On-line-поддержка книги

bhv

ЕЛЕНА БЕНКЕН

PHP, MySQL, XML

ПРОГРАММИРОВАНИЕ ДЛЯ ИНТЕРНЕТА

3-е издание

- Основы PHP
- PHP и MySQL
- Базовые сведения о XML
- PHP и XML
- Новостная лента RSS
- Генерация и обработка XML-документов
- Функции SimpleXML, DOM и SAX
- Русификация программного кода
- Практические примеры и задания для самостоятельной работы

+cd

AVALON.RU — СОВЕТУЮТ ПРОФЕССИОНАЛЫ

Елена Бенкен

PHP, MySQL, XML
ПРОГРАММИРОВАНИЕ ДЛЯ ИНТЕРНЕТА
3-е издание

Санкт-Петербург

«БХВ-Петербург»

2011

УДК 681.3.068+800.92
ББК 32.973.26-018.1
Б46

Бенкен Е. С.

Б46 PHP, MySQL, XML: программирование для Интернета. — 3-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2011. — 304 с.: ил. + (CD-ROM)

ISBN 978-5-9775-0724-0

Рассмотрено применение PHP для работы с базами данных MySQL и XML-документами. Описана установка и настройка сервера Apache с модулем PHP 5 и сервера MySQL 5. Изложены основы языка PHP и его расширения. Подробно излагается работа с базами данных MySQL от построения запросов до использования утилит командной строки. Приведены базовые сведения о языке XML. Описан формат новостной ленты RSS и представлены практические примеры обработки XML-документов с помощью расширений PHP 5, таких как SimpleXML, DOM-функциями и функциями событийного программирования SAX. В третьем издании внесены изменения, связанные с новыми возможностями языка PHP и сервера Apache, уделено особое внимание русификации программного кода. Компакт-диск содержит дистрибутивы Web-сервера, модуля PHP и сервера MySQL, распространяемые по лицензии GNU/GPL, а также примеры из книги.

Для Web-программистов

УДК 681.3.068+800.92
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Нина Седых*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Наталия Першакова*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.06.11.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 24,51.
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 978-5-9775-0724-0

Оглавление

Вступительное слово

Автор книги, Елена Сергеевна Бенкен, является высококвалифицированным специалистом и обладает многолетним успешным опытом преподавания. Это профессионал, который не только отлично разбирается в предметной области, но и обладает способностью увлечь аудиторию и донести свои знания до нее. Методика подачи предложенного в книге материала была апробирована на сотнях слушателей самых разных компаний, которые проходили обучение в нашем учебном центре. Благодаря этому при написании книги учитывалось, какой материал был наиболее тяжел для освоения слушателями, какие возникали проблемы, какие навыки будут важны для практической работы. Система изложения построена таким образом, чтобы читатели получили не только хорошо структурированный материал, но и приобрели реальные практические навыки. Поэтому в книге предусмотрены различные задания для самостоятельной работы.

От имени администрации и преподавательского состава учебного центра мы рекомендуем книгу Елены Сергеевны Бенкен как специалистам, так и начинающим изучение IT-технологий. Книга написана хорошим профессиональным языком, отличается последовательностью изложения, легко читается. Примеры и задания помогают в восприятии материала и иллюстрируют возможности практического применения полученных знаний. Книга Елены Сергеевны входит в серию книг, подготовленных преподавателями факультета переподготовки специалистов СПбГТУ (**www.avalon.ru**). Мы гордимся уровнем квалификации и профессионализмом наших преподавателей, написанные ими книги пользуются заслуженной популярностью, и поэтому решено было начать объединение этих книг под единой серией.

Как сотрудники крупного учебного центра мы понимаем, как важен контакт между преподавателем и обучаемыми, как необходима для успешного усвоения материала возможность задать преподавателю вопрос и получить квалифицированный ответ. Поэтому если после прочтения книги у вас появились вопросы, на которые вы не нашли ответов, если вы хотите получить еще больше практических заданий, чтобы повысить свою квалификацию, если вы просто хотите поблагодарить автора данной книги, то можете сделать это на форуме нашего учебного центра в соответствующем разделе: **http://forums.avalon.ru**.

На ваши вопросы ответит лично автор книги, а также другие преподаватели нашего учебного центра.

Директор ИГУИ СПбГПУ Александр Речинский

Об учебном центре

Институт государственного управления и информатизации Санкт-Петербургского государственного политехнического университета (известный в Интернете как AVALON.RU) представляет собой сегодня многопрофильный учебный центр, занимающий лидирующие позиции в области обучения информационным технологиям в Северо-Западном регионе. В 2010 году на различных образовательных программах у нас успешно прошли обучение более 7000 человек.

Благодаря высококвалифицированному преподавательскому составу, мощной технической базе и многолетнему опыту мы предлагаем действительно качественное обучение. Все наши преподаватели являются специалистами в своих областях и грамотными педагогами. Это подтверждается различными международными сертификатами учебного центра и преподавателей, а также тысячами положительных отзывов слушателей. За последние 10 лет наш учебный центр выработал определенный стиль, методику преподавания, у нас работает команда высококвалифицированных преподавателей. Основные направления деятельности:

- *"Академия информатики для школьников"*. Это программа обучения школьников информационным технологиям: базовый уровень, разработка программного обеспечения, сетевые технологии, дизайн. Общий объем программы составляет 16 семестров обучения. На сегодняшний день в программе участвуют около 700 школьников;
- *"Краткосрочные компьютерные курсы"*. На постоянной основе проводится более 30 курсов и семинаров различного уровня сложности по разным направлениям: офисное, интернет-технологии, компьютерный дизайн, 3D-графика, компьютерные системы и сети. Одним из направлений деятельности является комплексное обучение персонала по заказам компаний;
- *"Авторизованные и авторские курсы для IT-специалистов"*. Предлагается более 50 курсов и семинаров разного уровня сложности. Проводится обучение программным продуктам и технологиям как в ранге авторизованных курсов (Microsoft, Citrix, D-Link, Linux и т. д.), так и в виде авторских курсов (Cisco, Autodesk, Adobe, Oracle, Analog Devices, EMC и т. д.);
- *"Школа практического программирования"*. Совместный проект факультета с ведущими предприятиями города. Представляет собой систему тренингов, проводимых совместно преподавателями факультета и сотрудниками предприятий, с целью моделирования ситуаций работы над реальными проектами в области разработки программного обеспечения;
- *"Второе высшее образование в области информационных технологий"*. Обучение проводится в течение 2,5—3 лет по специальностям: "Математическое обеспечение и администрирование информационных систем", "Вычислительные машины, системы, комплексы и сети", "Дизайн";

- *"Второе высшее образование в экономике"*. Обучение проводится в течение 2,5—3 лет по специальностям: "Экономика фирмы", "Государственное и муниципальное управление";
- *"Высшее образование в области информационных технологий"*. Обучение проводится по специальности "Математическое обеспечение и администрирование информационных систем";
- *"Высшее образование в области экономики"*. Обучение проводится по специальности "Государственное и муниципальное управление";
- *"Авторизованное и авторское тестирование"*. Кроме авторизованного тестирования (авторизованный центр тестирования Prometric), проводится и авторское тестирование более чем на 100 различных тестах, разработанных преподавателями института.

Посвящаю, как и все, что я делаю,
моим детям Владимиру и Константину

Введение

Для кого написана эта книга

Книга предназначена для тех, кому надо самостоятельно освоить принципы работы Интернета, программирование на PHP и работу с базами данных, в первую очередь MySQL. Полезной книга может оказаться и тем, кому сегодня начальник сказал, что через месяц предстоит работа с XML-документами, причем данные из них придется помещать на Web-сайте вашей фирмы.

Предполагается, что читатель знаком с языком HTML, имеет опыт написания Web-страниц на HTML, а также твердо владеет основными пользовательскими навыками работы на компьютере с операционной системой Windows.

В Интернете имеется достаточно информации, чтобы научиться использовать любые технологии. Проблема в том, что документация по PHP и MySQL написана отнюдь не для новичков, не говоря уже о документации по языку XML.

Автор этой книги ежедневно занят ответами на вопросы студентов относительно Apache, PHP и MySQL. Поэтому есть некоторая надежда, что в книге удастся объяснить то, что чаще всего является камнем преткновения при первом знакомстве с предметом.

Книга предназначена для самостоятельного и независимого изучения материала. Это означает, что, скачав из Интернета необходимые пакеты, вы сможете установить их и начать работать, не заглядывая в другие книги.

Внимание! Ни в одной из глав не содержится исчерпывающей информации по обсуждаемому вопросу! Цель автора — научить читателя основам программирования на PHP так, чтобы он получал удовольствие от чтения купленной книги.

Итак, наша задача:

- установить Web-сервер Apache, модуль PHP и сервер MySQL в операционную систему Windows, научиться управлять этими серверами и узнать основы их администрирования;
- узнать принципы передачи информации в Интернете;
- научиться писать и запускать сценарии на PHP;
- познакомиться с основами SQL, научиться создавать базы данных, составлять запросы к ним и работать с базами данных с помощью PHP-сценариев;
- изучить основы языка XML, обрабатывать XML-документы с помощью PHP.

Как работать с книгой

Для полноценной работы вам потребуется компьютер с операционной системой Windows, на котором у вас есть право устанавливать программы. Кроме того, вам потребуется выход в Интернет, чтобы скачать оттуда дистрибутивы используемых программ.

В ходе чтения следует выполнять на компьютере примеры, описываемые в книге. Каждый пример стоит изменять и переделывать самому с тем, чтобы лучше понимать, как он работает или как сделать так, чтобы все перестало работать.

Автор приложил все усилия, чтобы изложить материал с наибольшей точностью, но не исключает возможности ошибок и опечаток. Автор также не несет ответственности за последствия использования сведений, изложенных в книге.

Источники информации

Главное, чему должен научиться человек, знакомящийся с интернет-технологиями, — это усвоению информации. Все остальное приложится.

Существуют два типа источников информации: первоисточники (если пользоваться термином прошедшей эпохи), содержащие исходные коды программ и скомпилированные из них пакеты, а также документация, поставляемая фирмой-изготовителем или созданная сообществами программистов. Скачивать программы следует только с серверов известных сообществ или фирм-производителей. С документацией сложнее: лучше читать ее в оригинале на английском языке, в имеющихся переводах порой теряется не только изящество изложения, но и смысл. К тому же документация написана для тех, кто уже много чего знает, а теперь решил изучить и этот вопрос. Ряд книг тоже можно отнести к первоисточникам — это книга для подготовки к сертификационному экзамену по PHP, а также написанные разработчиками PHP и MySQL. Правда, они страдают теми же недостатками, что и документация производителя.

Итак, первоисточники в Интернете:

- **http://apache.org**;
- **http://php.net**;
- **http://dev.mysql.com**;
- **http://w3schools.com** — школы W3C по XML;
- **http://www.w3.org/** — всеобъемлющая информация по стандартам Интернета;
- **http://xml.nsu.ru** — переводы школ консорциума W3C по XML.

Вторая категория источников информации — труды тех, кто работает с PHP и MySQL. Можно рекомендовать следующие русскоязычные сайты:

- **http://www.php.ru**;
- **http://php.su**;
- **http://php.net/manual/ru**.

Благодарности

- Моему мужу Александру, чья непоколебимая вера в меня помогла пережить немало сложных моментов, а также моим родителям.
- Константину Хоматьяно, познакомившему меня с PHP. Костя, я надеюсь на снисхождение.
- Егору Орлову — за идею книги и помощь в ее создании.
- Нашим студентам и слушателям — за удовольствие наблюдать огонь понимания и интереса, который зажигается в их глазах.
- Игорю Шишигину, проведшему автора по всем положенным кругам публикации книги.

ЧАСТЬ I

Основы языка PHP

Глава 1

Основы клиент-серверного взаимодействия в Интернете

При создании Web-страницы на HTML или сценария на JavaScript текст с разметкой и программным кодом сохраняется в файле с расширением html. Для того чтобы просмотреть результат работы, в Windows достаточно открыть страницу двойным щелчком по значку файла в окне Проводника. При этом страница, содержащая каскадные стили или сценарий на JavaScript, может быть и динамической, т. е. ее вид меняется в результате действий пользователя или просто с течением времени. Пользователь может просмотреть исходный код страницы и понять, в результате работы какого именно фрагмента кода происходят те или иные изменения внешнего вида страницы.

Но в Интернете есть множество страниц, работающих иначе. Например, на Web-сайте может проводиться опрос: "Какую марку автомобиля вы предпочитаете?" Ответ вписывается в поле формы. Введенный текст отправляется по щелчку на кнопку с надписью "Отправить ответ". После этих действий на экране появляется новая страница, на которой, например, написано, что этот вариант ответа выбрали еще 15% посетителей этого сайта, и выводится диаграмма распределения предпочтений посетителей. При этом в исходном коде страницы нет ничего поясняющего, откуда взялись эти сведения.

Выяснение, что же происходит в этом случае, начинается с изучения того, как браузер на пользовательском компьютере взаимодействует с сервером, с которого была получена страница с информацией о выборе марки автомобиля посетителями сайта.

Но сначала дадим определения некоторых понятий. Чем точнее сделаны определения, тем легче будет разобраться в обсуждаемом предмете.

Необходимые определения

Будем называть компьютер, работающий в Интернете, *хостом* (от англ. *host*).

Се́рвер (от англ. *serve* — служить) — это компьютер и программное обеспечение на нем, предоставляющие клиенту доступ к определенным ресурсам.

Web-сервер — это сервер, предоставляющий доступ к сайтам World Wide Web (WWW). Когда пользователь дает браузеру команду открыть тот или иной документ на сайте, браузер подключается к соответствующему серверу и запрашивает у него содержимое документа.

IP-адрес

Каждый хост характеризуется уникальным адресом. По действующему ныне стандарту этот адрес состоит из четырех целых положительных чисел, разделенных точками. Каждое число не может превышать 255. Заданный таким образом адрес называют *IP-адресом* хоста. Например: 192.168.1.66.

Если ваш домашний компьютер выходит в Интернет, то он получает IP-адрес от провайдера.

Запоминать IP-адреса трудно, да и нет необходимости: всем IP-адресам Web-серверов в Интернете однозначно ставится в соответствие *полное доменное имя*, например **www.avalon.ru**.

Среди IP-адресов есть специальные, например так называемый *адрес интерфейса обратной петли*. Это адрес, по которому каждый компьютер может обратиться к самому себе, используя сетевое программное обеспечение. Этим обращением можно проверить, функционируют ли программы поддержки работы с сетью.

Представьте себе, что вы ждете гостей, а их все нет и нет. Решив выяснить причину этого, вы начинаете с проверки, исправен ли звонок на входной двери. Выйдя на лестничную площадку, вы нажимаете кнопку звонка. На языке компьютерных сетей это и есть обращение к своему хосту по адресу интерфейса обратной петли.

Обратиться по этому адресу можно только к самому себе, как человек может обратиться на "я" только к себе. Для интерфейса обратной петли выделен адрес 127.0.0.1. Ему соответствует доменное имя localhost. Мы будем вводить в адресной строке браузера это имя для того, чтобы обратиться к Web-серверу, установленному на нашей машине.

Протокол

Браузер является программой-клиентом. Клиент посылает запросы серверу.

Программа-клиент взаимодействует с сервером, используя определенный протокол.

Протоко́л — это набор правил для передачи информации. Программа-клиент и программа-сервер могут работать как на одном компьютере, так и на разных.

Таким образом, клиентский и серверный компьютеры взаимодействуют друг с другом, используя адреса и доменные имена для поиска ресурсов.

Порт

На одном и том же сервере может работать несколько программ разного назначения. Для определения того, какая программа требуется клиенту, используется понятие порта. *Порт* — это номер, указывающий на программу, к которой хочет обратиться клиент. Например, Web-сервер обычно идентифицируется портом 80. В свою очередь, клиентский браузер тоже использует порт, но программе-клиенту выделяются порты с номерами, превышающими число 1024.

Информация о номере порта, по которому работает Web-сервер, вовсе не является избыточной для начинающего Web-программиста. Нам придется еще

не раз вспоминать об этом номере. Дело в том, что по каждому порту может работать только один сервер (иногда вместо слова "сервер" говорят "служба"). Если вы пытаетесь запустить вторую копию Web-сервера, то получите возмущенное сообщение системы с указанием того, что 80-й порт занят. Сообщение может быть неудобочитаемым, так что единственная ниточка, позволяющая вам понять, что происходит, — это указанный номер порта.

Итак, передача информации между клиентом и сервером осуществляется по определенным правилам — *протоколу*. Основным протоколом, по которому взаимодействуют компьютеры при запросе и получении Web-страницы, в настоящее время является HTTP-протокол (HyperText Transfer Protocol, протокол передачи гипертекста) версии 1.1.

Правила взаимодействия между компьютерами во Всемирной паутине регламентируются документами, издаваемыми консорциумом World Wide Web Consortium (W3C), осуществляющим стандартизацию средств Интернета.

Протокол HTTP

Полное описание HTTP содержится в спецификации, опубликованной на сайте **http://www.w3.org/Protocols/** или в RFC 2616 (**ftp://ftp.isi.edu/in-notes/rfc2616.txt**). В данной главе лишь кратко и неформально излагаются необходимые сведения.

