src="jquery.js" ></script>
    <script type="text/javascript">
    $(document).ready(function() {
      $("p").css("color", "blue");
      $(".green").css("color", "green");
      $("#id").css("color", "red");
    });
    </script>
  </head>
  <body>
    <p>Тест будет окрашен в синий цвет.</p>
    <p class='green'>Тест будет окрашен в зеленый цвет.</p>
    <div class='green'>Тест будет окрашен в зеленый цвет.</div>
    <p id='id'>Тест будет окрашен в красный цвет.</p>
    <p>Тест будет окрашен в синий цвет.</p>
  </body>
</html>
```

Для того чтобы получить доступ к библиотеке jQuery, необходимо сослаться на файл jquery.js в теге `<script>`. Как видно из листинга 9.1, для доступа к элементам библиотеки используется специальная функция `$`, в качестве параметра которой передается строка, повторяющая синтаксис CSS-выражений: для тегов указываются их названия, CSS-классы предваряются точкой, а идентификаторы — символом диеза `#`. Метод `ready()` используется для того, чтобы код, расположенный внутри анонимной функции, выполнился только после загрузки документа. Метод `css()` позволяет установить для CSS-стиля (имя которого передается в первом параметре) значение (из второго параметра).

ЗАМЕЧАНИЕ

За полным описанием возможностей библиотеки следует обратиться к справочной информации на сайте **http://docs.jquery.com/**.

Атрибуты и селекторы можно комбинировать, как в обычной CSS-разметке, которую jQuery использует для выбора элементов. Например, чтобы обратиться ко всем div-тегам класса `green`, достаточно перечислить имя тега и класса через пробел: `$("div .green")`.

9.3. Обработка событий

Главной ценностью JavaScript-скриптов является возможность обработки событий пользователя и среды, такие как нажатие клавиши мыши или клавиатуры, окончание загрузки документа или потеря фокуса элемента управления. В листинге 9.2 приводится пример назначения обработчика события `click` (щелчок левой кнопки мыши) обычному тексту в теге `<p>` с идентификатором `id_text`. Щелчок по тексту приводит к смене цвета (с синего на красный, и наоборот). Для удобства CSS-свойство `cursor` устанавливается в значение `pointer`, характерное для ссылок и кнопок.

Листинг 9.2. Назначение обработчика средствами jQuery

```
<html>
  <head>
    <title>jQuery</title>
    <script type="text/javascript" src="jquery.js" ></script>
    <script type="text/javascript">
    $(document).ready(function() {
      // Меняем указатель курсора
      $("#id_text").css("cursor", "pointer");
      // Назначаем обработчик события click
      $("#id_text").bind("click", function(){
        if($("#id_text").css("color") == "#0000ff")
          // Если цвет синий — меняем на красный
          $("#id_text").css("color", "#ff0000");
        else
          // В противном случае назначаем синий цвет
          $("#id_text").css("color", "#0000ff");
      });
    });
    </script>
  </head>
  <body>
    <p id="id_text">Тест меняет цвет.</p>
  </body>
</html>
```

Как видно из листинга 9.2, событие привязывается к тегу с идентификатором `id_text` при помощи метода `bind()`, первым аргументом в котором передается на-

звание события, а вторым следует функция-обработчик. В листинге 9.3 приводится альтернативная реализация, где в качестве обработчика используется именованная функция `handler()`.

Листинг 9.3. Использование обычной функции в качестве обработчика

```
...
$(document).ready(function() {
   // Меняем указатель курсора
   $("#id_text").css("cursor", "pointer");
   // Назначаем обработчик события click
   $("#id_text").bind("click", handler);
 });
 function handler(){
  if($("#id_text").css("color") == "#0000ff")
    // Если цвет синий, меняем на красный
    $("#id_text").css("color", "#ff0000");
  else
    // В противном случае назначаем синий цвет
    $("#id_text").css("color", "#0000ff");
 }
```

События, которые используются в библиотеке jQuery, повторяют JavaScript-события, однако они не предваряются префиксом `on`. В табл. 9.1 приводится список наиболее популярных событий и их описание.

Таблица 9.1. *События JavaScript*

Событие	Описание
blur	Возникает при потере фокуса элементом
change	Возникает при выборе значения в выпадающем списке `<select>`. Для других элементов (главным образом, `<input>` и `<textarea>`) это событие возникает при потере фокуса, при условии, что с момента получения фокуса значение поля изменилось
click	Возникает при щелчке левой кнопкой мыши по элементу
dblclick	Возникает при двойном щелчке левой кнопкой мыши по элементу
error	Возникает, если происходит ошибка при загрузке изображения в теге `<img>`
focus	Возникает при получении элементом фокуса
keydown	Возникает при нажатии клавиши клавиатуры
keypress	Возникает при нажатии и отпускании клавиши клавиатуры
keyup	Возникает при отпускании клавиши клавиатуры
load	Возникает в момент полной загрузки документа
mousedown	Возникает при нажатии клавиши мыши

Таблица 9.1 *(окончание)*

Событие	Описание
mousemove	Возникает при перемещении курсора мыши
mouseout	Возникает при перемещении курсора мыши за границы элемента
mouseover	Возникает при перемещении курсора мыши над элементом
mouseup	Возникает при отпускании клавиши мыши
resize	Возникает при изменении размеров окна
scroll	Возникает при прокручивании окна
select	Возникает при выделении текста
submit	Возникает при отправке данных из HTML-формы
unload	Возникает при выгрузке документа или набора фреймов

9.4. Манипуляция содержимым страницы

При помощи методов библиотеки jQuery можно менять не только CSS-свойства, атрибуты, но также добавлять, удалять и изменять текст и теги на странице. В листинге 9.4 приводится пример изменения текста внутри тегов `<p>` на "доступно после регистрации". Для этого к полученному набору тегов применяется метод `html()`.

Листинг 9.4. Использование метода `html()` для изменения текста

```
<html>
  <head>
    <title>jQuery</title>
    <script type="text/javascript" src="jquery.js" ></script>
    <script type="text/javascript">
    $(document).ready(change_text);
    function change_text()
    {
      $("p").html("Доступно после регистрации");
    }
    </script>
  </head>
  <body>
    <p>Тест будет изменен.</p>
    <div>Тест останется без изменений.</div>
    <p>Тест будет изменен.</p>
  </body>
</html>
```

Одной из особенностей листинга 9.4 является тот факт, что вместо анонимной функции методу `ready()` передается функция `change_text()`. Иногда такой способ более предпочтителен и позволяет избежать повторения кода, например, в случаях, когда добавленным элементам необходимо назначить обработчики.

Рассмотрим задачу построения формы для загрузки произвольного количества файлов. Элемент управления `file` снабжается двумя кнопками: + для добавления новых элементов управления и − для удаления элементов управления, если добавлены лишние (рис. 9.1).

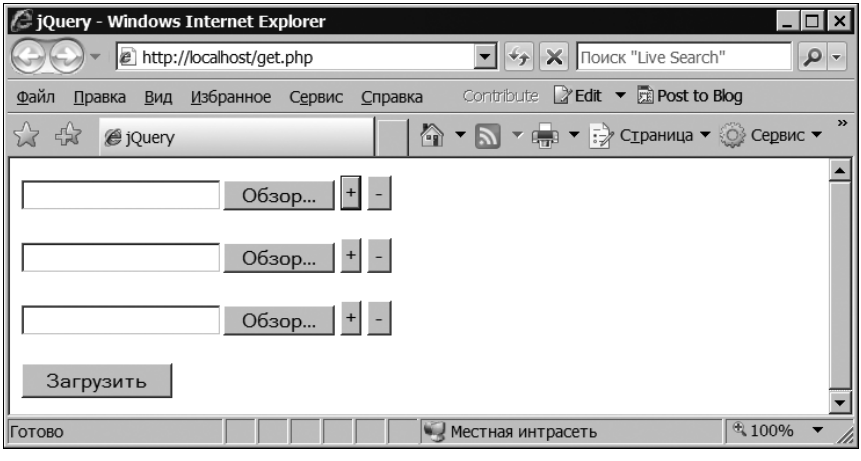


Рис. 9.1. Форма для загрузки произвольного количества файлов

В листинге 9.5 приводится пример построения такой формы средствами jQuery.

Листинг 9.5. Форма для загрузки произвольного количества файлов