Запрос клиента

Работая с браузером, пользователь набирает адрес интересующего его ресурса и дает команду выполнить запрос. В результате этих действий в браузере формируется запрос серверу, состоящий из нескольких строк.

Первая строка обычно начинается с команды `GET` или `POST`. Эти команды иначе называют *методами*. Оба метода используются для запроса страниц с Web-сервера, различия между ними мы кратко рассмотрим в этом разделе, а подробнее — в дальнейшем, когда вы сможете наблюдать их на примерах. Первая строка запроса может выглядеть так:

```
GET avalon.ru/index.html HTTP/1.1
```

Здесь `avalon.ru/index.html` — URI (Uniform Resource Identifier, универсальный идентификатор ресурса), иначе говоря, это адрес страницы.

После этой строки идут строки заголовков (Headers). Каждый заголовок имеет определенное стандартом имя, после которого ставится двоеточие, а после двоеточия пишется значение, которое требуется знать серверу. Эти значения передают серверу информацию о браузере: тип браузера, какие данные он может обработать, на каком языке и т. д. В зависимости от того, что сообщит о себе браузер, сервер и будет решать, какие данные следует отобрать для передачи клиенту.

Приведем примеры используемых заголовков.

Заголовок `User-Agent` сообщает о типе браузера клиента:

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.7.2) Gecko/20040803
```

В заголовке `Accept` указываются типы данных, которые следует передавать клиенту:

```
Accept: image/gif, image/jpeg, image/*, */*
```

Типы данных указываются в формате стандарта MIME.

MIME (Multipurpose Internet Mail Extensions) — многоцелевые расширения почтового стандарта Интернета. Этот стандарт описывает, как пересылать по электронной почте исполняемые, графические, мультимедийные и другие данные, не являющиеся простым текстом. Изначально MIME был создан для указания, какого типа документ вложен в сообщение электронной почты. MIME-тип задается в виде "тип/подтип". Например: `text/html`.

Если надо указать несколько MIME-типов, то они разделяются запятыми. Символ звездочки означает любой тип. В приведенном заголовке `Accept` типы данных указываются в порядке предпочтения браузера. То есть при наличии данных, представленных в различных форматах, в первую очередь должны передаваться графические данные в формате `image/gif`. Если данных в этом формате нет, то передаются данные в формате `image/jpeg`, в отсутствие которых можно передавать любой графический формат, а прочие данные браузер готов получать в любом виде.

Стандарт MIME определяет 7 типов данных:

- `application`;
- `audio`;
- `image`;
- `message`;
- `multipart`;
- `text`;
- `video`.

Заголовок `Referer` указывает на страницу, с которой пользователь перешел по ссылке на текущую. Конечно, такой ссылки могло и не быть, тогда этот заголовок не посылается. Пример:

```
Referer: www.host.ru/index.html
```

Каждый заголовок завершается символом конца строки (`\n`), а после заголовков идет пустая строка, т. е. два символа `\n`.

Если клиент делает запрос методом `GET`, то на этом запрос и заканчивается. Если же запрос идет методом `POST`, то после пустой строки могут идти данные, передаваемые в запросе. Например, это могут быть данные о предпочитаемой марке автомобиля или файл с фотографией, который вы закачиваете на сайт для размещения в фотоальбоме. В случае отправки таких данных необходимо передать их размер в соответствующем заголовке.

При использовании метода `GET` данные могут быть переданы прямо в адресной строке так:

```
GET avalon.ru/index/html?данные HTTP/1.1
```

Теперь давайте представим, что нам надо заполнить поле формы на Web-странице и отправить эти данные на сервер методом `GET`. Метод указывается в виде значения атрибута `method` в открывающем теге присланной нам формы. В полученной форме есть поле `login` (имя пользователя), мы заполняем его, нажимаем кнопку для отправки и видим в адресной строке браузера, что запрос идет в таком виде:

```
http://www.site.ru/index.html?login=Mike
```

Если же имя пользователя содержит пробел, русские буквы или специальные символы, то они кодируются перед отправкой во избежание ошибочных ситуаций в запросе.

Бродя по Интернету вы наверняка видели в адресной строке длинные адреса, включающие в себя символы %. Это и есть закодированные значения передаваемых параметров. При отправке методом GET нескольких параметров мы увидим в адресной строке примерно такую запись:

```
http://www.site.ru/index.html?login=Mike&age=25
```

При отправлении пар "параметр=значение" они разделяются символом &.

Ответ сервера

Рассмотрим теперь, как Web-сервер обрабатывает запрос клиента.

Web-сервер проверяет, есть ли в его распоряжении запрошенный ресурс, и имеет ли клиент право его получить. Если ресурс может быть предоставлен, то сервер отправляет клиенту ответ, начинающийся со статусной строки:

```
HTTP/1.1 200 OK
```

Здесь 200 — это код ответа, OK — расшифровка этого кода.

Если по каким-то причинам запрошенная страница не может быть передана, то код и расшифровка будут другими. Полный перечень кодов содержится в спецификации протокола HTTP, но в табл. 1.1 приведены некоторые часто встречающиеся варианты ответа.

Таблица 1.1. *Коды ответа сервера*

Код ответа	Описание
200 OK	Клиентский запрос успешен, и в Web-ответе сервера содержатся запрошенные данные
403 Forbidden	Запрос отклонен по причине, которую Web-сервер не хочет или не может сообщить клиенту
404 Not Found	Документ с указанным адресом не существует
500 Internal Server Error	Код указывает на возникновение аварийной ситуации в какой-то части сервера

Если документ найден и клиент имеет право его получить, то после строки статуса сервер посылает клиенту заголовочные данные о самом себе и о запрошенном документе, например:

- дата отправки документа: `Date: Fri, 22 Sep 2006 08:13:54 GMT`;
- тип передаваемого клиенту документа: `Content-type: text/html`;
- размер документа — количество байтов, пересылаемых в теле ответа:
  `Content-Length: 26457`;
- время последнего изменения пересылаемых данных:
  `Last-Modified: 22 Sep 2006`.

В случае если запрос оказался успешным, после заголовков Web-сервер отправляет клиенту запрошенные данные.

CGI

Большое количество Web-приложений основано на использовании внешних программ, управляемых Web-сервером. Такие приложения генерируют информацию динамически, выбрав ее из баз данных или других источников. Для связи между Web-сервером и вызываемыми программами используется стандарт CGI (Common Gateway Interface, общий интерфейс шлюза).

— это способ соединения или связи. Интерфейс определяет границу между сущностями. В нашем случае сущностями являются Web-сервер и PHP-программа, а интерфейс определяет способ их взаимодействия. Стандарт CGI разработан таким образом, чтобы для создания приложений можно было использовать любой язык программирования.

Программу, которая работает совместно с Web-сервером по стандарту CGI, принято называть *сценарием (скриптом)* или CGI-программой. Получив от клиента HTTP-запрос, Web-сервер определяет, что запрос адресован какому-либо сценарию. Например, в запросе клиента содержатся имя пользователя и пароль для регистрации на сервере. Web-сервер запускает запрошенный сценарий на исполнение. При запуске сценария сервер должен выполнить несколько операций: вызвать сценарий и обеспечить его необходимыми данными, посылаемыми от браузера, снабдить сценарий значениями переменных окружения. *Переменные окружения* содержат информацию о браузере, который посылает запрос, и о сервере, который его обрабатывает. Рассматриваемый нами сценарий должен проверить, что присланное имя пользователя (логин) зарегистрировано на сервере, а пароль указан верно. Сценарий читает параметры запроса и выполняет проверку, обращаясь к базе данных или к файлу с паролями. Результатом работы сценария является HTML-страница.

Сервер должен обработать результат работы скрипта, в том числе обеспечить включение дополнительной заголовочной информации, необходимой для браузера, чтобы тот мог успешно прочитать полученные данные. Web-сервер отправляет обработанный результат в ответ на запрос клиента. В странице ответа может быть указано, что пользователь прошел регистрацию и допущен к определенным ресурсам сервера.

Переменные окружения CGI чувствительны к регистру, и каждая из переменных определена в спецификации CGI. Вот некоторые из них.

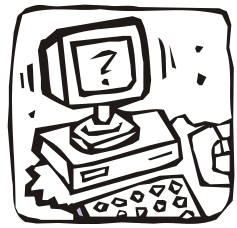
- Переменная `QUERY_STRING` используется скриптами для того, чтобы получить в текстовой форме информацию, которая следует справа от знака вопроса после URI переданного в запросе от пользователя скрипту для обработки. Например, при запросе вида:

  ```
  GET avalon.ru/index/html?a=1&b=3 HTTP/1.1
  ```

  значение переменной `QUERY_STRING = "a=1&b=3"`.
- Переменная `REQUEST_METHOD` используется для того, чтобы определить тип HTTP-запроса, который послан браузером серверу. Например, если браузер посылает запрос методом `GET`, то переменная окружения имеет значение `REQUEST_METHOD = GET`.

- Переменная `SCRIPT_NAME` используется для определения пути к скрипту, который будет запущен сервером. Например, если имеется URL **http://www.avalon.ru/our_site/somescript.php**, то переменная окружения примет следующее значение:
  ```
  SCRIPT_NAME = our_site/somescript.php
  ```
- Переменная `HTTP_ACCEPT` служит для того, чтобы определить, какие MIME-типы может принимать браузер. Они указаны в HTTP-заголовках, которые браузер послал серверу. Например, переменная окружения может принимать следующее значение:
  ```
  HTTP_ACCEPT = audio/mpeg, text/html, text/plain
  ```
- Переменная `HTTP_USER_AGENT` используется для того, чтобы идентифицировать тип браузера, который делает запрос серверу. Значение переменной получается из заголовков запроса клиента. Например:
  ```
  HTTP_USER_AGENT = Mozilla/2.01 Gold(Win95PE)
  ```

Данные из HTML-формы передаются в сценарий в виде набора переменных. CGI-сценарий использует эти данные при выполнении программы.



Глава 2

Установка Web-сервера Apache 2.2 и модуля PHP 5 в Windows

Данная книга рассчитана на среднестатистического российского Web-программиста, который разрабатывает сценарии для сайта на компьютере под управлением операционной системы Windows (вследствие ограниченной функциональности не подходит версия Windows XP Home Edition). Созданные сценарии переносятся на сервер хостера, работающий под управлением одной из систем семейства UNIX.

Преподавательский опыт автора позволяет настаивать на том, чтобы читатель ставил и использовал именно те программные продукты, которые описаны в этой книге. Установка различных пакетов, содержащих и Web-сервер, и сервер баз данных, и текстовый редактор, приводит к тому, что написать несложный код на PHP начинающий Web-мастер может, а вот понять, как он работает, — нет. Кроме того, разобраться, как при необходимости изменить настройки этих пакетов, оказывается сложно, к тому же навыки такой настройки мало что дают, т. к. на рабочих серверах они выполняются иначе. Web-технологии непросты, и попытки их упростить приводят к скверному результату.

При установке программ вы должны работать от имени пользователя, имеющего достаточные привилегии, чтобы писать файлы в системный каталог Windows и каталог Program Files системного диска.

Установка сервера Apache

Для установки Web-сервера Apache необходимо скачать дистрибутив со страницы **http://httpd.apache.org/download.cgi**. Нам потребуется бинарный пакет (Win32 Binary without crypto) в файле с расширением msi (MS Installer).

> ***Примечание***
>
> При скачивании пакетов используйте только проверенные сайты, этим вы обеспечите безопасность своего компьютера, а также сможете быть уверены, что скачали самые свежие версии программного обеспечения.

Приведенные далее инструкции касаются установки сервера Apache 2.2.

После запуска на исполнение пакета с дистрибутивом вам потребуется ответить на ряд вопросов (рис. 2.1).

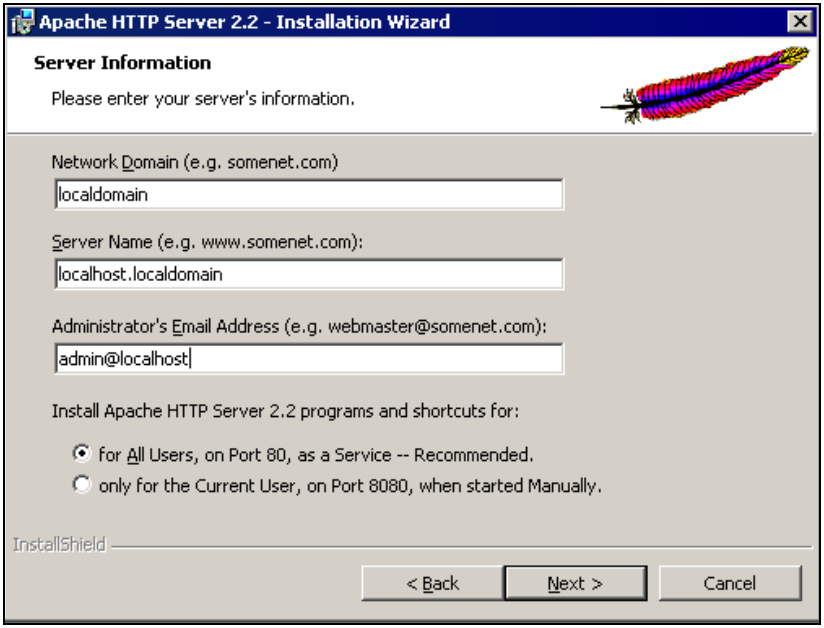


Рис. 2.1. Запрос информации о сервере

Если вы работаете дома, то, возможно, у вас нет ни сети, ни, следовательно, домена, в который может входить ваш компьютер. Тогда в первом поле следует указать значение `localdomain`.

Если же ваш компьютер работает в сети, то узнайте у администратора сети название домена и укажите его здесь. Впрочем, если вы работаете в домене, то инсталлятор может и сам вписать имя домена в поле ввода.

Во втором поле на рис. 2.1 вам надо указать имя сервера. Таким именем может быть либо полное доменное имя компьютера, либо его IP-адрес, либо специальное имя `localhost`.

В третьем поле следует указать электронный адрес администратора сервера — в зависимости от ситуации вы можете ввести подлинный или вымышленный (последний вполне годится для домашнего сервера, который кроме вас никто не увидит).

Наконец, надо разрешить запускать Apache автоматически при запуске системы как службу Windows и открыть 80-й порт для работы с сервером.

Затем придется ответить на вопрос о том, в какой каталог следует установить Apache. Инсталлятор предложит каталог Program Files на системном диске (рис. 2.2.). Это не очень удобно, но всякие изменения в этом пункте приводят к неоправданным для новичка сложностям.

После прохождения всех окон вы увидите, как установится пакет, а затем и запустится. Об успешности запуска надо судить по черному DOS-окну, в котором могут появиться ошибки (рис. 2.3). Если ошибки не появились — отлично, черное окно закроется, что означает успешный запуск сервера.

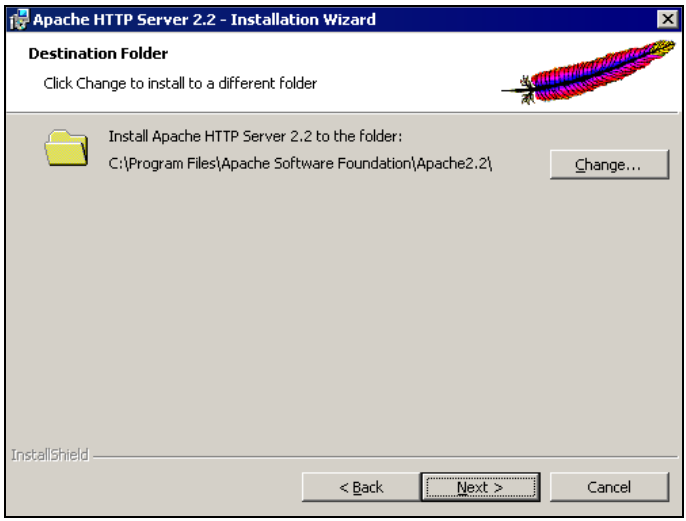


Рис. 2.2. Выбор каталога, в который будет установлен сервер

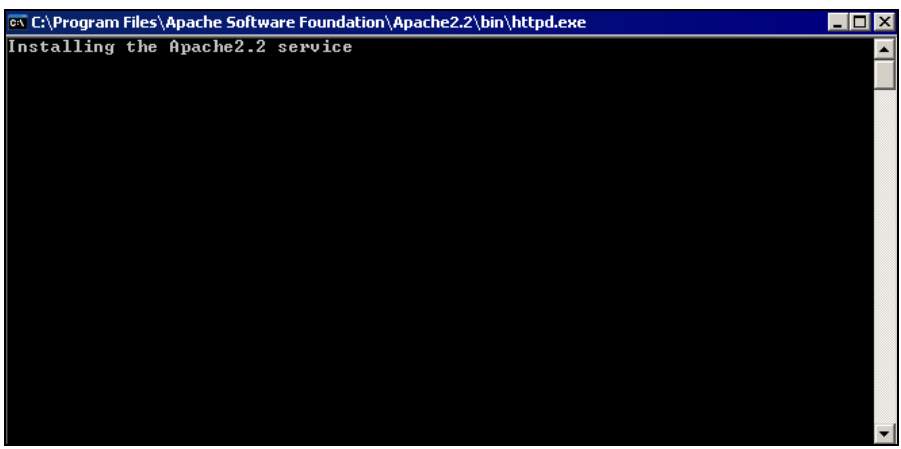


Рис. 2.3. Окно запуска сервиса Apache

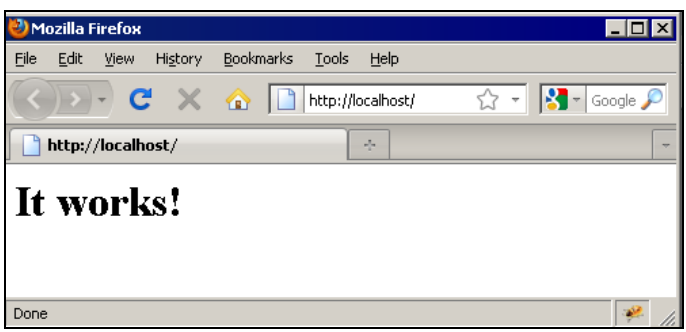


Рис. 2.4. Стартовая страница сайта на сервере Apache

Откройте окно браузера и отправьте HTTP-запрос вашему серверу, указав адрес: **http://localhost**. Вы должны увидеть то, что показано на рис. 2.4.

Теперь посмотрите, какие процессы запустились в системе после установки Apache: нажмите комбинацию клавиш <Ctrl>+<Alt>+<Del> и в окне Диспетчера задач Windows выберите вкладку **Процессы.**

Вы обнаружите несколько процессов, связанных с Apache. Apachemonitor — это процесс, слушающий порт 80 и ожидающий клиентские запросы. На эти запросы он не отвечает сам, а передает их дочерним процессам Apache, которые и занимаются их обработкой. Таких дочерних процессов запускается несколько — в расчете на большое количество запросов клиентов. Монитор следит за тем, чтобы количество дочерних процессов всегда было достаточным с учетом загрузки Web-сервера, и при необходимости создает новые дочерние процессы или удаляет лишние.

Посмотрим теперь, какие каталоги мы создали в процессе установки и что в них находится. Зайдите в каталог Program Files на системном диске и найдите в нем каталог Apache Software Foundation, а в нем каталог Apache2.2. Каталог bin внутри Apache2.2 содержит исполняемые модули сервера, некоторые библиотеки и вспомогательные программы. Каталог htdocs — место хранения HTML-страниц и сценариев на языке PHP. Содержимое только этого каталога может передаваться клиенту по запросу.

Директивы конфигурации Apache

Каталог conf содержит конфигурационный файл httpd.conf сервера Apache. Apache читает этот файл при запуске. Строки конфигурационного файла, начинающиеся символами `#`, — это комментарии, которые Apache не читает. Файл httpd.conf содержит директивы, определяющие режим работы Apache.

Конфигурирование сервера Apache 2.2 значительно сложнее, чем в предыдущих версиях. Кроме основного файла конфигурации появились небольшие файлы, разложенные по различным каталогам внутри каталога conf и подключаемые в основной файл с помощью директив `Include`. В связи с чем начинающим пользователям трудно уловить логику конфигурации и для начала надо ограничиться набором установленных файлов и настройкой одного-единственного параметра — кодировки отправляемых клиенту сообщений.

Рассмотрим некоторые важные директивы конфигурационного файла.

- Директива `ServerAdmin` получает в качестве значения адрес администратора сервера (**you@your.address**), который будет появляться в сгенерированных сервером сообщениях об ошибках.
- Директива `ServerName` задает имя хоста, на котором работает Web-сервер. Это должно быть зарегистрированное доменное имя. На домашнем компьютере в качестве имени хоста можно указать localhost.
- В директиве `DocumentRoot` задается каталог, из которого берутся передаваемые клиентам документы.
- Директива `DirectoryIndex` указывает имя файла, используемого в качестве страницы-указателя или оглавления. Например:

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

Эта директива позволяет задать название документа, возвращаемого по запросу, который не содержит в строке URL названия документа. Например, в адресе **http://www.shop.com/** отсутствует название документа, и будет возвращен документ, указанный в директиве `DirectoryIndex`. Поскольку в директиве указано имя файла index.html, сервер передаст клиенту документ index.html из каталога DocumentRoot на сервере.

В директиве `DirectoryIndex` можно задать несколько имен файлов. Если первый документ, указанный в строке, не найден в каталоге, то сервер ищет следующий документ и, в случае успеха, передает его клиенту.

Эта настройка также предоставляет важный уровень защиты информации. По умолчанию, если клиент указывает в адресной строке браузера адрес каталога, то сервер Apache передает в ответ список файлов, имеющихся в этом каталоге. Создав в каталоге индексный файл, вы лишите пользователей возможности получить список всех файлов в этом каталоге.

Для разработчика наличие индексного файла неудобно, поэтому удалите из каталога htdocs все индексные файлы. Обратившись после этого к своему серверу, вы увидите только пустой каталог без каких-либо документов в нем.

- Директива `AddDefaultCharset` позволяет генерировать в ответе сервера указание браузеру переключиться в заданную кодировку.
- Директива `AddType` полезна для добавления новых типов предоставляемых клиентам документов на основе использования MIME-типов. Вам может потребоваться указать серверу, что документ является PHP-сценарием (судя по расширению имени файла) и нужно использовать модуль PHP для его обработки. Директива `AddType` дает такую возможность.
- При помощи директивы `ErrorLog` задается местоположение журнальных файлов, в которых регистрируются ошибки доступа к серверу. В файле, указанном в директиве `ErrorLog`, сервер сохраняет сообщения диагностики, включая сообщения об ошибках, выдаваемые сценариями CGI.
- Директива `CustomLog` указывает на расположение и формат журнальных файлов, в которых регистрируются все обращения к Web-серверу.
- Пути отсчитываются от каталога, указанного в директиве `ServerRoot` (C:\Program Files (x86)\Apache Software Foundation\Apache2.2).
- Формат журнальных файлов задается директивой `LogFormat`. Полное описание формата вы найдете в документации на Apache, поставляемой вместе с сервером, а директива `CustomLog` указывает, какой формат использует журнальный файл. Например:

```
CustomLog logs/access.log common
```

Здесь указано, что журнальный файл access.log имеет формат common.

Подробное описание Apache на английском языке поставляется вместе с сервером и доступно по адресу **http://localhost/manual**, и это лучшее, что можно прочитать об Apache.

Итак, единственное, что надо сделать для начала, это дописать в конец файла httpd.conf директиву `AddDefaultCharset`, отвечающую за выбор кодировки документа по умолчанию. Это можно сделать и более элегантным способом, чем описано далее,

вы о нем сами догадаетесь по прошествии времени, когда наберете некоторый опыт работы с Apache, но сейчас приведем самый простой способ достичь желаемой цели.

Установите значение `AddDefaultCharset` равное `utf-8`, добавив эту директиву в конец конфигурационного файла. Сохраните файл httpd.conf и перезапустите сервер Apache, используя для этого пункт системного меню **Старт | Все программы | Apache HTTP Server 2.2 | Control Apache server | Restart**.

Установка модуля PHP

Дистрибутивы PHP 5 следует копировать с сайта **www.php.net**. Для установки PHP 5 под Windows потребуется инсталлятор с расширением msi. Если же после установки модуля выяснится, что вы забыли установить какие-либо библиотеки, которые в PHP обычно называют расширениями (extensions), то подключить их можно, достав из архива с расширением zip.

Далее рассматривается установка модуля версии 5.3.5. От цифр, идущих после первой пятерки, зависит конкретный порядок установки, который несколько различен, скажем, для версий PHP 5.3 и 5.2. На диске, приложенном к этой книге, находится модуль PHP именно версии 5.3.5.

Начинать установку следует с запуска инсталлятора с расширением msi. Если системный диск вашего компьютера — C:, то инсталлятор предложит установить модуль в каталог C:\Program Files\PHP\ — соглашайтесь.

Затем вас спросят, с каким веб-сервером вы собираетесь использовать PHP. Есть два способа подключения PHP — как модуля Web-сервера и как CGI-приложения. В качестве модуля PHP становится частью Web-сервера и при запуске последнего тоже запускается и постоянно готов к работе. При работе в качестве CGI-приложения выполнение PHP как отдельной программы происходит при каждом запросе Web-страницы. PHP 5 работает быстрее, если его подключить как модуль. Нужно только учесть, что при таком способе подключения все изменения, вносимые в файл php.ini, вступят в силу только после перезапуска сервера Apache.

Выберите вариант, предусматривающий работу PHP в качестве модуля сервера Apache 2.2 (рис. 2.5).

Затем инсталлятор захочет узнать, в каком каталоге находится конфигурационный файл сервера Apache (рис. 2.6). Дело в том, что модуль PHP постарается внести в этот файл информацию о своем местоположении и необходимости загружать модуль PHP при запуске сервера Apache.

В следующем диалоговом окне потребуется выбрать библиотеки, которые вы собираетесь использовать в дальнейшей работе (рис. 2.7). Как неудачно для начинающего! Ведь он еще не знает, что ему потребуется. Но делать нечего, просто выберите перечисленные далее библиотеки, они будут использованы в этой книге. Необходимый для дальнейшей работы набор расширений включает в себя:

- gd2.dll — графическая библиотека;
- mbstring.dll — расширение для работы с многобайтными кодировками;
- mysqli.dll — расширение для работы с сервером MySQL 5;
- xsl.dll — библиотека функций для XSL-преобразований.

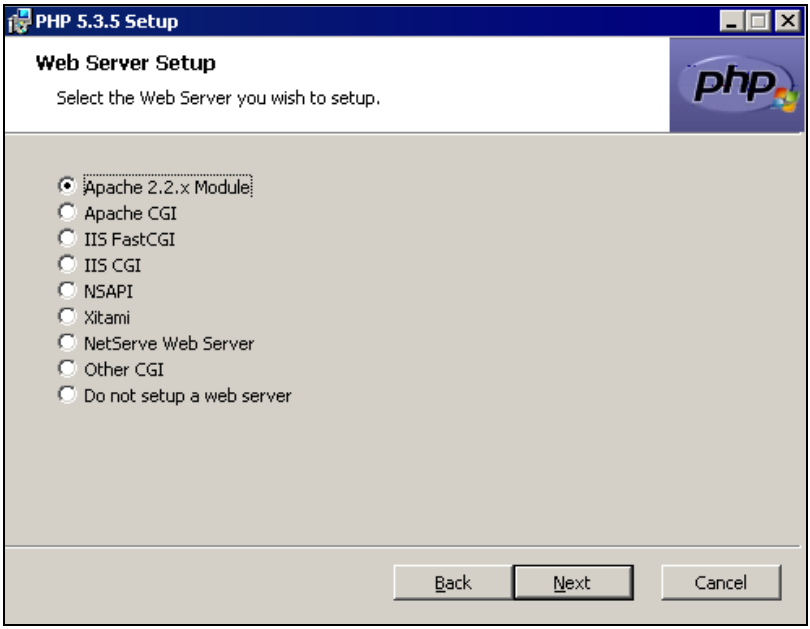


Рис. 2.5. Указание способа подключения модуля PHP к Web-серверу

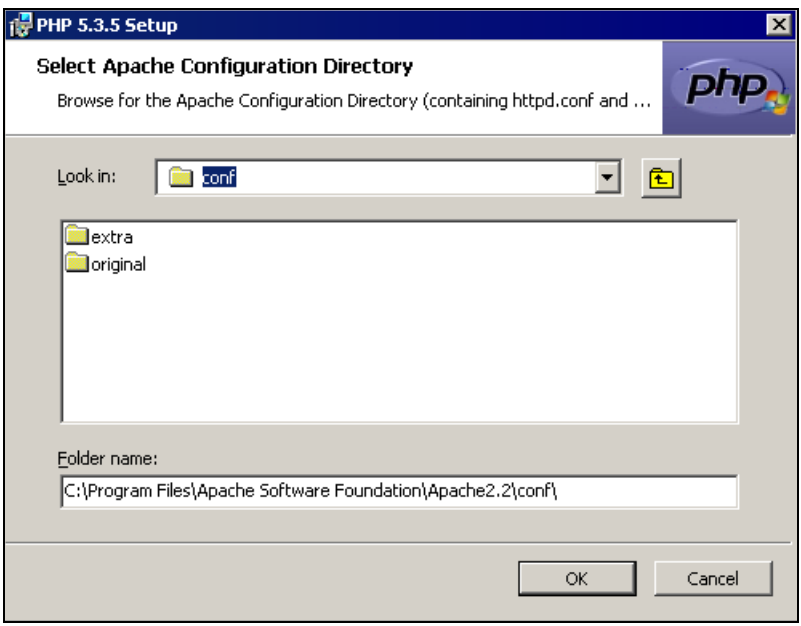


Рис. 2.6. Выбор каталога, в котором находится конфигурационный файл сервера Apache

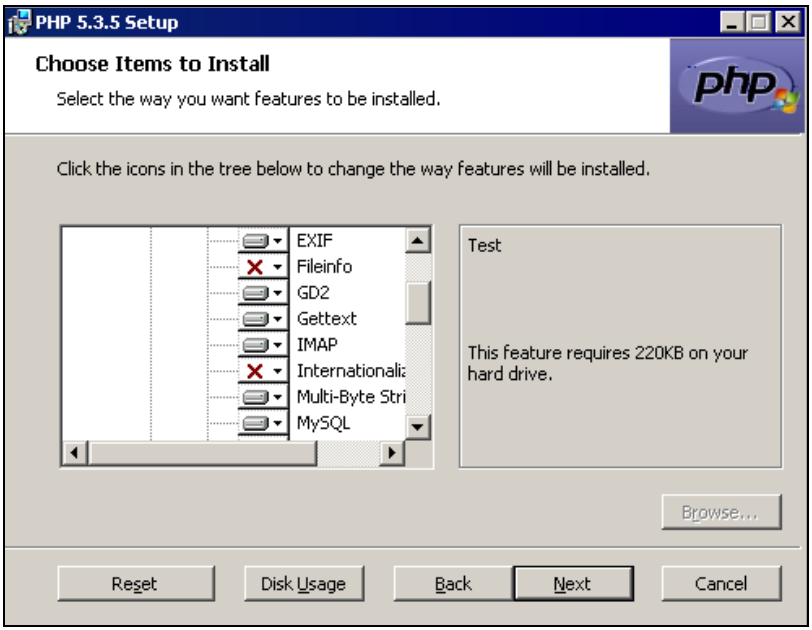


Рис. 2.7. Установка библиотек

Не устанавливайте лишних библиотек: они занимают память, а некоторые могут привести к нежелательным проблемам при работе модуля PHP (перефразируя рекламу, отметим, что не все расширения одинаково полезны).

В результате установки модуля PHP в конфигурационный файл сервера Apache будут внесены следующие строки (ищите в самом конце файла httpd.conf):

```
AddDefaultCharset utf-8
AddType application/x-httpd-php .php
#BEGIN PHP INSTALLER EDITS - REMOVE ONLY ON UNINSTALL
PHPIniDir "C:/Program Files/PHP/"
LoadModule php5_module "php5apache2_2.dll"
#END PHP INSTALLER EDITS - REMOVE ONLY ON UNINSTALL
```

Строки, начинающиеся с символа решетки `#`, являются комментариями, а вот остальные обязательно должны присутствовать в файле httpd.conf.

Директива `AddType` связывает расширение файла запрашиваемого клиентом ресурса и MIME-тип, известный серверу Apache.

Директива `PHPIniDir` указывает местонахождение конфигурационного файла php.ini. Директива `LoadModule` заставляет Apache подгружать указанный модуль. После установки модуля PHP обязательно проверьте наличие этих строк и при необходимости допишите отсутствующие директивы. Это необходимо, т. к. инсталляционные программы разных версий различным образом добавляют директивы в httpd.conf.

Ну вот, теперь модуль PHP установлен. Конфигурационный файл этого модуля — php.ini. В этом файле, который модуль PHP читает при запуске, большинство

строк начинается с символа точки с запятой или решетки — это комментарии для нас, модуль PHP их не обрабатывает. Остальные строки содержат директивы для PHP. Найдите директиву `display_errors` и поменяйте ее значение на `On`. На сервере хостера ошибки не должны демонстрироваться посетителю, но при разработке сценариев без просмотра ошибок не обойтись. Кроме того, надо настроить поясное время, отредактировав следующую директиву:

```
date.timezone = "Europe/Moscow"
```

Сохраните изменения и перезапустите Apache. Если ошибок нет, то черное DOS-окно откроется и закроется без всяких слов.

> ***Примечание***
>
> При внесении изменений в файл httpd.conf начинающие Web-мастера часто пытаются еще раз запустить, а не перезапустить Apache. Это приводит к попытке запустить еще одну службу, слушающую 80-й порт. Система с негодованием известит вас об этом. Перезапустите Apache, и все пойдет как надо.

Как показывает преподавательский опыт автора, если при установке Apache и PHP возникли ошибки, лучше удалить все установленное и повторить установку компонентов с самого начала. Многие ошибки исчезают сами собой.

Глава 3

Создание сценариев на PHP. Типы данных, переменные, операторы

Редакторы для работы с PHP

В качестве редактора для скриптов PHP можно использовать любой текстовый редактор, сохраняющий данные в текстовом формате (не так, как Microsoft Word), подойдет и обычный Блокнот (Notepad). Обычно выбор разработчика падает на один из следующих редакторов.

- Notepad++ — свободно распространяемый удобный редактор, поддерживающий все актуальные кодировки русского языка. Дистрибутив Notepad++ вы найдете на диске, приложенном к этой книге.
- PHP Expert Editor (для русскоговорящих граждан — бесплатная регистрация, **http://www.phpexperteditor.com**). Этот редактор предназначен для работы под Windows. Файл PHP размечен по строкам, и на каждой строке можно поставить так называемую точку останова. При запуске скрипта отладчик дойдет только до этой точки.
- Adobe Dreamweaver — прекрасный редактор, имеющий коммерческий статус.
- Zend Development Environment (ZDE) — идеальное решение для профессиональных PHP-разработчиков. ZDE — коммерческий продукт, эта среда разработана фирмой-производителем PHP. Он позволяет отлаживать код PHP 4 и PHP 5, не требуя установленного Web-сервера. У него нет проблем с кодировками, и вам не нужно скачивать русификаторы для того, чтобы меню были на русском. Кроме того, в поставку входит PHP Manual.

Базовый синтаксис

Документация расшифровывает аббревиатуру PHP как "рекурсивный акроним словосочетания ”PHP: Hypertext Preprocessor“".

В ходе своей работы модуль PHP читает и исполняет сценарий — текстовый файл, содержащий набор команд. Результат выполнения сценария обычно представляет собой HTML-документ, который PHP передает Apache, а тот уже клиентскому браузеру.

PHP-сценарии следует сохранять в файлах с расширением php (в соответствии с выполненными настройками Web-сервера) в каталоге htdocs сервера Apache. Запустить сценарий our_script.php, хранимый в корневом каталоге сервера, можно только из браузера, в адресной строке которого следует набрать **http://localhost/our_script.php.** Таким образом мы создаем запрос по протоколу HTTP. В этом случае Apache передаст PHP-сценарий на обработку модулю PHP.

> ***Примечание***
>
> Частая ошибка начинающих состоит в том, что они, желая запустить скрипт на выполнение, щелкают дважды по имени файла сценария в окне Проводника. Но эта программа не умеет отправлять HTTP-запросы!

Модуль PHP будет обрабатывать и выполнять только те команды скрипта, которые заключены в специальные теги — *дескрипторы PHP*.

PHP поддерживает следующие теги:

- XML-стиль:

```
<?php Здесь идет код на PHP ?>
```

- HTML-стиль:

```
<script language="php"> Здесь идет код на PHP </script>
```

- краткий стиль:

```
<? Здесь идет код на PHP ?>
```

- ASP-стиль:

```
<% Здесь идет код на PHP %>
```

Использование двух последних стилей в настоящее время обычно запрещается администраторами Web-серверов. Найдите директивы, задающие этот запрет (`short_open_tag` и `asp_tags`), в файле конфигурации модуля PHP php.ini.

Действия, которые должен выполнить PHP-модуль, указываются PHP-операторами, помещаемыми между открывающим и закрывающим дескрипторами.

Сохраните следующий пример в файл, например, 1.php в каталоге htdocs вашего Web-сервера:

```
<?php
  echo "Этот текст вы увидите на экране!";
?>
```

Запустите браузер и наберите в адресной строке: `localhost`. В *главе 2* об установке Apache мы удалили все индексные файлы, поэтому в окне браузера вы должны увидеть список файлов, хранящихся в каталоге htdocs.

Найдите в списке сценарий 1.php и запустите его. Если вы правильно установили и настроили Apache и PHP, то увидите в окне браузера сообщение "Этот текст вы увидите на экране!" Инструкция `echo` в PHP означает, что текст, следующий за ней, должен быть выведен в браузер.

Команды на языке PHP можно встраивать в HTML-документ. Модуль PHP передаст Web-серверу HTML-теги без изменений, начнет обработку только тех

команд, которые встроены в HTML-документ с помощью специальных тегов. Например, сохраните следующий пример в файл 1.php:

```
<html>
  <head>
    <title>Our first page</title>
  </head>
<body>
<?php
  echo "Этот текст вы увидите на экране!";
?>
</body>
</html>
```

Обратитесь к этому файлу с помощью браузера (не забудьте, что в адресной строке надо указать протокол, имя хоста, путь к файлу и его имя: http://localhost/1.php). Просмотрите исходный код этой страницы в браузере — вы не увидите кода PHP: скрипт был выполнен на сервере, а в браузер был передан только результирующий HTML-документ.

Типы данных

Основой любого языка программирования является хранение и обработка данных. При этом важно знать, данные каких типов могут использоваться в конструкциях языка. PHP может оперировать следующими типами данных:

- `integer` — для целых чисел;
- `double` — для действительных чисел (с плавающей точкой).

  В PHP не поддерживается европейская нотация записи действительных чисел с помощью запятой, отделяющей дробную часть числа. Число с плавающей точкой может быть представлено в экспоненциальном представлении: `1.2e2` — это число 120.

  Кроме десятичных чисел в скрипте можно использовать и шестнадцатеричные числа. Перед таким числом должны стоять символы `0x`. Допустимо также применять восьмеричные числа. Перед таким числом надо ставить `0`. Таким образом, число `012` — это восьмеричное число, в десятичным выражении равное 10;
- `boolean` — логические данные. Могут принимать значение `true` (истина) или `false` (ложь);
- `string` — для строк символов;
- `array` — для хранения нескольких элементов данных одного типа;
- `object` — для хранения экземпляров классов;
- `NULL` — специальное значение, указывающее, что данные не имеют значения, даже нуля или пустой строки, т. е. это говорит об отсутствии значения какого-либо типа;
- `resource` — специальный тип данных, содержащих ссылку на какой-то внешний по отношению к скрипту источник данных, например текстовый файл на диске или графические данные в памяти сервера.

Комментарии

Комментарии используются для пояснения назначения сценария, почему он написан именно таким образом и т. п. Считается, что комментарий должен составлять около 30% от объема кода. Следует писать комментарии сразу на этапе создания и отладки сценария — позже до этого руки у вас никогда не дойдут. Перед телом функции всегда помещайте комментарий, объясняющий ее назначение. Добавляйте комментарии в сомнительных участках кода, когда нет уверенности, что он будет работать как надо. Если назначение кода неочевидно, внесите информацию о предназначении этого участка.

PHP-модуль не анализирует комментарии, которые для него равнозначны пробелам. Многострочные комментарии должны начинаться с символов `/*` и завершаться символами `*/`:

```
/*
  Данный сценарий позволяет
  распечатать все предложения фирмы "IT консалтинг".
*/
```

Часто используются и однострочные комментарии:

```
echo "Привет";  // Вывод сообщения на экран
```

Все символы, следующие за символом комментария (`//`) вплоть до конца строки или до завершающего дескриптора PHP, в зависимости от того, что встретится раньше, рассматриваются как комментарий.

Выражения и операторы

Для того чтобы работать с данными, необходимо идентифицировать область памяти, в которой они хранятся. *Идентификаторы* используются для именования данных или блоков кода (имена функций).

Идентификаторы могут начинаться с буквы или символа подчеркивания. Последующие символы в идентификаторе могут быть буквами, цифрами или знаками подчеркивания.

Почти все в PHP является *выражением*. Простое и точное определение выражения — "все что угодно, имеющее значение". Основными формами выражений являются константы и переменные. Если вы записываете `$a = 5`, то присваиваете значение `5` переменной `$a`.

Оператор состоит из одного или более выражений. Операторы бывают трех видов:

- во-первых, это унарные операторы, которые работают только с одним аргументом, например, `!` (оператор отрицания) или `++` (инкремент);
- вторую группу составляют бинарные операторы, работающие с двумя аргументами, например, `+` (сложение) или `/` (деление);
- третью группу составляет тернарный оператор `?:`. При работе этого оператора проверяется выполнение условия, и в зависимости от результата возвращается одно из двух значений. Например:

```
$a = ($b > 5)? 7 : 0;
```

Если `$b` больше 5, то переменная `$a` получит значение 7, в противном случае — 0.

Операторы PHP разделяются точками с запятой (`;`). Отсутствие точки с запятой в конце команды заставляет PHP выводить на экран сообщение об ошибке.

Константы

Для задания значений, которые не будут изменяться в ходе выполнения программы, можно применять *константы*, которые определяются с помощью конструкции `define`, например:

```
<?php
  define('OUR_CONST', 55);
  echo OUR_CONST;
?>
```

В PHP имена констант традиционно записываются в верхнем регистре. Если вы хотите, чтобы другие программисты могли легко воспринимать ваш код, придерживайтесь этого правила.

Переменные

Переменная — это поименованная ячейка памяти, в которой хранится значение. Идентификаторы переменных в PHP начинаются со знака доллара (`$`). Имена переменных чувствительны к регистру, т. е. `$var` и `$Var` — это разные переменные. При назначении переменных и функций старайтесь давать им осмысленные имена. Иногда ничего не приходит в голову, и появляется желание назвать переменную как попало — остерегайтесь этого. В свое время было потрачено немало часов из-за неудачно названных переменных, иногда отладить код удавалось лишь потому, что переменные были переименованы подобающим образом.

Оператор присваивания используется для задания переменной некоторого значения. Например:

```
$a = 25;
```

В PHP тип переменной определяется присвоенным ей значением. В следующем примере переменная `$b` имеет тип `double`:

```
$b = 1.23;
```

При этом в программе дальше может следовать такая строка:

```
$b = "Привет";
```

В этом случае переменная `$b` поменяет свой тип на тип `string`. PHP в любой момент времени изменяет тип переменной в соответствии с данными, хранящимися в ней.

Строковую переменную можно определить тремя способами.

Во-первых, с помощью одинарных кавычек:

```
$str ='Голубое небо';
```

Если в строковое значение необходимо добавить одинарную кавычку, то надо предварить ее обратным слэшем:

```
$str = 'Д\'Артаньян';
```

Второй способ — определить строковое значение в двойных кавычках.

```
$str = "синее море, белый пароход";
```

При этом можно вставить некоторые специальные символы, используя escape-последовательности:

- `\n` — конец строки;
- `\r` — возврат каретки;
- `\t` — горизонтальная табуляция;
- `\\` — обратный слэш.

Слово escape можно перевести как "управлять", т. е. escape-последовательности управляют способом отображения данных. Например:

```
$str="синее море, \n белый пароход";
```

Написав затем в скрипте конструкцию `echo $str;`, вы не увидите в браузере перехода на новую строку, но откройте исходный код — все на месте, просто браузер удалил символ новой строки.

Третий способ задания строкового значения — с помощью heredoc-синтаксиса:

```
$str= <<<MARKER
Это сам текст, который может занимать
несколько строк и должен заканчиваться таким же
маркером, каким начинался, причем маркер должен быть
единственным текстом в строке кроме точки с запятой.
MARKER;
```

Приведем еще один пример встраивания PHP-кода в HTML-документ. Пусть переменная `$jpg` получает строковое значение (им будет имя файла):

```
<?php
  $jpg="1.jpg";
  echo "<img border=1 src=\"$jpg\" width=200 height=120>";
?>
```

Сохраните любой рисунок в формате JPEG под именем 1.jpg в тот же каталог, в который сохранили только что приведенный пример, запустите пример и посмотрите исходный код.

Можно включать имя переменной в строку, при этом происходит подстановка значения переменной, если использовать двойные кавычки или heredoc-синтаксис. Следующий пример выведет в браузере строку `"5 котят"`:

```
<?php
  $num = 5;
  echo "$num котят";
?>
```

Поменяйте в приведенном примере двойные кавычки на одинарные, и вы увидите, что PHP не подставляет значение переменной в строку, ограниченную одинарными кавычками. Поэтому для ускорения выполнения сценария рекомендуется использовать при задании строк именно одинарные кавычки — PHP не разбирает текст, заключенный в них.

Для строк определена операция *конкатенации* — объединение строк:

```
<?php
  $start = "Первая часть строки ";
  $stop = " Вторая часть строки";
  echo $start.$stop;
?>
```

Сравните два оператора:

```
<?php
  echo "Другое начало строки ".$stop;
  echo "Другое начало строки $stop";
?>
```

Конструкции `echo` выведут в браузер одну и ту же строку, но первая конструкция работает на 30% быстрее.

Ссылки

Внутри механизма PHP (Zend engine) существует разделение между именем переменной и ее значением. Создадим три переменные:

```
$a = 2; $b = 4; $c = 5;
```

Выведем их значения:

```
print $a;   print $b;   print $c;
```

Внутреннее представление переменных в PHP будет таким, как представлено на рис. 3.1.

В PHP механизм ссылок позволяет создать несколько имен для одной и той же переменной. Важно понять это, чтобы правильно использовать ссылки в своем коде. Создадим теперь переменную так:

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

Символ `&` перед именем переменной указывает на использование ссылки. Переменная `$b` определена как ссылка на переменную `$a`, т. е. она не имеет своей области данных, а пользуется той, которая есть у переменной `$a`. Механизм здесь тот же, что и в жестких связях в UNIX (hard link). При изменении значения любой из переменных, `$b` или `$a`, будет меняться и значение второй переменной. Посмотрим, как в этом случае будут выглядеть внутренние связи между именами переменных и их значениями (рис. 3.2).

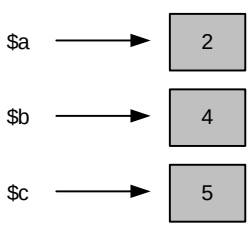


Рис. 3.1. Создание переменных

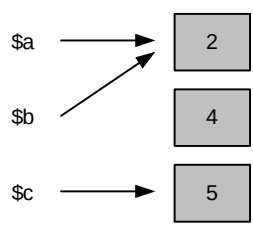


Рис. 3.2. Создание ссылки

Старое значение переменной `$b` теперь не имеет ни одного имени, связанного с ней, и поэтому больше недоступно из PHP-кода. Ядро PHP автоматически уничтожает такие переменные, поэтому значение 4, ранее бывшее значением переменной `$b`, перестает существовать с этого момента.

Поскольку теперь первое значение в таблице значений имеет два имени (`$a` и `$b`), то обращение к любому из этих имен будет рассматриваться как обращение к этому значению:

```
print $a;       // Результат: 2
print $b;       // Результат: 2
print $c;       // Результат: 5
```

При задании переменной путем приравнивания ее другой переменной (`$b = $a`) происходит копирование значения переменной `$a` в область памяти, которая обозначается как `$b`. При этом при изменении значения переменной `$a` значение переменной `$b` не меняется.

Глава 4

Операции и управляющие конструкции

Программа обычно состоит из последовательности инструкций (statements). Инструкции могут представлять собой объявление переменных, вызов функции и включать в себя операции. *Операции* являются базовыми элементами языка программирования. Наиболее часто применяются арифметические, логические и строковые операции.

Арифметические операции

В PHP поддерживаются обычные *арифметические операции*, работающие с двумя операндами:

- сложение (`+`);
- вычитание (`-`);
- умножение (`*`);
- деление (`/`);
- остаток от деления (`%`).

Можно сохранять результат операции в переменную:

```
$a = 5 + 7;
```

Кроме обычных операций можно применять комбинированные. Например, если надо взять значение переменной `$a`, прибавить к нему число 5 и сохранить как новое значение переменной `$a`, то это можно сделать двумя способами:

```
$a = 10;
$a = $a + 5;
```

или:

```
$a = 10;
$a += 5;
```

Разрешены комбинированные операции, представленные в табл. 4.1.

Примечание

Остаток `$a % $b` будет отрицательным для отрицательных значений `$a`.

Кроме того, PHP заимствовал из языка C операции префиксного и постфиксного инкремента (`++`) и декремента (`--`).

Таблица 4.1. *Комбинированные операции присваивания PHP*

Символ операции	Пример использования	Эквивалентная операция
+=	`$a += $b`	`$a = $a + $b`
-+	`$a -= $b`	`$a = $a - $b`
*=	`$a *= $b`	`$a = $a * $b`
/=	`$a /= $b`	`$a = $a / $b`
%= (остаток от деления)	`$a %= $b`	`$a = $a % $b`
.= (конкатенация)	`$a .= $b`	`$a = $a . $b`

Префиксный инкремент увеличивает значение переменной, перед которой записывается, на 1, а затем возвращает ее новое значение. Например, `++$a`.

Постфиксный инкремент возвращает значение переменной, после которой записывается, а потом увеличивает ее значение на 1. Например, `$a++`.

Префиксный декремент уменьшает значение переменной, перед которой записывается, на 1, а затем возвращает ее новое значение. Например, `--$a`.

Постфиксный декремент возвращает значение переменной, после которой записывается, а потом уменьшает ее значение на 1. Например, `$a--`.

Поэкспериментируйте с этими операциями, чтобы хорошенько с ними разобраться. Например:

```
<?php
  $a = 10;
  $b = $a++;
  echo $a;
?>
```

Вот пример, дающий неочевидный результат:

```
<?php
  $a = 1;
  $a = $a-- + 1;
  echo $a;
?>
```

Результат приведенных здесь операций будет 2. Дело в том, что операция `$a--` выполняется после операции `$a + 1`, но перед операцией присваивания. Следовательно, к тому моменту, когда `$a + 1` присваивается переменной `$a`, декремент будет просто потерян.

Поразрядные операции

Поразрядные операции, перечисленные в табл. 4.2, позволяют обрабатывать числа, представляя их в двоичном виде.

Таблица 4.2. *Поразрядные операции PHP*

Пример	Название	Результат
`$a & $b`	Побитовое И	Устанавливаются только те биты, которые установлены и в `$a`, и в `$b`
`$a | $b`	Побитовое ИЛИ	Устанавливаются те биты, которые установлены либо в `$a`, либо в `$b`
`$a ^ $b`	Исключающее ИЛИ	Устанавливаются только те биты, которые установлены либо только в `$a`, либо только в `$b`
`~ $a`	Отрицание	Устанавливаются те биты, которые в `$a` не установлены, и наоборот
`$a << $b`	Сдвиг влево	Все биты переменной `$a` сдвигаются на `$b` позиций влево (каждая позиция подразумевает умножение на 2)
`$a >> $b`	Сдвиг вправо	Все биты переменной `$a` сдвигаются на `$b` позиций вправо (каждая позиция подразумевает деление на 2)

Оператор подавления ошибки

Обычно PHP сообщает об ошибках в коде, давая подробное и полезное сообщение для объяснения происходящего. Иногда следует подавить вывод сообщения об ошибке. *Оператор подавления ошибки* `@` ставится перед операцией, в ходе которой может возникнуть ошибка.

Например, попытка деления на ноль приведет к ошибке, но оператор подавления ошибки отменит вывод сообщения на экран:

```
<?php
  @$a = 1/0;
?>
```

Естественно, что применять этот оператор просто для подавления непонятной ошибки — плохая идея. Его использование оправдано, только если точно известна причина возникновения ошибки. Этот оператор потребуется нам, когда придется работать с переменными, не будучи уверенными в том, что они определены. Тогда сообщение об ошибке будет излишним, и мы его подавим.

Операции сравнения

Операции сравнения (табл. 4.3) используются для определения отношений между двумя операндами. Результат сравнения — это логическая величина (`true` или `false`).

Для выполнения операций сравнения PHP преобразует сравниваемые переменные к одному типу. Иногда такие преобразования могут приводить к нежелательному результату. Если необходимо сравнивать не только значения, но и тип переменной, следует использовать оператор `===`, который проверяет равенство значений и совпадение типов переменных, а оператор `!==` определяет различие как значений переменных, так и их типов.

Таблица 4.3. *Операторы сравнения*

Пример	Название	Результат
`$a == $b`	Равно	`true`, если `$a` равно `$b`
`$a === $b`	Тождественно равно	`true`, если `$a` равно `$b` и имеет тот же тип
`$a != $b`	Не равно	`true`, если `$a` не равно `$b`
`$a <> $b`	Не равно	`true`, если `$a` не равно `$b`
`$a !== $b`	Тождественно не равно	`true`, если `$a` не равно `$b` или в случае, если они разных типов
`$a < $b`	Меньше	`true`, если `$a` строго меньше `$b`
`$a > $b`	Больше	`true`, если `$a` строго больше `$b`
`$a <= $b`	Меньше или равно	`true`, если `$a` меньше или равно `$b`
`$a >= $b`	Больше или равно	`true`, если `$a` больше или равно `$b`

Следующий пример демонстрирует, что при сравнении целочисленной и строковой переменных с помощью оператора `==` различие типов будет проигнорировано (результат проверки будет `true`):

```
<?php
  $a = "1";
  $b = 1;
  if ($a == $b) echo "Равны";
?>
```

При такой проверке пустая строка окажется равной нулю (`$a = ""; $b = 0;`), но добавьте внутрь скобок с проверкой условия еще один знак равенства (`===`), и тогда PHP откажется считать значения этих переменных равными.

Логические операции PHP

Операторы этой группы представлены в табл. 4.4, они работают с логическими переменными и используются в управляющих конструкциях.

Таблица 4.4. *Логические операторы*

Пример	Название	Результат
`$a and $b`	Логическое И	`true`, если и `$a`, и `$b` `true`
`$a or $b`	Логическое ИЛИ	`true`, если или `$a`, или `$b` `true`
`$a xor $b`	Исключающее ИЛИ	`true`, если `$a` или `$b` `true`, но не оба
`! $a`	Отрицание	`true`, если `$a` `false`
`$a && $b`	Логическое И	`true`, если и `$a`, и `$b` `true`
`$a || $b`	Логическое ИЛИ	`true`, если или `$a`, или `$b` `true`

Смысл двух разных вариантов для операторов `and` и `or` в том, что они работают с различными приоритетами.

Преобразование типов

PHP автоматически преобразует типы данных при операциях, но можно сделать это и явно с помощью оператора преобразования типов:

- `(int)` — преобразование величины в ее целочисленный эквивалент;
- `(float)` — преобразование величины в действительное число;
- `(string)` — преобразование величины в строку;
- `(array)` — приведение к массиву;
- `(object)` — приведение к объекту.

Следующий код выведет на экран "1": так преобразуется к целому значению логическое `true`.

```
<?php
  $b = 5;
  $a = ((++$b) > 5);
  echo (int)$a;
?>
```

Тернарная операция

Эта операция, `?:`, работает точно так же, как в C. Она записывается в форме:

условие `?` *значение, если условие истинно* `:` *значение, если условие ложно*

Тернарная операция в следующем примере присвоит переменной `$result` значение `"Выдержал"`, если значение условия `$grade > 50` окажется `true`, в случае же его ложности переменная `$result` получит значение `"Не выдержал"`:

```
$result = ($grade > 50 ? "Выдержал" : "Не выдержал");
```

Управляющие конструкции

Управляющие конструкции существуют именно для того, о чем говорит их название, — для управления ходом выполнения программы. Управляющие конструкции позволяют указывать условия, при которых должен выполняться тот или иной фрагмент кода. *Условие* — это выражение, возвращающее булево значение.

Условные операторы

Конструкции, которые указывают программе о необходимости принятия решений, называются *условными операторами*.

Условный оператор *if...else*

Для принятия решений применяется оператор `if`. Чтобы его использовать, ему необходимо передать условие. Конструкция `if` имеет следующий синтаксис:

```
if (условие) действие;
```

Если условие в скобках оказывается истинным (имеет значение `true`), то выполняется действие. Например:

```
<?php
  if ($name == 'Anna') echo 'Привет, Анна!';
?>
```

Если проверка условия в скобках даст значение `true`, то в браузер будет выведен текст "Привет, Анна!" Действие может состоять из нескольких строк кода, тогда эти строки следует поместить в фигурные скобки:

```
<?php
  if ($name == 'Anna')
  {
    echo 'Привет, Анна!';
    $age = 25;
  }
?>
```

Часто требуется принимать решение не только о выполнении того или иного действия, но и выбирать определенный набор действий в противном случае. Оператор `else` позволяет определить альтернативное действие, которое должно выполняться, если значением условия в операторе `if` окажется `false` (листинг 4.1).

Листинг 4.1. Конструкция `if...else`

```
<?php
if ($name == 'Anna')
{
  echo 'Привет, Анна!';
  $age = 25;
}
else
{
  echo 'Привет всем!';
  $name = 'unknown';
}
?>
```

Примечание

Частая ошибка начинающих состоит в том, что они ставят точку с запятой после круглой скобки с условием, а затем с удивлением видят, что код работает не так, как ожидалось. Но операция-то заканчивается только после строк, содержащих действия.

Для многих принимаемых решений может существовать более двух возможностей. Последовательность множества вариантов действий создается с использованием оператора `elseif`, который представляет собой комбинацию операторов `else` и `if`. При наличии последовательности условий программа может проверять каж-

дое из них до тех пор, пока не отыщет то, значением которого является `true`. Рассчитаем скидку на покупку в зависимости от количества заказанных товаров, которое является значением переменной `$number`. Скидка не предоставляется, если заказано меньше 10 штук. Скидка составляет 5% в случае заказа от 10 до 50 штук. При заказе более 50 товаров скидка составляет 10% (листинг 4.2).

Листинг 4.2. Конструкция `elseif`

```
<?php
if ($number < 10)
  $discount = 0;
elseif ($number >= 10 && $number <= 49)
  $discount = 5;
elseif ($number >= 50)
  $discount = 10;
?>
```

Выход из управляющей конструкции или сценария

Если при некоторых условиях требуется завершить выполнение всего PHP-сценария, необходимо применить оператор `exit`. Обычно этот оператор используется при проверке на ошибки. Например, ранее приведенный пример можно было бы изменить следующим образом:

```
if ($age == 0)
{
  echo 'Пожалуйста, проверьте вводимые данные.';
  exit;
}
```

Вызов оператора `exit` прекращает выполнение оставшейся части PHP-сценария.

Посмотрим, как условные конструкции позволяют проверить, все ли данные получены скриптом. Значения переменных `$name`, `$family_name` и `$age` должны быть получены от пользователя через форму. Поскольку мы пока не умеем этого делать, то будем считать, что в зависимости от действий пользователя переменные либо определены и имеют значения, либо не определены, т. е. их значения — `NULL`. В приведенном далее примере (листинг 4.3) все переменные определены, и в браузере появится надпись "Спасибо, данные записаны".

Листинг 4.3. Проверка значений переменных

```
<?php
$name = 'Anna';  $family_name = 'Nikolaeva';  $age = 25;
if (!$name || !$family_name || !$age)
{
  echo 'Пожалуйста, укажите все данные.';
```

```
  exit;
}
echo 'Спасибо, данные записаны.';
?>
```

Если хотя бы одна переменная не была определена, надо попросить пользователя вернуться и внести данные. В этом случае продолжать выполнение сценария не следует. Удалите в приведенном примере присвоение значения какой-нибудь переменной, например `$age`, и вы увидите в браузере:

```
Notice: Undefined variable: age in
C:\Program Files\Apache Group\Apache2\htdocs\my_book\if.php on line 3 Пожалуйста,
укажите все данные.
```

Использование оператора `exit` привело к тому, что сценарий закончил работу, выведя предупредительное сообщение, не дойдя до исполнения инструкции `echo 'Спасибо, данные записаны'`.

Но кроме предусмотренного сообщения пользователь получил и непонятное сообщение модуля PHP об ошибке. Хуже того, он увидел, что в сценарии задействована переменная `$age`, а именно подобная информация и нужна злоумышленнику, пытающемуся нарушить работу серверного скрипта. Следовательно, передача данных должна быть организована так, что либо значение переменной в худшем случае будет пустой строкой, но все-таки переменная будет определена, либо каждая переменная должна отдельно проверяться с помощью функции `isset()`. Например, как в листинге 4.4.

Листинг 4.4. Использование функции `isset()` для проверки значений переменных

```
if (!isset($name) && !isset($family_name)&& !isset($age))
{
  echo 'Пожалуйста, укажите все данные.';
  exit;
}
else
{
  echo 'Спасибо, данные записаны.';
}
```

Последний пример позволяет проверить, все ли переменные определены, и вывести соответствующие сообщения.

Условный оператор *switch*

Оператор `switch` работает аналогично оператору `if`, но позволяет условному выражению иметь в качестве результата более двух значений. В операторе `if` условие принимает значение `true` или `false`. В операторе `switch` условие может принимать любое количество различных значений простого типа (`integer`, `string`

или double). Для обработки каждого из значений, на которые требуется реагировать, необходимо записать оператор case, а также (необязательно) можно определить случай для обработки по умолчанию любых значений, для которых специально не был задан оператор case (листинг 4.5).

Листинг 4.5. Оператор switch

```
<?php
switch($go)
{
  case "Налево пойдешь" :
    echo "Получишь полцарства";
    break;
  case "Направо пойдешь" :
    echo "Коня получишь";
    break;
  case "Прямо пойдешь" :
    echo "Принцессу спасешь";
    break;
  default:
    echo "Лентяям ничего не светит";
    break;
}
?>
```

Переменная $go может получить одно из предусмотренных значений, а иногда и какое-либо непредусмотренное. После каждого echo следует добавлять оператор break, предназначенный для того, чтобы завершить работу конструкции.

Циклы

Циклы применяются в том случае, когда необходимо выполнить повторяющиеся действия.

Цикл *while*

Цикл while выполняет блок операторов многократно до тех пор, пока условие принимает значение true. В общем случае цикл while используется, когда неизвестно, сколько итераций потребуется для выполнения условия. Пример представлен в листинге 4.6.

Листинг 4.6. Цикл while

```
<?php
$status = 'Солнце высоко';
while ($status == 'Солнце высоко')
```

```
{
  echo 'Светло и тепло';
  $status = 'Солнце садится';
}
?>
```

В приведенном примере цикл будет выполнен один раз. Условие проверяется в начале каждой итерации. Если оно принимает значение `false`, блок не будет выполняться и цикл завершается. После этого выполняется оператор, следующий за циклом.

> ***Примечание***
>
> Можно изменить цикл так, чтобы он выполнялся бесконечно. Для этого в примере достаточно удалить последний оператор. Когда вам надоест наблюдать "мучения" браузера в бесконечном цикле, просто закройте его окно.

Можно предусмотреть выход из бесконечного цикла с помощью оператора `break`. При его использовании выполнение сценария будет продолжаться, начиная с оператора, следующего за циклом (листинг 4.7).

Листинг 4.7. Применение оператора `break`

```
<?php
$i = 0;  $j = "солнце светит";
while ($j != "солнце село")
{
  echo "Муравей прошел $i шагов, ";
  if ($i >= 10) break;
  $i++;
  echo $j . "<br>";
}
?>
```

Иногда неудобно, что код внутри цикла `while()` не выполнится ни одного раза, если условие изначально `false`. Если желательно, чтобы код исполнялся один раз, а потом проверялось выполнение условия, то можно использовать цикл `do...while` (листинг 4.8).

Листинг 4.8. Цикл `do...while`

```
<?php
$i = 0;
do
```

```
{
   echo $i;
} while ($i > 0);
?>
```

Цикл *for*

Основная структура цикла `for` имеет вид:

```
for (выражение1; условие; выражение2) выражение3
```

Здесь:

- *выражение1* выполняется один раз в начале цикла. Обычно в нем устанавливается начальное значение счетчика;
- *условие* проверяется перед каждой итерацией. Если это выражение возвращает значение `false`, итерация прекращается. Обычно в нем значение счетчика сравнивается с предельным значением;
- *выражение2* выполняется в конце каждой итерации. Обычно в нем изменяется значение счетчика;
- *выражение3* выполняется один раз для каждой итерации. Обычно это выражение представляет собой блок кода и содержит собственно тело цикла.

Пример представлен в листинге 4.9.

Листинг 4.9. Цикл `for`

```
<?php
for ($i = 10; $i > 4; $i--)
{
  echo "$i матрешек вложены одна в другую. Откроем матрешку. <br>";
}
?>
```

Можно использовать цикл и с несколькими операциями инициализации цикла (листинг 4.10).

Листинг 4.10. Цикл `for` с несколькими операциями инициализации

```
<?php
for ($i = 0, $j = "солнце светит"; $j != "солнце село"; $i++)
{
  echo "Муравей прошел $i шагов, ";
  if ($i >= 10) $j="солнце село";
  echo $j . "<br>";
}
?>
```

Оператор `continue` используется в циклах для того, чтобы пропустить остаток действий в текущей итерации и продолжить выполнение цикла со следующей итерации. В представленном в листинге 4.11 примере в цикле распечатаются только числа 0, 1, 3 и 4.

Листинг 4.11. Использование оператора `continue` в цикле `for`

```
<?php
for ($i = 0; $i < 5; ++$i)
{
  if ($i == 2)
    continue;
  echo "$i\n";
}
?>
```

Глава 5

Функции и повторное использование кода

Функция — это часть программы, обозначенная определенным именем, выполняющая определенную задачу и необязательно возвращающая результат.

Встроенные функции

В ядро PHP встроено большое количество функций. Кроме того, в PHP есть функции, которые требуют, чтобы PHP был собран со специфическими расширениями, в противном случае вы получите сообщение об ошибке, вызванной использованием неизвестной модулю PHP функции. Например, чтобы воспользоваться функциями доступа к серверу MySQL, в PHP 5 необходимо подключить специальную библиотеку для работы с MySQL. Тем не менее есть много встроенных функций, которые доступны всегда: например, функции для работы с переменными.

Вызвав функцию `phpinfo()`, можно получить информацию о подключенных модулях, параметрах настройки Apache, установленных переменных окружения и др.

Вызов функции происходит по ее имени. Если функции для работы необходима некоторая информация, то она передается в виде аргумента (параметра). Описание функций, встроенных в PHP, количество принимаемых ими параметров, и что представляет собой каждый из них, можно узнать на сайте **http://www.php.net/manual/ru/index.php**.

Описание функции включает в себя *прототип*, содержащий информацию о возвращаемом ею значении и аргументах, принимаемых этой функцией. Например, функция:

```
float round(float val[, int precision])
```

возвращает округленное значение *val* с указанной точностью *precision* (количество цифр после запятой). Точность может быть отрицательной или нулем (по умолчанию).

Перед именем функции в описании указывается тип возвращаемого значения, а перед именем аргумента — тип передаваемого в функцию значения. Если функция принимает необязательные параметры, то они указываются в квадратных скобках.

Функции для работы с переменными

Библиотека функций для работы с переменными PHP позволяет различными способами манипулировать переменными и проверять их.

Проверка и установка типов переменных

Большинство этих функций связано с проверкой типов. Двумя наиболее общими функциями являются `gettype()` и `settype()`.

При вызове функции `gettype()` ей необходимо передать переменную. Функция определяет тип переменной и возвращает строку, содержащую имя типа, или `"unknown type"`, если тип переменной не является одним из стандартных типов: `integer`, `double`, `string`, `array` или `object`.

При вызове функции `settype()` ей необходимо передать переменную, тип которой требуется изменить, и строку, содержащую новый тип переменной из приведенного ранее списка.

Вот пример применения этих функций:

```
<?php
$a = 56;
echo gettype($a)."<br>";
settype($a, "double");
echo gettype($a)."<br>";
?>
```

Перед первым вызовом функции `gettype()` переменная `$a` имеет тип `integer`. После вызова функции `settype()` ее тип изменится на `double`.

PHP предоставляет также ряд функций проверки типов. Каждая из этих функций принимает переменную в качестве аргумента и возвращает значение `true` или `false`. Вот их перечень:

- `is_array()`;
- `is_double()`, `is_float()`, `is_real()` — это одна и та же функция;
- `is_long()`, `is_int()`, `is_integer()` — это одна и та же функция;
- `is_string()`;
- `is_object()`.

Проверка состояния переменных

PHP имеет несколько функций, предназначенных для проверки состояния переменных. Первая из них — `isset()`. Эта функция принимает в качестве аргумента имя переменной и возвращает значение `true`, если переменная существует, и `false` в противном случае.

Переменную можно удалить, используя функцию `unset()`. Эта функция удаляет переменную и возвращает значение `true`.

Пример:

```
<?php
$a = 5;
if (isset($a))
{
  unset($a);
}
echo $a;
?>
```

Если ваш модуль PHP настроен так, что на экран выводятся все ошибки (в соответствии с настройками, описанными в *главе 2*), то в окне браузера вы увидите:

```
Notice: Undefined variable: a in C:\Program Files\Apache
Group\Apache2\htdocs\1.php on line 7
```

Сначала успешно прошла проверка существования переменной, потом ее удалили, после чего удивляться сообщению о том, что переменная не определена, не приходится.

Функция `empty()` проверяет существование переменной и наличие у нее непустого, ненулевого значения и возвращает, соответственно, значение `true` или `false`. Следовательно, проверка переменной с помощью этой функции приведет в следующем примере к выводу на экран сообщения "Переменная пуста":

```
<?php
$a = "";
if (empty($a))
{
  echo "Переменная пуста";
}
else
{
  echo "a: ".$a;
}
?>
```

Повторная интерпретация переменных

Можно достичь эффекта, эквивалентного преобразованию типа переменной, при помощи встроенных функций. Для этого применяются три функции:

- `intval();`
- `doubleval();`
- `strval().`

Каждая из них принимает в качестве аргумента переменную и возвращает значение переменной, преобразованное к соответствующему типу.

Следующий пример показывает, как нужно использовать описанные функции для преобразования типа переменной:

```
<?php
$a = 5;
if (is_int($a))
{
  $a = strval($a);
  echo gettype($a);
}
?>
```

Встроенные функции для работы с датой и временем

Функция *date()*

Функция `date()` возвращает отформатированную строку, содержащую текущую дату:

```
echo date("F j, Y, g:i a");
```

Описание функции `date()` выглядит так:

```
string date(string format[, int timestamp])
```

Обязательный параметр *format* задает формат вывода даты и может содержать как простой текст, так и ряд специальных символов, описание которых дано в табл. 5.1. Необязательный параметр *timestamp* задает момент времени, который будет использоваться при форматировании. Если аргумент *timestamp* не задан, то функция возвращает текущее системное время.

В UNIX-системах время записывается в виде количества секунд (*timestamp* — метка времени), прошедших с полуночи 1 января 1970 г. по Гринвичу — эпохи UNIX (The Unix Epoch). PHP использует такое представление, и вы можете узнать текущее время с помощью функции `time()`, которая возвращает текущую метку времени:

```
echo time();
```

Таблица 5.1. *Символы, распознаваемые в параметре* `format`

Символ в строке *format*	Описание	Пример возвращаемого значения
a	Ante meridiem или Post meridiem в нижнем регистре	am или pm
A	Ante meridiem или Post meridiem в верхнем регистре	AM или PM
B	Время в стандарте Swatch Internet	От 000 до 999
c	Дата в формате ISO 8601	2004-02-12T15:19:21+00:00
d	День месяца, 2 цифры с ведущими нулями	От 01 до 31
D	Сокращенное наименование дня недели, 3 символа	От Mon до Sun
F	Полное наименование месяца	От January до December
g	Часы в 12-часовом формате без ведущих нулей	От 1 до 12
G	Часы в 24-часовом формате без ведущих нулей	От 0 до 23
h	Часы в 12-часовом формате с ведущими нулями	От 01 до 12

Таблица 5.1 *(окончание)*

Символ в строке *format*	**Описание**	**Пример возвращаемого значения**
H	Часы в 24-часовом формате с ведущими нулями	От 00 до 23
i	Минуты с ведущими нулями	От 00 до 59
I (прописная)	Признак летнего времени	1, если дата соответствует летнему времени, иначе 0
j	День месяца без ведущих нулей	От 1 до 31
l (строчная)	Полное наименование дня недели	От Sunday до Saturday
L (прописная)	Признак високосного года	1, если год високосный, иначе 0
m	Порядковый номер месяца с ведущими нулями	От 01 до 12
M	Сокращенное наименование месяца, 3 символа	От Jan до Dec
n	Порядковый номер месяца без ведущих нулей	От 1 до 12
O	Разница со временем по Гринвичу в часах	Например: +0200
r	Дата в формате RFC 2822	Например: Thu, 21 Dec 2000 16:01:07 +0200
s	Секунды с ведущими нулями	От 00 до 59
S	Английский суффикс порядкового числительного дня месяца, 2 символа	st, nd, rd или th. Применяется совместно с j
t	Количество дней в месяце	От 28 до 31
T	Временная зона на сервере	Примеры: EST, MDT...
U	Количество секунд, прошедших с начала Эпохи UNIX	
w	Порядковый номер дня недели	От 0 (воскресенье) до 6 (суббота)
W	Порядковый номер недели года, первый день недели — понедельник	Например: 42 (42-я неделя года)
Y	Порядковый номер года, 4 цифры	Примеры: 1999, 2006
y	Номер года, 2 цифры	Примеры: 99, 06
z	Порядковый номер дня в году (нумерация с 0)	От 0 до 365
Z	Смещение временной зоны в секундах. Для временных зон западнее UTC это отрицательное число, восточнее UTC — положительное	От -43200 до 43200

Любые другие символы, встреченные в строке *format*, будут выведены в возвращаемой строке без изменений.

> ***Примечание***
>
> Для большинства систем допустимыми являются даты с 13 декабря 1901 г., 20:45:54 GMT по 19 января 2038 г., 03:14:07 GMT. Для Windows допустимы даты с 1 января 1970 г. по 19 января 2038 г.

Исполнение следующего кода:

```
echo date("Y-m-d");
```

приведет сегодня к такому выводу на экран: `2008-02-21`. Русское слово, добавленное в строку формата:

```
echo date("Сегодня Y-m-d");
```

благополучно появится на экране: `Сегодня 2008-02-21`.

Неудобство работы с этой функцией состоит в том, что писать дату по-русски она не умеет. Для этого надо применять другую функцию, но вначале необходимо настроить модуль PHP так, чтобы он использовал национальные особенности вывода данных.

Совокупность таких особенностей называется параметрами локализации. Локализации подлежат форматы распечатки даты и времени, денежных единиц, стандартных сообщений диалоговых окон и многое другое, что вытекает из культурно-исторических традиций различных языков. Для задания локализации применяется функция `setlocale()`, а уже после нее можно вызывать функции, распечатывающие данные с учетом локализации так, как это показано в следующем примере:

```
setlocale(LC_TIME, "rus_RUS");
echo strftime("%A %B");
```

Обратите внимание, что у функции `setlocale()` в качестве первого параметра указан аспект локализации (время). Формат вывода даты функции `strftime()` отличается от функции `date()` (подробнее о нем следует читать на страницах документации языка PHP), в данном примере вы увидите на экране месяц и день недели, записанные русскими буквами. Правда по загадочным причинам функция возвращает текст в кодировке windows-1251. Для того чтобы преобразовать символы в кодировку utf-8, следует использовать функцию `iconv()` со следующими параметрами:

```
echo iconv("windows-1251", "utf-8", strftime("%A %B"));
```

Функция *mktime()*

Функция `mktime()` возвращает метку времени для заданной даты. Синтаксис ее такой:

```
int mktime([int hour[, int minute[, int second[, int month
           [, int day[, int year[, int is_dst]]]]]]])
```

> ***Примечание***
>
> Обратите внимание на неочевидный порядок аргументов, делающий неудобной работу с необязательными аргументами.

Функция `mktime()` возвращает метку времени UNIX, соответствующую дате и времени, которые заданы аргументами этой функции. Аргументы могут быть опущены в порядке справа налево. Отсутствующие аргументы считаются равными соответствующим компонентам локальной даты.

Аргумент *is_dst* может быть установлен в 1, если заданной дате соответствует летнее время, 0 — в противном случае, или –1 (значение по умолчанию), если неизвестно, действует ли летнее время на заданную дату. В последнем случае PHP пытается определить это самостоятельно.

Можно посчитать возраст человека, взяв дату рождения (листинг 5.1).

Листинг 5.1. Расчет возраста

```
<?
$sec = 10;
$min = 12;
$hour = 19;
$day = 21;
$month = 5;
$year = 1982;
// Теперь вычислим метку UNIX для указанной даты
$birthdate_unix = mktime($hour, $min, $sec, $month, $day, $year);
// Вычислим метку UNIX для текущего момента
$current_unix = time();
// Посчитаем разность меток $period_unix=$current_unix - $birthdate_unix;
// Возраст, измеряемый годами
$age_in_years = floor($period_unix / (365*24*60*60));
// Теперь выводим все это на экран
echo "Ваш возраст составляет $age_in_years.";
?>
```

Вот еще пример использования функций работы с датами. Выведем на экран дату двухдневной давности:

```
date("Y-m-d", time()-2*24*60*60);
```

Определение и вызов пользовательских функций

Программист может сам создать функцию, в документации по PHP такая функция называется *пользовательской*.

Функция объявляется с помощью оператора `function`. В PHP объявление функции может быть записано после вызова этой функции. Если функция принимает аргументы, они объявляются как переменные в объявлении функции.

В листинге 5.2 приведены описание и вызов функции, переводящей температуру в градусах Цельсия в градусы Фаренгейта.

Листинг 5.2. Определение и вызов функции

```
<?php
function c2f($temperature)
{
  return $temperature = round(($temperature - 32) * 5/9,2);
}
$t = 109;
echo c2f($t);
?>
```

Переменная `$temperature` создается при вызове функции, и ей присваивается значение, указанное в вызове этой функции. Оператор `return` используется для возврата значения из функции, это значение становится доступно вызывающему скрипту.

Передача параметров функции по ссылке

Обычно параметры передаются в функцию *по значению*. То есть копия значения переменной `$t` передается переменной `$temperature` при вызове функции, и изменения значения переменной `$temperature` никак не повлияют на переменную `$t`.

Можно передавать параметр и *по ссылке* (листинг 5.3), тогда изменения, происходящие с переменной `$temperature` при выполнении функции, отразятся и на переменной `$t`.

Листинг 5.3. Передача параметра в функцию по ссылке

```
<?php
function c2f(&$temperature)
{
  return $temperature = round(($temperature-32)*5/9,2);
}
$t = 109;
echo c2f($t);
echo "<br>".$t;
?>
```

Можно задать значение по умолчанию для параметра при описании функции. В следующем примере функция вызывается без передачи значения параметру, вместо него будет использоваться значение по умолчанию (листинг 5.4).

Листинг 5.4. Назначение параметра функции по умолчанию

```
<?php
function c2f($temperature = 32)
{
  return $temperature = round(($temperature-32)*5/9,2);
}
echo c2f();
?>
```

При этом, в отличие от PHP 4, в PHP 5 возможно задать значение по умолчанию для параметра, передаваемого по ссылке.

Функции и область действия переменной

Область действия (scope) переменных управляет тем, где переменная видима и применима.

- Переменные, которые объявлены внутри функции, действуют в области от оператора, в котором они объявлены, до закрывающей скобки в конце функции. Эта область называется *областью функции*, а такие переменные — *локальными переменными*.
- Переменные, которые объявлены вне функции, действуют в области от оператора, в котором они объявлены, до конца файла, но не внутри функций. Эта область называется *глобальной областью*, а такие переменные — *глобальными переменными*.

Надо отметить, что между переменной, определенной внутри функции, и одноименной переменной, определенной вне функции, нет никакой связи. Рассмотрим пример (листинг 5.5).

Листинг 5.5. Попытка использования глобальной переменной, не определенной внутри функции

```
<?php
function c2f()
{
  return $temperature = $temperature*32;
}
$temperature = 109;
echo c2f();
?>
```

Переменная `$temperature` внутри функции не определена, о чем мы и получаем сообщение в браузере. Соответственно, значение, возвращаемое функцией, будет равно 0.

Для того чтобы переменные внутри функции стали глобальными, следует использовать выражение `global` (листинг 5.6).

Листинг 5.6. Объявление глобальной переменной внутри функции

```
<?php
function c2f()
{
  global $temperature;
  return $temperature = $temperature*32;
}
$temperature = 10;
echo c2f();
?>
```

Теперь значение переменной `$temperature` доступно в функции, где его уже можно менять. Причем изменения, произошедшие с переменной внутри функции, скажутся и на значении переменной вне функции, т. е. в этом случае ситуация напоминает передачу параметра по ссылке, а не по значению.

Статические переменные

Есть и другой способ определить переменную так, чтобы ее значение хранилось вне функции, но было доступно также и внутри функции. Способ состоит в использовании *статических переменных*. Переменную можно объявить как статическую в начале функции до того, как будут заданы какие-либо команды. Эта переменная сохранит присвоенное ей значение после завершения работы функции.

Пример использования статической переменной представлен в листинге 5.7.

Листинг 5.7. Объявление статической переменной

```
<?php
function Test()
{
   static $a = 0;
   echo $a;
   $a++;
}
Test();
echo "<br>".$a;
Test();
?>
```

Переменная `$a` не действует в глобальной области — исполнение инструкции `echo` между вызовами функции приводит к сообщению об ошибке. Но, вызывая

функцию `Test()` второй раз, мы убеждаемся, что переменная `$a` сохранила свое значение. Статическая переменная как коробочка: в области функции она открыта, в глобальной области закрыта, но никуда не делась.

Повторное использование кода

Разработав полезную и красивую функцию, вы, возможно, захотите использовать ее в нескольких сценариях. Сохраним такую функцию в отдельном файле и посмотрим, какие средства предлагает нам PHP для повторного использования кода.

Операторы `include()` и `require()` загружают, анализируют и исполняют код из указанного файла.

Вот наша функция. Сохраним ее в файле f.php:

```
<?php
function c2f($temperature)
{
  return $temperature = round(($temperature-32)*5/9,2);
}
?>
```

А вот сценарий (сохраните его в отдельном файле), в котором подключается файл с функцией (листинг 5.8).

Листинг 5.8. Повторное применение кода с помощью инструкции `include`

```
<?php
include("f.php");
$t = 109;
echo c2f($t);
?>
```

Аналогично работает инструкция `require()`. Разница в том, что `require()`, не найдя указанный файл, выведет сообщение об ошибке и заставит сценарий прекратить работу, а `include()` в этом случае выведет предупреждение, а сценарий продолжит работу.

Использование операторов `require()` и `include()` не влияет на область действия переменных.

Операторы `include_once()` и `require_once()` включают и исполняют указанные файлы так же, как и операторы `require()` и `include()`. Различие состоит в том, что код, который уже был подключен, не будет подключаться повторно.

Инструкции `include_once()` и `require_once()` используются в случае, когда один и тот же файл может быть подключен не один раз в ходе исполнения скрипта, и хочется быть уверенным, что этого не произойдет и, как следствие, не возникнет проблем с повторным определением переменных и функций.

Глава 6

Массивы

Массив — это структура, в которой хранится упорядоченный набор значений. Хранящиеся в массиве значения называются *элементами* массива. Каждый элемент массива может содержать единственное значение, такое как текст или число, или другой массив.

Каждый элемент массива имеет связанный с ним *индекс* (или *ключ*), который используется для доступа к элементу. Ключ указывает место элемента в массиве.

Ключ может быть целочисленным или представлять собой набор символов, чаще всего строковых.

В PHP численно-индексированный массив можно создать с помощью конструкции `array()` следующим образом:

```
$zoo = array('лев', 'тигр', 'волк');
```

Обратите внимание, что конструкция `array()` не требует определения размера массива, т. е. указания того, сколько элементов хранится в массиве. Доступ к элементу массива осуществляется через имя массива и индекс элемента. Индекс пишется в квадратных скобках после имени массива. Нумерация элементов начинается с нуля. Так что распечатать значение первого элемента можно так:

```
echo $zoo[0];
```

Вывести на экран все элементы массива можно с помощью цикла (листинг 6.1).

Листинг 6.1. Перебор элементов массива

```
<?php
$zoo = array('лев', 'тигр', 'волк');
for ($i = 0; $i < 3; $i++)
echo "$zoo[$i]\n";
?>
```

Можно добавить новый элемент в массив на лету́:

```
$zoo[] = 'слон';
```

При этом не требуется указывать индекс для этого элемента — PHP автоматически увеличит на единицу значение последнего индекса и создаст новый элемент с новым индексом.

Можно даже с самого начала создавать массив, не указывая ни инструкции `array`, ни индексов:

```
$my_arr[] = 11;
$my_arr[] = 12;
```

В результате будет создан массив `$my_arr`, состоящий из двух элементов.

Функция `var_dump()` распечатывает структуру массива, но смотреть ее вывод удобнее в виде исходного кода, поскольку браузер удаляет символы конца строки и весь текст выводится подряд.

В ходе выполнения скрипта может быть создан массив заранее неизвестного размера. Тогда для удобной работы с циклом можно использовать функцию `count()`, возвращающую размер массива.