```
<?php
  if(!empty($_FILES))
  {
    echo "<pre>";
    print_r($_FILES);
    echo "</pre>";
    exit();
  }
?>
<html>
  <head>
    <title>jQuery</title>
    <script type="text/javascript" src="jquery.js" ></script>
    <script type="text/javascript">
    // Назначить обработчики события click
    // после загрузки документа
    $(document).ready(handlers);
```

```
    // Назначаем обработчики кнопкам + и -
    function handlers()
    {
      // Назначаем обработчики события click
      $("input[type=button]").
        not("[value=+]").
        bind("click", remove_field);
      $("input[type=button]").
        not("[value=-]").
        bind("click", add_field);
    }
    // Обработчик для кнопки +
    function add_field(){
      // Добавляем новое поле в конец
      $("p:last").clone().appendTo("p:last");
      // Назначаем обработчик полям данного типа
      handlers();
    }
    // Обработчик для кнопки -
    function remove_field(){
      // Удаляем последнее поле
      $("p:last").remove();
    }
    </script>
  </head>
  <body>
    <form enctype='multipart/form-data' method="post">
    <p><input type="file" name="filename[]" />
       <input type="button" value="+">
       <input type="button" value="-"></p>
    <div><input type="submit" value="Загрузить"></div>
    </form>
  </body>
</html>
```

Как видно из листинга 9.5, изначально HTML-форма содержит лишь четыре элемента. Элемент управления типа `file` называется `filename[]`, и поскольку все динамические элементы будут иметь такое имя, необходимо оформить его в качестве массива (это позволит PHP сформировать суперглобальный массив `$_FILES` трехмерным). Две кнопки типа `button` со значением + и – обеспечивают добавление и удаление элементов управления с новыми строками. Последняя кнопка типа `submit` позволяет отправить файлы PHP-обработчику, расположенному в начале файла (в данном случае он не несет никакой полезной нагрузки, лишь выводит дамп полученного массива).

Клиентский код, обеспечивающий обработку клиентских событий, состоит из трех функций:

- `handlers()` — назначает обработчики для кнопок + и –, сразу после загрузки документа, а также при добавлении в форму новых кнопок + и –;
- `add_field()` — обработчик события нажатия на кнопку +, добавляет в конец формы строку с полем для загрузки файлов и две новые кнопки + и –;
- `remove_field()` — обработчик события нажатия на кнопку –, удаляет из конца формы строку полем для загрузки файлов и парой кнопок.

Функция `handlers()`, ответственная за назначение событий кнопкам, начинает поиск кнопок, выделяя все теги `input`, которые содержат атрибут `type` со значением `button`, из этой группы исключаются кнопки со значением атрибута `value`, равным `+` или `–`, в зависимости от того, какая группа кнопок нам нужна. Кнопкам со значением – в качестве обработчика назначается функция `remove_field()`, кнопкам со значением `+` — `add_field()`.

Функция `add_field()` при помощи селектора `p:last` ищет последний тег `p`, для которого делает копию `clone()`. Эта копия вставляется тут же при помощи метода `appendTo()`. Так как новым кнопкам `+` и `–` не были назначены события при загрузке структуры документа, в этой функции необходимо повторно вызвать метод `handlers()`.

Функция `remove_field()` находит последний тег `p` (`p:last`) и удаляет его.

Результат работы функции при загрузке трех изображений может выдать следующий дамп массива `$_FILES`.

```
Array
(
  [filename] => Array
    (
      [name] => Array
        (
          [0] => 1.gif
          [1] => 2.jpg
          [2] => 3.jpg
        )
      [type] => Array
        (
          [0] => image/gif
          [1] => image/jpeg
          [2] => image/jpeg
        )
      [tmp_name] => Array
        (
          [0] => C:\WINDOWS\Temp\php38CB.tmp
          [1] => C:\WINDOWS\Temp\php38CC.tmp
          [2] => C:\WINDOWS\Temp\php38CD.tmp
        )
      [error] => Array
```

```
        (
          [0] => 0
          [1] => 0
          [2] => 0
        )
      [size] => Array
        (
          [0] => 112425
          [1] => 6264
          [2] => 5055
        )
    )
  )
)
```

9.5. Асинхронное обращение к серверу

Наиболее простым методом для формирования AJAX-запроса к серверу является метод `load()`. В качестве первого параметра метод принимает запрашиваемый адрес, в качестве второго необязательного параметра принимает данные, которые передаются на сервер, а в качестве третьего необязательного параметра принимает JavaScript-функцию обратного вызова, которая вызывается после того, как данные загружены с сервера.

Для демонстрации принципов работы метода создадим простейшее приложение, в котором щелчком по ссылке будет загружаться текущее время с сервера (листинг 9.6). Разумеется, данное приложение не может похвастаться впечатляющей функциональностью и нужно лишь для демонстрации работы метода `load()`. Более сложные и полезные приложения будут рассмотрены в следующих разделах.

> ***ЗАМЕЧАНИЕ***
>
> В качестве кодировки данного, а также всех последующих скриптов используется UTF-8. При помощи AJAX можно работать с любой кодировкой и даже обойти проблемы Internet Explorer 6, который при неправильных настройках кодировок клиента и сервера просто перестает принимать ответы сервера, без выводов каких-либо ошибок, однако проще всего использовать UTF-8 и в таблицах, и в HTML-страницах. Такой подход позволяет исключить целый класс ошибок.

Листинг 9.6. Использование метода `load()` (index.php)

```
<html><head>
<title>Использование метода load()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/javascript" src="jquery.js" ></script>
<script type="text/javascript">
  // Назначить обработчики события click
  // после загрузки документа
```

```
  $(document).ready(function(){
    $("#id").bind("click", function(){
      $('#info').load("time.php");
    })
  });
</script>
</head>
<body>
<div><a href='#' id='id'>Получить время</a></div>
<div id='info'></div>
</body>
</html>
```

Как видно из листинга 9.6, ссылке с идентификатором `id` назначается обработчик события `click`, который в фоновом режиме загружает информацию со страницы time.php (листинг 9.7). Полученная строка выводится внутри div-тега с идентификатором `info`.

Листинг 9.7. Содержимое файла time.php

```
<?php
  echo date("d.m.Y H:i:s");
?>
```

9.6. AJAX-обращение к базе данных

Рассмотрим более сложный пример AJAX-загрузки данных. Создадим страницу, выводящую список пользователей в виде ссылок на информацию о пользователе, открывающуюся без перезагрузки страницы.

В листинге 9.8 приводится SQL-дамп таблицы `users`, которая будет использоваться для построения информационной системы.

Листинг 9.8. SQL-дамп таблицы `users`

```
CREATE TABLE users (
  id_position INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  pass TINYTEXT NOT NULL,
  email TINYTEXT NOT NULL,
  fname TINYTEXT NOT NULL,
  ffamily TINYTEXT NOT NULL,
  fpatronymic TINYTEXT NOT NULL,
  registerdate DATETIME NOT NULL,
  block ENUM('block','unblock') NOT NULL DEFAULT 'unblock',
  PRIMARY KEY (id_position)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8;
```

```
INSERT INTO users VALUES
(1, 'cheops', 'pass', 'igor@softtime.ru',
 'Игорь', 'Симдянов', 'Вячеславович',
 '2011-05-10 19:40:01', 'unblock'),
(2, 'makkuz', 'pass', 'kuznetsov@softtime.ru',
 'Максим', 'Кузнецов', 'Валерьевич',
 '2011-05-27 19:22:41', 'unblock');
```

Как видно из листинга 9.8, таблица `users` содрежит девять полей, которые имеют следующее значение:

- `id_position` — первичный ключ таблицы, снабженный атрибутом `AUTO_INCREMENT`, позволяющим автоматически назначать уникальное значение новым записям таблицы;
- `name` — имя для входа;
- `pass` — пароль для входа;
- `email` — электронный адрес пользователя;
- `fname` — имя пользователя;
- `ffamily` — фамилия пользователя;
- `fpatronymic` — отчество пользователя;
- `registerdate` — дата регистрации;
- `block` — статус пользователя, активен `unblock` или неактивен `block`.

Выведем текущих пользователей базы данных в виде списка ссылок, при этом каждую ссылку будем снабжать идентификатором `id` со значениями вида `usr1`, `usr2` и т. д., где номер будет совпадать со значением поля `id_position` в таблице `users` (листинг 9.9).

Листинг 9.9. Вывод информации о пользователе (index.php)

```
<html><head>
<title>Список пользователей</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<link href="index.css" rel="stylesheet" type="text/css" />
<script type="text/javascript" src="jquery.js" ></script>
<script type="text/javascript">
  // Назначить обработчики события click
  // после загрузки документа
  $(document).ready(function(){
    $("#list>div>a").bind("click", function(){
      // Формируем ссылку для AJAX-обращения
      var url = "user.php?id_position=" + this.id.substr(3,1);
      // Кодируем спецсимволы
      url = encodeURI(url);
```

```
      // Отправляем AJAX-запрос и выводим результат
      $('#info').load(url, null, $('#info').removeClass("hidden"));
    })
  });
</script>
</head>
<body>
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Выводим список пользователей
  $query = "SELECT * FROM users
            ORDER BY name";
  $usr = mysql_query($query);
  if(!$usr) exit("Ошибка выполнения запроса — ".mysql_error());
  if(mysql_num_rows($usr))
  {
    echo "<div id='list'>";
    while($users = mysql_fetch_array($usr))
    {
      echo "<div><a href='#' ".
           "id='usr".$users['id_position']."'>".
           htmlspecialchars($users['name'])."</a></div>";
    }
    echo "</div>";
  }
?>
<div id='info' class='hidden'></div>
</body>
</html>
```

Главная страница содержит два главных блока: первый div-блок с идентификатором `list` для вывода списка пользователей, второй div-блок с идентификатором `info` предназначен для вывода информации, полученной от сервера. Изначально результирующий div-блок `info` скрыт при помощи CSS-класса `hidden`:

```
.hidden {
  display: none;
}
```

При переходе по ссылке срабатывает обработчик события `click`, в результате чего осуществляется AJAX-запрос к странице user.php?id_position=N, где N — идентификатор пользователя в таблице `users`. После загрузки данных и помещения их в div-блок `info`, CSS-класс `hidden`, скрывающий этот блок, удаляется при помощи метода `removeClass()`. В листинге 9.10 приводится возможная реализация скрипта user.php.

Листинг 9.10. Извлечение информации о пользователе (user.php)

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
  // Обрабатываем GET-параметры
  $_GET['id_position'] = intval($_GET['id_position']);
  // Запрашиваем данные пользователя
  $query = "SELECT * FROM users
            WHERE id_position = {$_GET['id_position']}";
  $usr = mysql_query($query);
  if(!$usr) exit("Ошибка выполнения запроса — ".mysql_error());
  if(mysql_num_rows($usr))
  {
    // Извлекаем данные из результирующего набора
    $user = mysql_fetch_array($usr);
    // Обрабатываем данные перед выводом
    $user['name']        = htmlspecialchars($user['name']);
    $user['email']       = htmlspecialchars($user['email']);
    $user['fname']       = htmlspecialchars($user['fname']);
    $user['ffamily']     = htmlspecialchars($user['ffamily']);
    $user['fpatronymic'] = htmlspecialchars($user['fpatronymic']);
    // Формируем структуру ответа
    echo "<p>".
         "<span class='p'>Никнейм:</span>".
         "<span class='r'>{$user['name']}</span>".
         "</p>";
    echo "<p>".
         "<span class='p'>Email:</span>".
         "<span class='r'>{$user['email']}</span>".
         "</p>";
    echo "<p>".
         "<span class='p'>Имя:</span>".
         "<span class='r'>{$user['fname']}</span>".
         "</p>";
    echo "<p>".
         "<span class='p'>Фамилия:</span>".
         "<span class='r'>{$user['ffamily']}</span>".
         "</p>";
    echo "<p>".
         "<span class='p'>Отчество:</span>".
         "<span class='r'>{$user['fpatronymic']}</span>".
         "</p>";
  }
?>
```

Скрипт user.php получает GET-параметр `id_position` и извлекает соответствующую ему информацию из таблицы `user`. После этого формируются строки параметр/значение со следующей структурой:

```
<p><span>параметр</span><span>значение</span></p>
```

Такая организация данных, дополненная CSS-классами, позволяет позиционировать и оформлять данные в довольно широком диапазоне (рис. 9.2).

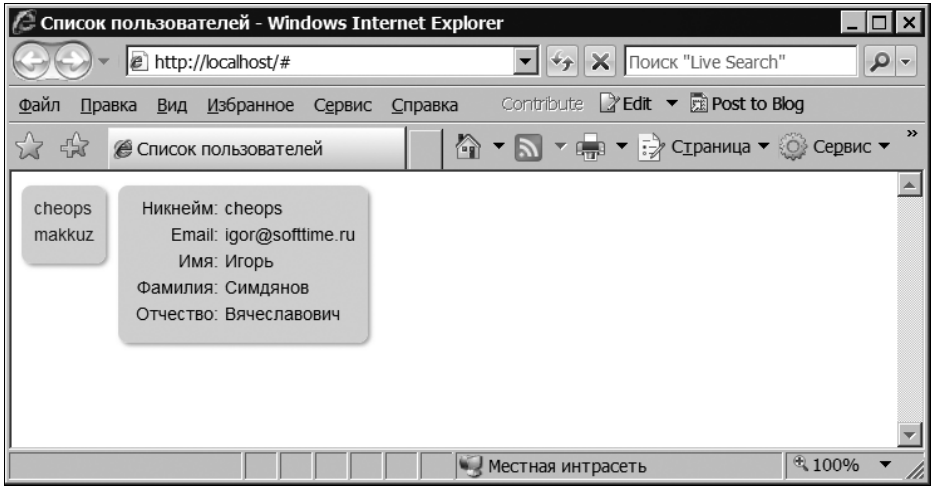


Рис. 9.2. Результат выполнения AJAX-приложения

Для оформления приложения необходим CSS-файл index.css, его возможная реализация приводится в листинге 9.11.

Листинг 9.11. CSS-оформление (index.css)

```
body {
  font-family: Arial, sans-serif;
  font-size: 14px;
}
#list {
  float: left;
  padding: 10px;
  background: #ddd;
  border-radius: 8px;
  box-shadow: 2px 2px 4px #bdbcb0;
  margin-right: 10px;
}
#list div {
  margin-bottom: 5px;
}
#list div a {
  color: #333;
```

```
  text-decoration: none;
}
#list div a:hover {
  text-decoration: underline;
}
#info {
  float: left;
  padding: 10px;
  background: #ddd;
  border-radius: 8px;
  box-shadow: 2px 2px 4px #bdbcb0;
}
#info p span.p {
  display: block;
  float: left;
  width: 70px;
  text-align: right;
  padding-bottom: 5px;
}
#info p span.r {
  display: block;
  float: left;
  text-align: left;
  padding: 0px 0px 5px 5px;
}
#info p {
  clear: both;
}
.hidden {
  display: none;
}
```

9.7. Отправка данных методом POST

Метод `load()` библиотеки jQuery позволяет осуществлять POST-запросы к серверу. Для демонстрации возможности отправки данных методом POST создадим HTML-форму размещения комментариев, которые будут сохраняться в таблице базы данных `comments` (листинг 9.12).

Листинг 9.12. Таблица `comments`

```
CREATE TABLE comments (
  id_position INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  comment TEXT NOT NULL,
  putdate DATETIME NOT NULL,
```

```
  hide ENUM('show','hide') NOT NULL,
  PRIMARY KEY (id_position)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Как видно из листинга 9.12, таблица `comments` содержит пять полей:

- `id_position` — первичный ключ таблицы, снабженный атрибутом `AUTO_INCREMENT`, позволяющим автоматически назначать уникальное значение новым записям таблицы;
- `name` — автор сообщения;
- `comment` — тест сообщения;
- `putdate` — дата размещения сообщения;
- `hide` — статус сообщения, поле принимает значение `'hide'` (для скрытых сообщений) и `'show'` (доступных для просмотра).

На рис. 9.3 приводится возможный результат выполнения Web-приложения.

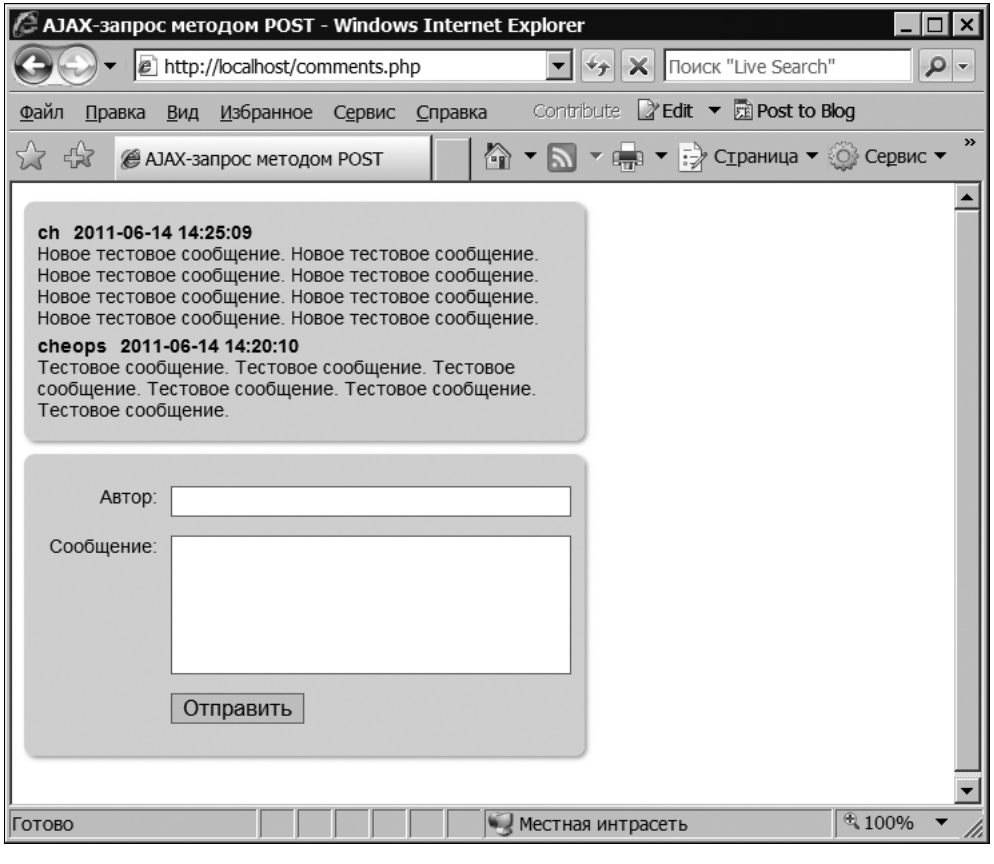


Рис. 9.3. Размещение комментариев методом AJAX

Точкой входа приложения будет файл comments.php, содержащий HTML-форму для размещения сообщений (div-блок form), а также ленту уже размещенных сообщений (div-блок info) (листинг 9.13).

Листинг 9.13. Форма размещения сообщений (comments.php)

```
<html><head>
<title>AJAX-запрос методом POST</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<link rel="stylesheet" type="text/css" href="comments.css" />
<script type="text/javascript" src="jquery.js" ></script>
<script type="text/javascript">
  // Назначить обработчики события click
  // после загрузки документа
  $(document).ready(function(){
    $("#id_submit").bind("click", function(){
      // Проверяем корректность заполнения полей
      if($.trim($("#id_name").val()) === "")
      {
        alert('Пожалуйста, заполните поле "Автор"');
        return false;
      }
      if($.trim($("#id_comment").val()) === "")
      {
        alert('Пожалуйста, заполните поле "Сообщение"');
        return false;
      }
      // Блокируем кнопку отправки
      $("#id_submit").attr("disabled", "disabled");
      // AJAX-запрос
      $("#info").load("addcom.php",
                      {name: $("#id_name").val(),
                       comment: $("#id_comment").val()},
                       $("#id_submit").attr("disabled", ""));
    })
  });
</script>
</head>
<body>
<div id='info'>
<?php
  require_once("addcom.php");
?>
</div>
<div id='form'>
<p>
  <span class='ttl'>Автор:</span>
```

```
  <span class='fld'>
    <input id='id_name' type='text' />
  </span>
</p>
<p>
  <span class='ttl'>Сообщение:</span>
  <span class='fld'>
    <textarea rows='5' id='id_comment' type='text'></textarea>
  </span>
</p>
<p>
  <span class='ttl'> </span>
  <span class='fld'>
    <input id='id_submit' type='submit' value='Отправить' />
  </span>
</p>
</div>
</body>
</html>
```

HTML-форма для отправки сообщений состоит из текстового поля для имени автора с идентификатором `id_name`, текстовой области для сообщения с идентификатором `id_comment` и кнопки для отправки сообщений с идентификатором `id_submit`. Именно этой кнопке назначается обработчик `click`, в котором проверяется корректность заполнения полей формы, и в случае успеха данные из них отправляются методом POST файлу addcom.php (листинг 9.14).

Для того чтобы отправить POST-запрос серверу, задействуется второй параметр метода `load()` библиотеки jQuery, которому передается объект, содержащий имена POST-параметров (`name` и `comment`), а также POST-данные, которые извлекаются из текстового поля `id_name` и текстовой области `id_comment` при помощи метода `val()`.

Полученный ответ (текущий список сообщений) размещается в div-теге с идентификатором `info`. Для предотвращения случайного дублирования данных, на время получения ответа сервера, кнопка отправки сообщений блокируется за счет изменения состояния атрибута `disabled`.

Листинг 9.14. Регистрация и вывод списка сообщений (addcom.php)

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
  // 1. Проверяем, переданы ли POST-параметры,
  // если ответ положительный, помещаем новое
  // сообщение в базу данных
  if(!empty($_POST))
  {
    $error = array();
```

```
  if(empty($_POST['name']))
  {
    $error[] = "Отсутствует автор";
  }
  if(empty($_POST['comment']))
  {
    $error[] = "Отсутствует сообщение";
  }
  // Если нет ошибок, помещаем сообщение
  // в базу данных
  if(empty($error))
  {
    // Если отключен режим "магических кавычек",
    // экранируем спецсимволы
    if (!get_magic_quotes_gpc())
    {
      $_POST['name'] = mysql_escape_string($_POST['name']);
      $_POST['comment'] = mysql_escape_string($_POST['comment']);
    }
    $query = "INSERT INTO comments
              VALUES (NULL,
                      '{$_POST['name']}',
                      '{$_POST['comment']}',
                      NOW(),
                      'show')";
    if(!mysql_query($query))
    {
      $error[] = "Ошибка размещения сообщения в базе данных";
    }
  }
}
// 2. Выводим сообщения в порядке убывания
// даты из размещения
$query = "SELECT * FROM comments
          WHERE hide = 'show'
          ORDER BY putdate DESC";
$com = mysql_query($query);
if(!$com) exit("Ошибка извлечения комментариев — ".mysql_error());
// Если имеется хотя бы одно сообщение,
// выводим его
if(mysql_num_rows($com))
{
  while($comments = mysql_fetch_array($com))
  {
    // Обрабатываем сообщения перед выводом,
    // чтобы исключить вставку JavaScript-кода
  $comments['name'] = htmlspecialchars($comments['name']);
```

```
    $comments['comment'] = nl2br(htmlspecialchars($comments['comment']));
      // Выводим сообщение
      echo "<div>".
           "<span class='author'>{$comments['name']}</span>".
           "<span class='date'>{$comments['putdate']}</span>".
           "<span class='comment'>{$comments['comment']}</span>".
           "</div>";
    }
  }
?>
```

Файл addcom.php условно можно разбить на две части: регистрация сообщений и вывод ленты сообщений. Если файл addcom.php получает какие-либо POST-параметры, считается, что происходит попытка добавить сообщение в базу данных. В любом случае скрипт возвращает список текущих сообщений, отсортированных по времени добавления.

9.8. Двойной выпадающий список

При формировании двойного выпадающего списка выбор раздела приводит к загрузке выпадающего списка с подразделами из следующих элементов управления.

Создадим таблицу `catalogs`, которая будут содержать разделы и подразделы системы с выпадающими списками (листинг 9.15).

Листинг 9.15. SQL-дамп таблицы `catalogs`

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  pos INT(11) NOT NULL,
  hide ENUM('show','hide') NOT NULL,
  id_parent INT(11) NOT NULL,
  PRIMARY KEY (id_catalog)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8;
INSERT INTO catalogs VALUES(1, 'Материнские платы', 1, 'show', 0);
INSERT INTO catalogs VALUES(2, 'Жесткие диски', 2, 'show', 0);
INSERT INTO catalogs VALUES(3, 'Видеокарты', 3, 'show', 0);
INSERT INTO catalogs VALUES(4, 'Процессоры', 4, 'show', 0);
INSERT INTO catalogs VALUES(5, 'ASUS', 1, 'show', 1);
INSERT INTO catalogs VALUES(6, 'Biostar', 2, 'show', 1);
INSERT INTO catalogs VALUES(7, 'Foxconn', 3, 'show', 1);
INSERT INTO catalogs VALUES(8, 'GIGABYTE', 4, 'show', 1);
INSERT INTO catalogs VALUES(9, 'Intel', 5, 'show', 1);
INSERT INTO catalogs VALUES(10, 'MSI', 6, 'show', 1);
INSERT INTO catalogs VALUES(11, 'Supermicro', 7, 'show', 1);
```

```
INSERT INTO catalogs VALUES(12, 'Hitachi', 1, 'show', 2);
INSERT INTO catalogs VALUES(13, 'Samsung', 2, 'show', 2);
INSERT INTO catalogs VALUES(14, 'Seagate', 3, 'show', 2);
INSERT INTO catalogs VALUES(15, 'Western Digital', 4, 'show', 2);
INSERT INTO catalogs VALUES(16, 'ASUS', 1, 'show', 3);
INSERT INTO catalogs VALUES(17, 'GIGABYTE', 2, 'show', 3);
INSERT INTO catalogs VALUES(18, 'MSI', 3, 'show', 3);
INSERT INTO catalogs VALUES(19, 'Sapphire', 4, 'show', 3);
INSERT INTO catalogs VALUES(20, 'AMD', 1, 'show', 4);
INSERT INTO catalogs VALUES(21, 'Intel', 2, 'show', 4);
```

Таблица `catalogs` предназначена для хранения древовидной структуры и содержит пять полей:

- `id_catalog` — первичный ключ таблицы, снабженный атрибутом `AUTO_INCREMENT`, позволяющим автоматически назначать уникальное значение новым записям таблицы;
- `name` — название раздела;
- `pos` — позиция подраздела относительно остальных подразделов данного уровня;
- `hide` — статус доступа для просмотра;
- `id_parent` — идентификатор родительского раздела, для корневых разделов принимает значение 0, для вложенных разделов поле принимает значение `id_catalog` родительского раздела.

Первый выпадающий список формируется из корневых разделов каталога и содержит четыре позиции: "Материнские платы", "Жесткие диски", "Видеокарты" и "Процессоры". При выборе любой из этих позиций при помощи AJAX-запроса загружается выпадающий список с подразделами (рис. 9.4).

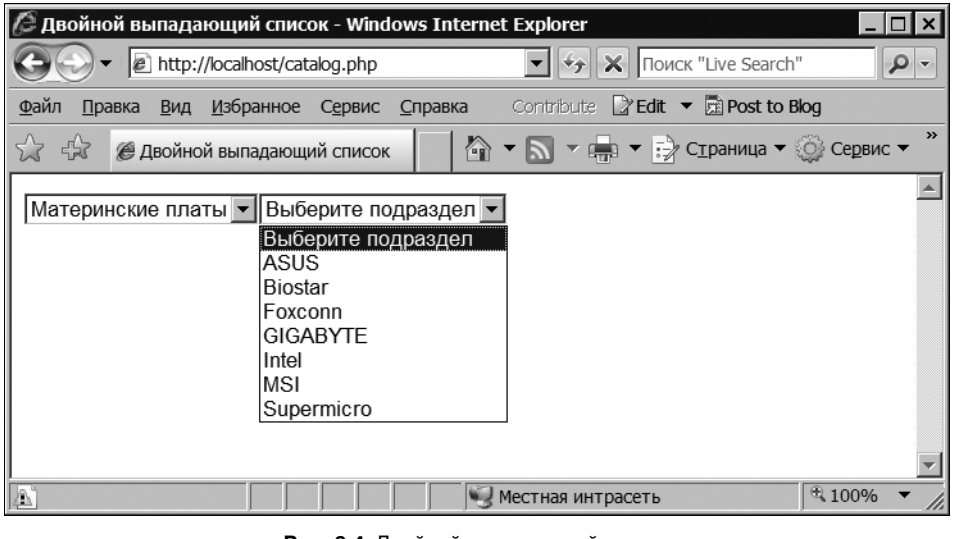


Рис. 9.4. Двойной выпадающий список

По умолчанию первый список с идентификатором `id_fst` активен, второй список с идентификатором `id_snd` дезактивирован при помощи атрибута `disabled`. Загрузка данных во второй выпадающий список привязана к событию `change` первого списка (листинг 9.16).

Листинг 9.16. Двойной выпадающий список (catalog.php)

```
<html><head>
<title>Двойной выпадающий список</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/javascript" src="jquery.js" ></script>
<script type="text/javascript">
  $(document).ready(function(){
    $("#id_fst").bind("change", function(){
      // AJAX-запрос
      $("#id_snd").load("select.php?id_catalog=" + $('#id_fst').val(),
                        null,
                        $("#id_snd").attr("disabled", ""));
    })
  });
</script>
</head>
<body>
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
  // Формируем выпадающий список корневых разделов
  $query = "SELECT * FROM catalogs
            WHERE id_parent = 0 AND hide = 'show'
            ORDER BY pos";
  $cat = mysql_query($query);
  if(!$cat) exit("Ошибка извлечения разделов — ".mysql_error());
  if(mysql_num_rows($cat))
  {
    echo "<select id='id_fst'>";
    echo "<option value='0'>Выберите раздел</option>";
    while($catalog = mysql_fetch_array($cat))
    {
      echo "<option value='{$catalog[id_catalog]}'>".
                                    "{$catalog[name]}</option>";
    }
    echo "</select>";
  }
?>
<select id='id_snd' disabled='disabled'>
<option value='0'>Выберите подраздел</option>
```

```
</select>
</body>
</html>
```

За формирование второго выпадающего списка ответственен скрипт select.php, который принимает GET-параметр `id_catalog` и формирует список подчиненных этому разделу подразделов (листинг 9.17).

Листинг 9.17. Формирование пунктов второго выпадающего списка (select.php)

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
  // Приводим значение GET-параметров к
  // целому значению
  $_GET['id_catalog'] = intval($_GET['id_catalog']);
  // Извлекаем подразделы
  $query = "SELECT * FROM catalogs
            WHERE id_parent = {$_GET['id_catalog']} AND
                  hide = 'show'
            ORDER BY pos";
  $cat = mysql_query($query);
  if(!$cat) exit("Ошибка извлечения подразделов — ".mysql_error());
  if(mysql_num_rows($cat))
  {
    echo "<option value='0'>Выберите подраздел</option>";
    while($catalog = mysql_fetch_array($cat))
    {
      echo "<option value='{$catalog['id_catalog']}'>".
                                    "{$catalog['name']}</option>";
    }
  }
?>
```

9.9. Запоминание состояний флажков

Еще одной распространенной задачей для AJAX-приложения может служить блок флажков, задающих настройки сайта. Пользователь получает возможность управлять состояниями флажков, изменяющих настройки сайта без перезагрузки сайта. Пусть состояния флажков по умолчанию сохраняются в сессии (в глобальном массиве `$_SESSION`). Внешний вид HTML-формы с флажками может выглядеть так, как это представлено на рис. 9.5.

Содержимое файла index.php, ответственного за вывод HTML-формы с флажками и отправку AJAX-запросов, представлено в листинге 9.18.

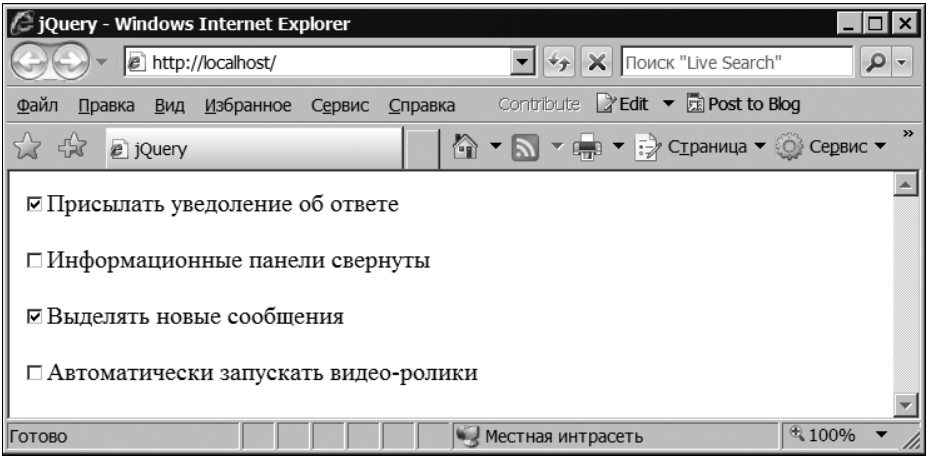


Рис. 9.5. AJAX-форма для запоминания состояния флажков

Листинг 9.18. Запоминание состояния флажков (index.php)

```
<?php
  // Инициируем сессию
  session_start();
?>
<html>
  <head>
    <title>jQuery</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf8" />
    <script type="text/javascript" src="jquery.js" ></script>
    <script type="text/javascript">
    $(document).ready(function(){
      $("input[type=checkbox]").bind("click", function(){
        $.post("check.php", {id: $(this).attr("id"),
                             status: $(this).attr("checked")});
      });
    });
    </script>
  </head>
  <body>
    <p><input id="id1" type="checkbox"
       <?php echo checkbox("id1"); ?> />Присылать уведомление об ответе</p>
    <p><input id="id2" type="checkbox"
       <?php echo checkbox("id2"); ?> />Информационные панели свернуты</p>
    <p><input id="id3" type="checkbox"
       <?php echo checkbox("id3"); ?> />Выделять новые сообщения</p>
    <p><input id="id4" type="checkbox"
       <?php echo checkbox("id4"); ?> />Автоматически запускать видеоролики</p>
  </body>
</html>
```

```
<?php
  function checkbox($key)
  {
    if(empty($_SESSION[$key])) return "";
    else return "checked='checked'";
  }
?>
```

Скрипт использует метод `post()` для отправки состояния выбранного флажка скрипту check.php (листинг 9.19). Первый параметр метода принимает имя файла, второй — JavaScript-объект, формирующий список POST-параметров. Всего передается два POST-параметра: `id` — идентификатор выбранного флажка и `status` — состояние флажка (отмечен флажок или нет).

Листинг 9.19. AJAX-обработчик состояния флажка (check.php)

```
<?php
  // Инициируем сессию
  session_start();
  // Изменяем состояние
  if($_POST['status'] == "true")
  {
    $_SESSION[$_POST['id']] = 1;
  }
  else
  {
    $_SESSION[$_POST['id']] = 0;
  }
?>
```

Обработчик check.php в зависимости от полученных данных меняет состояние массива `$_SESSION` для выбранного флажка. В зависимости от этого состояния функция `checkbox()` файла index.php выводит или не выводит атрибут `checked`, ответственный за статус флажка в HTML-форме.

ГЛАВА 10

Разные вопросы применения PHP

За годы развития Web-среды накопилось множество областей применения PHP, библиотек и отдельных приемов. Рассмотреть их все в одной книге не представляется возможным. Данная глава предназначена для частичной компенсации этого факта и содержит темы, которые не были освещены в предыдущих главах.

10.1. Локаль

В разных странах в силу сложившихся традиций имеются различия в представлении чисел, денежных сумм, времени и даты. Правила их представления для каждой страны называют *локальными настройками*, или сокращенно *локалью*. Каждая операционная система, будь то Windows или одна из UNIX-подобных ОС, поддерживает локальные настройки, которые позволяют изменять представление данных. Неправильно установленная локаль может служить причиной некорректной работы регулярных выражений и функций даты и времени. В PHP для установки локали используется функция `setlocale()`, которая имеет следующий синтаксис:

```
setlocale($category, $locale [, ...])
```

Функция возвращает значение только что установленной локали или `FALSE`, если операционная система не поддерживает работу с локальными настройками. В качестве первого аргумента `$category` функция принимает одно из значений табл. 10.1. В качестве второго аргумента выступает значение локали либо `NULL`, если установка нового значение не требуется и необходимо узнать текущие настройки.

***Таблица 10.1.** Переменные окружения, управляющие локальной настройкой*

Переменная окружения	Описание
`LC_COLLATE`	Определяет правила сравнения и сортировки строк для местного алфавита
`LC_CTYPE`	Определяет правила преобразования одиночных символов для местного алфавита. Позволяет правильно распознавать вид символа: цифра, буква, знак, верхний и нижний регистр
`LC_MONETARY`	Определяет правила национального представления денежных величин

Таблица 10.1 *(окончание)*

Переменная окружения	Описание
LC_NUMERIC	Определяет правила национального представления чисел с плавающей точкой
LC_TIME	Определяет правила национального представления даты и времени. Задает именование дней недели, месяцев, формат даты
LC_MESSAGES	Определяет формат информационных, диагностических и интерактивных сообщений операционной системы
LC_ALL	Особая переменная окружения, позволяющая устанавливать значения для всех перечисленных переменных

В листинге 10.1 приводится пример использования функции `setlocale()` для получения текущего значения локали.

Листинг 10.1. Получение текущего значения локали

```
<?php
  // Получение текущей локали
  echo setlocale(LC_ALL, NULL);
?>
```

К сожалению, значения переменных окружения не стандартизированы и в разных операционных системах могут принимать различные форматы, так для Windows с русскоязычной локалью скрипт вернет следующее значение:

```
Russian_Russia.1251
```

Для UNIX-подобных операционных систем локаль выглядит, как правило, следующим образом:

```
ru_RU.UTF-8
ru_RU.KOI8-R
ru_RU.CP1251
```

До точки располагается страна и язык, после точки — кодировка. В листинге 10.2 в качестве примера выставляется локаль для представления денежных величин.

Листинг 10.2. Установка локали для денежных величин

```
<?php
  // Получение текущей локали
  echo setlocale(LC_MONETARY, "ru_RU.CP1251");
?>
```

Функция `setlocale()` может принимать несколько значений через запятую. В листинге 10.3 осуществляется попытка установки русской локали, при этом функции передается несколько вариантов названия локали.

Листинг 10.3. Попытка установки русской локали

```
<?php
  $loc = setlocale(LC_ALL, 'ru', 'ru_RU', "rus", "russian");
  echo "На этой системе русская локаль имеет имя '$loc'";
?>
```

10.2. Сериализация

Сериализация — это процесс превращения сложной структуры (массива, объекта) в строку, которую легко сохранить на жесткий диск или передать по сети, чтобы впоследствии восстановить данную структуру.

Для сериализации в PHP используется функция `serialize()`, для восстановления данных из сериализованной строки — `unserialize()`.

Функция `serialize()` имеет следующий синтаксис:

```
serialize($value)
```

В качестве аргумента функция принимает массив или объект, возвращая его в виде закодированной строки. Симметричная ей функция `unserialize()` принимает в качестве аргумента закодированную строку, возвращая массив или объект.

```
unserialize($str)
```

В листинге 10.4 демонстрируется использование функций `serialize()` и `unserialize()`.

Листинг 10.4. Использование функций `serialize()` и `unserialize()`

```
<?php
  $poll[0] = 23;
  $poll[1] = 45;
  $poll[2] = 34;
  $poll[3] = 2;
  $poll[4] = 12;
  // Упаковываем массив в строку
  $str = serialize($poll);
  echo $str."<br>";
  // Извлекаем массив из строки
  $arr = unserialize($str);
  print_r($arr);
?>
```

Результатом работы скрипта из листинга 10.4 будут следующие строки:

```
a:5:{i:0;i:23;i:1;i:45;i:2;i:34;i:3;i:2;i:4;i:12;}
Array
(
    [0] => 23
```

```
    [1] => 45
    [2] => 34
    [3] => 2
    [4] => 12
)
```

Интересно отметить, что механизм сессий в PHP также использует для сериализации глобального массива `$_SESSION` функцию `serialize()`. Поэтому передача сериализованных строк через сессию может оказаться не очень хорошей идеей, т. к. зачастую невозможно восстановить сложный объект после повторного применения функции `serialize()`.

10.3. Уменьшение изображения

Среди расширений PHP одним из самых интересных является расширение GDLib, позволяющее манипулировать изображениями, в частности, создавать уменьшенные копии больших изображений. Создадим функцию `resizeimg()`, которая будет принимать в качестве аргументов:

- `$filename` — путь к увеличенному изображению;
- `$smallimage` — путь к уменьшенному изображению;
- `$w` — ширину уменьшенной копии;
- `$h` — высоту уменьшенной копии.

```
resizeimg($filename, $smallimage, $w, $h)
```

В листинге 10.5 приводится возможная реализация функции.

Листинг 10.5. Функция изменения размера изображения

```
<?php
  function resizeimg($filename, $smallimage, $w, $h)
  {
    // 1. Коррекция параметров $w и $h
    // Определим коэффициент сжатия изображения
    $ratio = $w / $h;
    // Получим размеры исходного изображения
    list($width, $height) = getimagesize($filename);
    // Если размеры меньше, то масштабирования не нужно
    if (($width < $w) && ($height < $h))
    {
      copy($filename, $smallimage);
      return true;
    }
    // Получаем коэффициент сжатия исходного изображения
    $src_ratio = $width/ $height;
```

```
    // Вычисляем размеры уменьшенной копии, чтобы при масштабировании
    // сохранились пропорции исходного изображения
    if ($ratio < $src_ratio) $h = $w/$src_ratio;
    else $w = $h*$src_ratio;

    // 2. Создание уменьшенной копии изображения
    // Создаем пустое изображение размером $w x $h пикселов
    $dest_img = imagecreatetruecolor($w, $h);
    // Открываем файл, который будет подвергаться сжатию
    $src_img = imagecreatefromjpeg($filename);

    // Масштабируем изображение
    imagecopyresampled($dest_img,
                       $src_img,
                       0,
                       0,
                       0,
                       0,
                       $w,
                       $h,
                       $width,
                       $height);
    // Сохраняем уменьшенную копию в файл
    imagejpeg($dest_img, $smallimage);
    // Чистим память от созданных изображений
    imagedestroy($dest_img);
    imagedestroy($src_img);
    return true;
  }
?>
```

Первая часть функции посвящена коррекции входных параметров `$w` и `$h`, задающих ширину и высоту уменьшенного изображения, чтобы оно не было искажено преобразованием (растянуто или сжато по вертикали или горизонтали). Вторая часть функции предназначена для создания уменьшенной копии `$smallimage` из исходного изображения `$filename`. Уменьшенное изображение создается при помощи функции `imagecreatetruecolor()`, которая возвращает дескриптор `$dest_img`. Дескриптор исходного изображения `$src_img` открывается при помощи функции `imagecreatefromjpeg()`. После этого исходное изображение переносится на уменьшенную копию при помощи функции `imagecopyresampled()`.

10.4. Водяные знаки на изображении

Для создания водяных знаков используют полупрозрачные изображения или полупрозрачные символы, которые рисуются средствами библиотеки GDLib. Для накладывания текста удобно воспользоваться готовым PNG-изображением, поддер-

живающим режим полупрозрачности. На рис. 10.1 приводится маска-изображение для защиты исходных изображений.

Рис. 10.1. Накладываемое изображение

Лучше всего нарисовать изображение в графическом редакторе (например, Adobe Photoshop), задав для него прозрачный фон, белый текст с прозрачностью 50%.

Для накладывания одного изображения на другое используется функция `imagecopy()`, которая имеет следующий синтаксис:

```
imagecopy($dst_im, $src_im, $dst_x, $dst_y, $src_x, $src_y, $src_w, $src_h)
```

Функция копирует область изображения `$src_im` на изображение `$dst_im`. Левый верхний угол копируемой области находится в точке с координатами `$src_x`, `$src_y`. Параметры `$src_w` и `$src_h` определяют ширину и высоту копируемой области соответственно. Скопированная область помещается на изображение `$dst_im` в точку с координатами `$dst_x`, `$dst_y`.

В листинге 10.6 представлен скрипт, накладывающий водяной знак из файла softtime.png на изображение img.jpg.

Листинг 10.6. Накладывание водяных знаков

```
<?php
// Имя файла
$filename = "img.jpg";
// Открываем защищаемый файл
$img = imagecreatefromjpeg($filename);
// Открываем маску
$filename = "softtime.png";
$msk = imagecreatefrompng($filename);
// Определяем размеры изображения
list($width, $height) = getimagesize($filename);
// Накладываем маску на изображение со
// смещение по оси Y на 50 пикселов вниз
imagecopy($img, $msk, 0, 50, 0, 0, $width, $height);

// Выводим изображение в браузер
header('Content-type: image/jpeg');
imagejpeg($img);
?>
```

Для удобства сравнения изображений поместим содержимое скрипта из листинга 10.6 в файл water_marks.php. Расположим два изображения рядом (листинг 10.7).

Листинг 10.7. Сравнение оригинального и защищенного изображений

```
<table>
  <tr align='center'>
    <td>Исходный</td>
    <td>Защищенный</td>
  </tr>
  <tr>
    <td><img width='200' src='img.jpg' /></td>
    <td><img width='200' src='water_mark.php' /></td>
  </tr>
</table>
```

На рис. 10.2 приводится результат работы страницы из листинга 10.7.

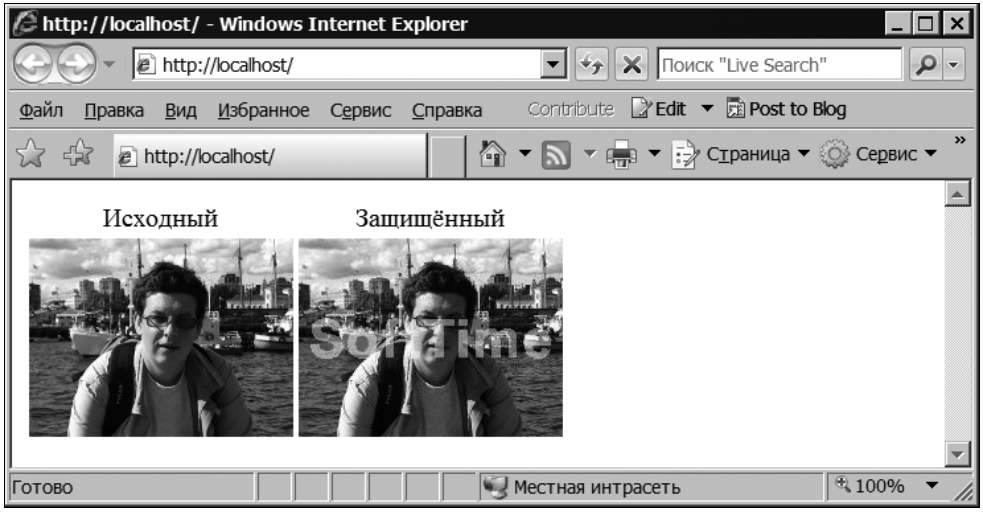


Рис. 10.2. Защита изображения водяным знаком

10.5. Запуск внешних программ

Чтобы выполнять внешние программы, команды операционной среды, получать результаты выполнения, в PHP предусмотрен специальный тип кавычек — обратные кавычки. В листинге 10.8 приводится пример использования обратных кавычек для вызова и получения результатов системной команды `dir` (или `ls` для UNIX-систем), которая выводит содержимое текущей папки.

Листинг 10.8. Использование обратных кавычек

```
<?php
  // echo `ls -l`; // UNIX
  echo `dir`;      // Windows
?>
```

Результат работы скрипта:

```
 Том в устройстве E имеет метку MEDIUM
 Серийный номер тома: EC68-EF90

 Содержимое папки E:\main

25.03.2004  20:41    <DIR>          .
25.03.2004  20:41    <DIR>          ..
23.10.2004  00:19               411 config.php
24.07.2004  12:43                54 index.php
03.07.2004  12:55             1 815 main.php
18.09.2003  19:43             2 789 top.php
25.03.2004  20:42    <DIR>          images
11.04.2004  16:30    <DIR>          Projects
25.03.2004  20:43    <DIR>          admin
30.03.2004  22:17    <DIR>          scripts
04.04.2004  12:27    <DIR>          Books
25.04.2004  12:52    <DIR>          tools
12.06.2004  09:46    <DIR>          test
26.09.2004  12:53    <DIR>          Site
               4  файлов          5 069 байт
              10 папок  18 157 846 528 байт свободно
```

Заключение

Чтобы ни говорилось, чтение книги — это всегда в том или ином виде слушание авторского монолога. В общем, это не очень хорошо, поскольку быстрая обратная связь в равной мере необходима как читателям, так и авторам. Чтобы авторский монолог превратить в полноценный диалог с читателем, специально для этой книги мы создали на своем сайте форум **http://www.softtime.ru/from/**. Авторы искренне надеются, что после того, как вы перевернете эту страницу, мы с вами не расстанемся, а встретимся на нашем форуме.

Предметный указатель

P

S

Б

В

Д

К

Л

М

П

Р

С

Т

Ф

X

Э

Я