```
<?php
$zoo = array('лев', 'тигр', 'волк');
for ($i = 0; $i < count($zoo); $i++)
echo "$zoo[$i]\n";
?>
```

Ассоциативные массивы

Если вы идете по зоопарку и разглядываете животных, вам неважно, какой номер указан на клетке. Важнее, правильно ли указано название животного. Номер может и вовсе отсутствовать. Отсутствие номера не означает, что порядок расположения клеток не существенен или отсутствует.

Ассоциативные массивы в PHP организованы именно по этому принципу. В них в качестве ключа элемента можно применять данные различных типов даже для одного и того же массива. Обычно используют массивы, имеющие строковые ключи. Создадим такой массив:

```
$photo = array("name"=>"dog.jpg", "size"=>"130k", "type"=>"image/jpeg");
```

Здесь ключи `"name"`, `"size"` и `"type"` указывают на значения элементов массива.

Доступ к содержимому осуществляется через имя массива и ключ. Распечатаем элемент массива:

```
echo $photo["size"];
```

При добавлении элемента в ассоциативный массив необходимо явно указать ключ:

```
$photo["description"] = "Фотография моей собаки";
```

Для распечатки всех элементов ассоциативного массива нам потребуется цикл, при этом не удастся перебирать элементы по номерам — ключи в данном случае не являются числами!

Воспользуемся циклом `while`, условие выполнения которого определяется значением, возвращаемым функцией `each()` (листинг 6.2).

Листинг 6.2. Использование функции `each()` для просмотра ассоциативного массива

```
<?php
$photo = array("name"=>"dog.jpg", "size"=>"130k", "type"=>"image/jpeg",
               "description"=>"Фотография моей собаки");
while ($m = each($photo))
{
  echo $m["key"];
  echo " — ";
  echo $m["value"];
  echo "<br>";
}
?>
```

В начале цикла внутренний указатель помещается на первый элемент массива. Функция `each()` возвращает пару "ключ/значение" из массива и продвигает указатель массива вперед на следующий элемент. Ключ и значение возвращаются в виде массива из четырех элементов. Элементы с ключами `"0"` и `"key"` содержат имя ключа, а элементы с ключами `"1"` и `"value"` — значение элемента массива `$photo`.

Если внутренний указатель массива устанавливается на конец массива, функция `each()` возвращает `false` и цикл завершается.

Значения, возвращаемые функцией `each($photo)`, присваиваем массиву `$m` и распечатываем его элементы. Проверьте, что вместо ключей `"key"` и `"value"` можно использовать ключи `"0"` и `"1"`.

Похожим образом можно перебрать массив, используя конструкцию `foreach`. При старте `foreach` внутренний указатель массива автоматически устанавливается на первый элемент массива. При каждом проходе значение текущего элемента присваивается переменной `$value`, а внутренний указатель массива передвигается на следующий элемент:

```
foreach ($a as $value)
{
  echo "Элемент массива: $value<br>\n";
}
```

А можно даже так:

```
<?php
$photo = array("name"=>"dog.jpg", "size"=>"130k", "type"=>"image/jpeg");
foreach ($a as $key => $value)
{
  echo "Ключ: $key; Элемент массива: $value<br>\n";
}
```

Здесь ключ текущего элемента присваивается переменной `$key`. Имена переменных в этих примерах не являются зарезервированными словами. Вы можете использовать любые допустимые в PHP имена.

Многомерные массивы

В качестве элементов массива могут выступать не только скалярные величины, но и сами массивы. В этом случае получаются так называемые *многомерные массивы*.

Для инициализации многомерных массивов используются вложенные конструкции `array()`. Обход многомерных массивов достигается при помощи вложенных циклов.

Допустим, нам надо хранить классификационные сведения о цветках. Каждый цветок имеет уникальное имя; кроме этого, надо записать количество лепестков, количество пестиков и тычинок, цвет и т. д. Объединим сведения об этих растениях в двумерном массиве и распечатаем их в цикле (листинг 6.3).

Листинг 6.3. Двумерный массив

```
<?php
$flowers = array(
  array('name'=>'magnolia', 'лепестки'=>'8', 'тычинки'=>'5',
        'пестики'=>'2', 'цвет'=>'белый'),
  array('name'=>'primula', 'лепестки'=>'10', 'тычинки'=>'5',
        'пестики'=>'5', 'цвет'=>'алый'),
  array('name'=>'campanula latifolia', 'лепестки'=>'10',
        'тычинки'=>'5', 'пестики'=>'3', 'цвет'=>'фиолетовый')
);
for ($row = 0; $row < 3; $row++)
{
  echo "|".$flowers[$row]["name"]."|".$flowers[$row]["лепестки"]."|".
      $flowers[$row]["тычинки"]."| <br>";
}
?>
```

Рассмотрим другой пример. Мы пишем сценарий фотоальбома, в который клиент может закачать свои фотографии и разрешить их просмотр другим посетителям сайта. При закачивании файлов с клиентской машины на сервер возникает необходимость собрать, возможно, более полные сведения об этих файлах. Сведения будут храниться в массиве `$_FILES`. Каждый элемент `$_FILES['userfile']`, где `'userfile'` — имя файла, в свою очередь будет массивом, ключи которого — параметры файла: оригинальное имя файла на клиентской машине, MIME-тип файла, размер загруженного файла в байтах, временное имя файла, под которым загруженный файл был сохранен на сервере. Для этого нам понадобится двумерный массив:

- `$_FILES['userfile']['name']` — оригинальное имя файла на компьютере клиента;
- `$_FILES['userfile']['type']` — MIME-тип файла, в случае, если браузер предоставил такую информацию, например: `"image/gif"`;
- `$_FILES['userfile']['size']` — размер в байтах принятого файла;

- `$_FILES['userfile']['tmp_name']` — временное имя, с которым принятый файл был сохранен на сервере;
- `$_FILES['userfile']['error']` — код ошибки, которая может возникнуть при загрузке файла.

Функции для работы с массивами

Существуют многочисленные встроенные в PHP функции для работы с массивами. Они служат для переупорядочения элементов, сравнения массивов, выборки части массива и т. п.

Одной из задач при создании сайтов электронной коммерции является привлечение внимания к фирме. Для этого на страницах сайта можно публиковать материалы, которые обновляются при каждом посещении. Например, каждый раз, когда посетитель заходит на сайт туристической фирмы, он видит новую фотографию курортного города или отеля.

Для этого необходимо иметь коллекцию рисунков и случайным образом выбирать часть из них при каждой загрузке страницы. Используем для перемешивания массива в случайном порядке функцию `shuffle()` (листинг 6.4).

Листинг 6.4. Перемешивание элементов массива

```
<?php
  $jpgs = array ("Kiev.jpg", "Novgorod.jpg", "Paris.jpg", "London.jpg",
                 "Vienna.jpg", "Berlin.jpg", "Sochi.jpg");
  shuffle($jpgs);
?>
<html>
<head>
  <title> Туристическая фирма </title>
</head>
<body>
<table>
  <tr>
  <?php
  for ($i = 0; $i < 3; $i++)
  {
    echo "<td ><img border=1 src=\"$jpgs[$i]\" ";
    echo "width=200 height=120></td> \n";
  }?>
  </tr>
</table>
</body>
</html>
```

Двойные кавычки, необходимые вокруг имени файла, должны помещаться после символа обратного слэша. Кавычки вокруг остальных значений атрибутов убраны для лучшей читаемости кода, хотя это и не соответствует стандарту HTML. Разобравшись с работой скрипта, вы можете исправить это несоответствие.

Функция `list()` позволяет одновременно присвоить значения нескольким переменным, не определенным ранее. В качестве значений они получают элементы массива:

```
<?php
$components = array ('Intel', '1 Gb', '40 Gb');
list($cpu, $ram, $hdd) = $components;
echo 'Процессор: '.$cpu.'<br> Память: '.$ram.'<br>Жесткий диск: '.$hdd;
?>
```

Функции `each()` и `list()` можно использовать для создания более элегантного перебора всех элементов двумерного массива:

```
<?php
for ($row = 0; $row < 3; $row++)
{
  while (list($key, $value) = each ($flowers[$row]))
  {
    echo "|".$value;
  }
  echo "|<br>";
}?>
```

Полезной является и функция `explode()`, позволяющая разбить строку на несколько фрагментов, каждый из которых помещается в отдельный элемент массива:

```
<?php
$str = "login | password | email | tel_number | address";
$account = explode("|", $str);
for ($i = 0; $i < count($account); $i++)
{
  echo $account[$i]."<br />";
}
?>
```

Существует и обратная функция — `implode()`, которая объединяет элементы массива в строку, используя для этого заданный разделитель:

```
<?php
$account[] = "login";
$account[] = "password";
$account[] = "email";
$account[] = "tel_number";
$account[] = "address";
$str = implode(" – ", $account);
echo $str;
?>
```

Автоглобальные массивы

Сервер Apache и модуль PHP обмениваются между собой данными в соответствии со стандартом CGI, в котором предусмотрена передача переменных и их значений.

Имена переменных определены стандартом, а значения формируются из нескольких источников:

- значения директив конфигурационного файла httpd.conf;
- заголовки запросов клиента и ответа сервера.

Apache передает в распоряжение модуля PHP много информации, получить доступ к которой скрипт может с помощью ассоциативных массивов, называемых *предопределенными* или *автоглобальными массивами*.

Автоглобальные массивы предназначены для хранения данных, полученных из ненадежного источника, т. е. из клиентского ввода, сведений о системе и настройках сервера. Эти данные содержатся в переменных, называемых *переменными окружения* (environment — окружающая среда). Apache и PHP существуют в среде, состоящей из переменных и их значений. Для того чтобы узнать, какие переменные окружения доступны вашему скрипту, вызовите функцию `phpinfo()` без параметров. Эта функция выводит в браузер множество сведений о вашей конфигурации PHP. Изучите раздел `Environment` — вы найдете список доступных в данной версии переменных окружения. В конце страницы, созданной функцией `phpinfo()`, вы увидите информацию о структуре и значениях элементов массива `$_SERVER`. Элементы автоглобальных массивов доступны в любой части скрипта. Если директива `register_globals` отключена в конфигурации PHP (такая настройка, начиная с версии 4.1, является стандартной), то на сервере будут сформированы следующие ассоциативные автоглобальные массивы.

- Массив `$_SERVER` содержит информацию о заголовках запросов, путях и каталогах для хранения скриптов. Существенная часть данных этого массива определена в спецификации CGI, но точный перечень предоставляемой информации определяется самим сервером.

  Ключами этого массива являются, например:

  - `'SERVER_ADDR'` — IP-адрес сервера, на котором выполняется сценарий;
  - `'SERVER_NAME'` — имя сервера, на котором выполняется сценарий;
  - `'SERVER_SOFTWARE'` — строка, идентифицирующая сервер, которую он помещает в HTTP-ответ сервера;
  - `'SERVER_PROTOCOL'` — имя и версия используемого протокола;
  - `'REQUEST_METHOD'` — используемый метод запроса;
  - `'DOCUMENT_ROOT'` — корневой каталог документов Web-сервера, значение определяется в директиве в файле httpd.conf;
  - `'HTTP_ACCEPT'` — содержимое заголовка `Accept` текущего запроса;
  - `'HTTP_REFERER'` — адрес страницы, с которой клиентский браузер пришел по ссылке, если таковая имеется;
  - `'HTTP_USER_AGENT'` — содержимое заголовка `User Agent` текущего запроса;

- `'REMOTE_ADDR'` — IP-адрес клиентской машины;
- `'SERVER_PORT'` — порт на серверной машине, используемый Web-сервером для связи. По умолчанию — 80;
- `'SCRIPT_NAME'` — путь от корня каталога с документами сервера до текущего скрипта. Вы можете узнать значение любого элемента этого массива:

```
<?php
echo $_SERVER['SCRIPT_NAME' ].'<br>';
 echo $_SERVER['DOCUMENT_ROOT'];
?>
```

- ❒ `$_ENV` — переменные, импортированные в PHP из окружения, в котором работает PHP. Их имена и значения определяются операционной системой;
- ❒ `$_COOKIE` — массив переменных, передаваемых скрипту через HTTP cookies. Ключом служит имя cookie;
- ❒ `$_GET` — массив переменных, передаваемых скрипту через запрос методом `GET`;
- ❒ `$_POST` — массив переменных, передаваемых скрипту через запрос методом `POST`;
- ❒ `$_FILES` — массив данных о файлах, загружаемых на сервер методом `POST`;
- ❒ `$_REQUEST` — массив, содержащий данные массивов `$_GET`, `$_POST` и `$_COOKIE`;
- ❒ `$_SESSION` — массив, содержащий переменные сессии;
- ❒ `$GLOBALS` — массив, содержащий ссылки на все переменные, определенные в скрипте как глобальные. Имена переменных являются ключами этого массива.

Мы научимся использовать эти массивы в следующих главах, когда начнем рассматривать передачу данных от клиента на сервер с помощью форм, загрузку файлов и работу с сеансами и куками.

Глава 7

Передача данных через HTML-формы

Получение данных от клиента часто происходит с помощью HTML-форм. Форма создается парным тегом `<form>`.

Теги формы

Открывающий тег `<form>` имеет следующие атрибуты:

- `method` — метод посылки сообщения с данными формы. Можно указать `GET` или `POST`;
- `enctype` — тип передаваемой информации в стандарте MIME;
- `action` — URL, который будет вызываться для обработки формы.

Внутри тега `<form>` можно использовать несколько типов тегов для обозначения различных полей ввода.

Тег *<input>*

Непарный тег `<input>` используется для ввода одной строки текста или одного слова. В зависимости от значения атрибута `type` этого тега он позволяет ввести текст в поле, осуществить одиночный или множественный выбор, создать кнопку и т. д. После выполнения действий пользователем на сервер будет отправлена строка "*имя=значение*", где *имя* определяется значением атрибута `name`, а *значение* — либо значением, указанным в атрибуте `value`, либо текстом, введенным пользователем. Атрибут `type` может принимать значения:

- `text` — это значение атрибута позволяет указать однострочное поле ввода. В этом случае в теге `<input>` также можно указать размер на экране отображаемого поля ввода с помощью атрибута `size`, и максимальную длину вводимого значения в символах `maxlength`. Кроме того, обязательно надо указать атрибут `name` для идентификации поля ввода и необязательно — атрибут `value`, значение которого — это значение по умолчанию, выводимое в поле формы при ее загрузке в браузер пользователя. Например:

```
<form action="script.cgi">
  Введите имя:
  <input type="text"   name="regname">
  <input type="submit" value="OK">
</form>
```

- `password` — поле для ввода пароля. Аналогично текстовому полю, за исключением того, что символы, набираемые пользователем, будут отображаться на экране в виде звездочек. Например:

```
<form action="script.cgi">
  Введите пароль:
  <input type="password" name="password1" maxlength="8"><br>
  <input type="submit"   value="OK">
</form>
```

- `radio` — переключатель, который имеет атрибуты `name` и `value`. Атрибут `name` задает его имя, а атрибут `value` — значение. Можно указать параметр `checked`, который указывает на то, что этот тег переключателя будет отмечен по умолчанию при загрузке страницы. Переключатели также можно объединять в группы, для этого они должны иметь одно и то же имя. Например:

```
<input type="radio" name="age" value="lt20" checked>До 20
<input type="radio" name="age" value="20_50">20-50
<input type="radio" name="age" value="gt50">старше 50
```

- `checkbox` — флажок, в отличие от переключателя можно выбрать несколько вариантов из предлагаемого списка. Например:

```
<input type="checkbox" name="pay1" value="cash" checked>наличные
<input type="checkbox" name="pay2" value="card">кредитная карта
<input type="checkbox" name="pay3" value="check">чек
```

  Сразу следует обратить внимание на то, что в случае, когда атрибут `type` имеет значение `checkbox`, на сервер будут отправлены только те пары "*имя=значение*", которые были отмечены пользователем.

- `hidden` — скрытое текстовое поле. Оно позволяет передавать сценарию какую-то служебную информацию, не отображая ее на странице. Скрытое поле имеет атрибуты `name` и `value`. Например:

```
<input name="email" type="hidden" value="info">
```

- `submit` — кнопка отправки формы. В случае применения значения `submit` значение атрибута `value` используется для надписи на кнопке. Атрибут `name` необходим, если кнопка не одна, а несколько. После нажатия кнопки сценарию передается строка *"имя=текст кнопки"*, а также все остальные данные формы, описанные выше. Например:

```
<input type="submit" value="Отправить">
```

- `reset` — кнопка, при нажатии на которую все поля формы примут значения, описанные для них по умолчанию. Например:

```
<input type="reset" value="Отменить">
```

Тег <*select*>

Парный тег `<select>` позволяет сделать выбор в раскрывающемся списке одного из значений, заданных при помощи тега `<option>`. Имя отправляемого параметра задается атрибутом `name` тега `<select>`, а значение параметра — атрибутом

value тега <option>. Если в теге <option> указан параметр selected, то строка будет изначально выбранной. Атрибут size тега <select> задает, сколько строк будет занимать список. Если size равен 1, то список будет раскрывающимся.

Пример:

```
<select name="transport" size=3>
  <option selected value="air">air
  <option value="bus">bus
  <option value="train">train
  <option value="car">car
</select>
```

Тег *<textarea>*

Многострочное поле ввода текста начинается с парных тегов <textarea></textarea>. Атрибут name задает имя многострочного поля. Также можно указать ширину поля (cols) и число строк (rows). Если необходимо, чтобы в многострочном поле ввода изначально находился какой-то текст, то его следует поместить между тегами <textarea></textarea>.

Пример:

```
<textarea name=wish" cols="40" rows="3">Текст по умолчанию</textarea>
```

Работа с формами в PHP

Все данные, введенные пользователем в поля формы, отправляются на Web-сервер в текстовом формате в тот момент, когда пользователь щелкает по кнопке submit.

Рассмотрим Web-страничку, содержащую простую форму (листинг 7.1).

Листинг 7.1. Простая HTML-форма

```
<html>
<head>
   <title>Простая форма</title>
</head>
<body>
  <form method="GET" action="1.php">
    Введите текст:
    <input type="text" name="new" value="">
    <input type="submit" value="Готово!">
  </form>
</body>
</html>
```

Данные из этой формы будут переданы методом `GET` Web-серверу после щелчка по кнопке с надписью "Готово!" Web-сервер вызовет сценарий 1.php на исполнение и передаст ему данные из формы.

Web-сервер передаст полученные данные скрипту, имя которого указано в атрибуте `action` — 1.php, в виде элемента массива `$_GET`. Этот ассоциативный массив создается на сервере для хранения данных, полученных от клиента по методу `GET`. В нашем случае в массиве создастся элемент с ключом `'new'`.

Таким образом, чтобы прочитать переданные данные, скрипту необходимо выполнить такие действия:

```
$our_var = $_GET['new'];
```

Надо заметить, что не при всех настройках модуля PHP передача данных происходит таким способом. До версии PHP 4.1 общепринятой была так называемая автоматическая генерация переменных, т. е. после получения данных из формы на сервере генерировалась переменная `$new` со значением, введенным клиентом в поле формы. Однако это приводило к тому, что пользователь из текста исходного кода формы мог узнать имя переменной и использовать его в своих целях.

Для предотвращения автоматической генерации переменных в файле php.ini существует директива `register_globals`. Если ее значение — `Off` (по умолчанию начиная с версии PHP 4.1), то автоматическая генерация переменных не производится.

Для демонстрации работы с формой создайте файл формы с расширением html и скрипт с именем, указанным в атрибуте `action`. Скрипт может выглядеть так:

```
<?php
$our_var = $_GET['new'];
echo $our_var;
?>
```

Сначала откройте в браузере форму, введите в нее данные и отправьте данные в скрипт щелчком по кнопке **Готово!**. Данные будут переданы в скрипт, он выполнится и выведет на экран значения из формы. Обратите внимание, что данные передаются в скрипт в адресной строке браузера. Русские буквы и пробелы кодируются при передаче.

Если вы забудете указать в форме метод передачи данных, то по умолчанию будет использован метод `GET`. Можно применить и метод `POST`; сделайте это, указав его в значении атрибута `action`. При этом введенные данные попадут в массив `$_POST` на сервере, и именно из него и нужно их теперь извлекать:

```
$our_var = $_POST['new'];
```

При использовании метода `POST` данные передаются в теле запроса и не видны в адресной строке. Кроме того, существует массив `$_REQUEST`, содержащий данные форм, переданные любым методом — `GET` или `POST`.

Перед использованием переменной, получившей значение из массивов `$_GET` или `$_POST`, необходимо проверить, получила ли эта переменная значение, т. е. установлена ли она, с помощью функции `isset()`:

```
if (isset($our_var)) echo 'Вот что мы узнали: $our_var <br/>';
```

Если же приходится объявлять переменную, не имея уверенности, что в массивах `$_GET` или `$_POST` найдется искомый элемент, то следует защититься от вывода сообщений об ошибке на экран оператором `@`, который подавляет этот вывод. Проверьте на практике, как это работает:

```
@$our_var = $_GET['new'];
```

Можно создать скрипт, содержащий форму и обеспечивающий передачу данных обратно в этот же скрипт (листинг 7.2).

Листинг 7.2. Передача данных формы обратно в сценарий

```
<form >
  Назовите ваше имя: <input name="name"><br>
  <input type="submit" value="OK">
</form>
<?php
@$name = $_GET['name'];
if (isSet($name))
{
  echo "Привет, $name!";
}
?>
```

Создадим форму с тегом `<select>` с динамической генерацией значений поля `option` (листинг 7.3).

Листинг 7.3. Динамическая генерация элементов формы

```
<form>
<select name="years">
<?php
$year = 2000;
for ($i = 0; $i <= 50; $i++)
{
  $new_years = $year + $i;
  echo '<option value='.$new_years.'>'.$new_years.'</option>';  }
?>
</select>
<input type="submit" value="OK">
</form>
```

Данные, полученные от клиента, могут содержать не только полезную информацию, но и тексты программ, предназначенных для взлома сервера. Первое, что следует делать с данными из формы — фильтровать их с помощью специальных функций.

Например, функция `htmlspecialchars()` преобразует специальные символы (знаки "больше", "меньше", амперсанд и др.) в HTML-сущности.

Если данные, полученные из формы, должны иметь определенный тип, то их можно преобразовать к этому типу, тем самым обезопасив сценарий от ввода некорректных значений:

```
$number = intval($number);
$stroka = strval($stroka);
```

Передача данных из полей *checkbox*

При передаче данных из формы, содержащей поле `checkbox`, можно выбрать несколько вариантов ответа. Для того чтобы все они были переданы на сервер, надо иметь несколько имен переменных:

```
<form>
  <input type="checkbox" name="pay1" value="cash" checked>наличные
  <input type="checkbox" name="pay2" value="card">кредитная карта
  <input type="checkbox" name="pay3" value="check">чек
  <input type="submit" value="Отправить">
</form>
```

Тогда на сервере потребуется генерировать три переменные. Вместо этого можно создавать массив:

```
<form>
  <input type="checkbox" name="pay[]" value="cash" checked>наличные
  <input type="checkbox" name="pay[]" value="card">кредитная карта
  <input type="checkbox" name="pay[]" value="check">чек
  <input type="submit" value="Отправить">
</form>
```

Обратите внимание, что на сервере будет создан массив из такого количества элементов, сколько их было выбрано пользователем в форме.

Глава 8

Работа с файлами

Запись и считывание данных из файла реализуются в три шага:

1. Открытие файла. Если файл еще не существует, его нужно создать. Если файл не может быть открыт, эта ситуация должна быть распознана, и следует предусмотреть корректный выход из нее.
2. Запись или считывание данных.
3. Закрытие файла.

Открытие файла

Для открытия файла в PHP используется функция `fopen()`. При открытии файла необходимо указать, как его предполагается использовать. Это называется *режимом файла*. Серверная операционная система должна знать, что нужно делать с открываемым файлом. Режимы файла предоставляют операционной системе механизм для определения способа обработки запросов на доступ, поступающих от других пользователей или от сценариев, а также метод проверки наличия доступа и прав для работы с конкретным файлом.

Первым параметром функции `fopen()` должно быть имя файла, который необходимо открыть, а вторым — код режима. Режимы описаны в табл. 8.1.

Таблица 8.1. *Возможные режимы для функции* `fopen()`

Режим	Описание
`r`	Открывает файл только для чтения; помещает указатель в начало файла
`r+`	Открывает файл для чтения и записи; помещает указатель в начало файла
`w`	Открывает файл только для записи и помещает указатель в начало. Если файл не существует — пробует его создать
`w+`	Открывает файл для чтения и записи и помещает указатель в начало файла. Если файл не существует — пробует его создать
`a`	Открывает файл только для записи; помещает указатель в конец файла. Если файл не существует — пытается его создать
`a+`	Открывает файл для чтения и записи; помещает указатель в конец файла. Если файл не существует — пытается его создать

Таблица 8.1 *(окончание)*

Режим	Описание
x	Создает и открывает файл только для записи; помещает указатель в начало файла. Если файл уже существует, вызов `fopen()` закончится неудачей. Если файл не существует, попытается его создать
x+	Создает и открывает файл для чтения и записи; помещает указатель в начало файла. Если файл уже существует, вызов `fopen()` закончится неудачей. Если файл не существует, попытается его создать

При работе в операционной системе Windows также можно явно указать, содержит ли файл двоичные (`b`) или текстовые (`t`) данные. В этом случае следует добавить `"b"` или `"t"` к строке режима. Если вы записываете текстовые данные в файл в Windows и укажете режим `"a+t"`, то строки будут разделяться символами `\n\r`, как и положено в Windows.

В случае удачного открытия файла функция `fopen()` возвращает дескриптор файла, в случае неудачи — `false`. Дескриптор файла служит идентификатором файла и используется операционной системой для операций с этим файлом. Возвращенный функцией дескриптор необходимо использовать во всех функциях, которые в дальнейшем будут работать с этим файлом.

```
$fp = fopen("richmond.txt", "rt");
```

Сценарий, пытающийся открыть файл, должен иметь необходимые для этого права. Другой, более редкой ошибкой при попытке открыть файл может быть отсутствие места на диске. Какова бы ни была причина, по которой не удается открыть файл, устранить ее может только администратор сервера, на котором лежат скрипты.

Права доступа к файлу

Права на работу с файлами устанавливаются администратором сервера, и, если вам не удается создать или открыть существующий файл, следует выяснить, как обстоит дело с раздачей прав на вашем сервере.

Обычно Web-мастеру, даже пишущему свой проект в системе Windows, приходится изучать механизм определения прав доступа к файлу, реализованный в UNIX. Приведенная далее информация о правах доступа никоим образом не является исчерпывающей, но должна служить стимулом узнать больше о принципах раздачи прав.

В системе семейства UNIX установлены три вида прав доступа к файлам и каталогам:

- на чтение (r — *read*);
- на запись (w — *write*) (т. е. модификация данных в файле и удаление файла);
- на исполнение (x — *execute*).

Есть три типа пользователей, относительно которых принимается решение о назначении прав доступа:

- пользователь-владелец файла (user);
- группа пользователей-владельцев (group), которым устанавливаются права, отличные от прочих пользователей;
- прочие пользователи (other).

user			group			other		
r	w	x	r	w	x	r	w	x
1	1	1	1	1	0	1	0	0

Рис. 8.1. Запись прав доступа в метаданных

Пользователь-владелец может быть, а может и не быть членом группы-владельца. У каждого файла есть область метаданных — область памяти, где хранится информация о файле, в том числе и данные о правах доступа. Если какое-либо из перечисленных прав установлено, то в соответствующую ячейку записывается "1", а если не установлено, то "0". Запись прав доступа в этих метаданных для каждого файла выглядит так, как представлено на рис. 8.1.

Если представить права доступа в приведенном здесь примере в виде восьмеричных чисел, а именно так их и записывают, то это будет число 764.

На сервере, работающем под управлением UNIX, можно вызвать функцию `chmod()` для установки права доступа к файлу или каталогу (от англ. *change mode* — изменить режим):

```
chmod("/somedir/somefile", 0755);
```

Обратите внимание, что права доступа записываются с ведущим нулем — он указывает на то, что записано восьмеричное число.

Установить все права на файл image1.jpg для пользователя-владельца и лишить их всех остальных можно так:

```
chmod("image1.jpg", 0700);
```

В Windows эта функция, естественно, работать корректно не может. На сайте **php.net** имеются ссылки на аналогичную функцию, разработанную для Windows сторонней организацией, но использовать ее надо с осторожностью.

Скрипт, содержащий функции для работы с файлами, впрочем, как и любой другой PHP-скрипт, запускается от имени пользователя, вовсе не располагающего всеобъемлющими правами в системе. Объем его прав зависит от конфигурации сервера и политики конкретного администратора.

В приведенном ранее примере с помощью функции `fopen()` открывался файл, находящийся в том же каталоге, что и сценарий, в котором происходит вызов этой функции. Можно указать путь относительно этого каталога:

```
$fp = fopen("catalog/some.txt", "w+");
```

Помните, что создать несуществующий файл эта функция при наличии прав может, а создать несуществующий каталог — нет.

Запись в файл

Для этого можно воспользоваться функцией `fwrite()`. Функция `fwrite()` принимает два обязательных параметра — дескриптор файла и строку, которую надо записать в этот файл:

```
fwrite($fp, $data_string[, $length]);
```

Третий необязательный параметр представляет собой максимальное количество байтов, которые требуется записать. При передаче этого параметра функция `fwrite()` будет записывать строку в файл, указанный параметром `$fp`, пока не встретит конец строки или не запишет столько байтов, сколько указано в третьем параметре. Функция `fwrite()` возвращает количество записанных байтов или `false` в случае ошибки.

Закрытие файла

По завершении использования файла его следует закрыть при помощи функции `fclose()`:

```
fclose($fp);
```

Эта функция возвращает значение `true` в случае успешного закрытия файла и `false`, если файл не был закрыт. В листинге 8.1 дан пример открытия файла для записи, запись в него одной строки и закрытие файла.

Листинг 8.1. Запись данных в файл

```
<?php
$fp = fopen("1.txt", "w");
$contents = "Строка, которая будет записана в файл. \n";
$res = fwrite($fp, $contents);
fclose($fp);
?>
```

Если по каким-то причинам открыть файл не удалось, в браузере пользователя появится сообщение об ошибке, пугать которым пользователя не рекомендуется. Для исключения возможности такого сообщения лучше обрабатывать вероятные ошибки в сценарии (листинг 8.2).

Листинг 8.2. Обработка ошибок при открытии файла

```
<?php
@$fp = fopen("1.txt", "w");
if (!$fp)
{
  echo "<p>Ваш заказ не удается обработать сейчас.
          Пожалуйста, повторите его позднее.</p>";
  exit;
}
$contents = "Строка, которая будет записана в файл. \n";
$res = fwrite($fp, $contents);
fclose($fp);
?>
```

Считывание данных из файла

Для чтения строки из открытого файла можно применить функцию `fgets()`. Первый параметр этой функции — указатель на файл, а второй (*`length`*) — длина строки в байтах.

`fgets()` возвращает строку размером *`length`* – 1 байт, прочитанную из указанного файла. Чтение заканчивается, когда количество прочитанных байтов достигает *`length`* – 1, когда будет достигнут конец строки или конец файла, что встретится первым. Если длина строки не указана, по умолчанию ее значение равно 1 Кбайт.

В результате выполнения сценария листинга 8.3 из файла удастся прочитать только одну строку.

Листинг 8.3. Чтение строки из файла

```
<?php
$fp = fopen("1.txt", "r");
$contents = fgets($fp, 100);
echo $contents;
fclose($fp);
?>
```

При открытии файла внутренний указатель устанавливается на начало файла и передвигается по файлу по мере чтения. Для чтения всего файла нам потребуется функция `feof($fp)`, которая проверяет, не достиг ли указатель конца файла (листинг 8.4).

Листинг 8.4. Чтение файла целиком

```
<?php
@$fp = fopen("1.txt", "r");
if (!$fp)
{
  echo "<p>Ваш заказ не удается обработать сейчас.</p>";
  exit;
}
while (!feof($fp))
{
  $contents = fgets($fp);
  echo $contents;
}
fclose($fp);
?>
```

Другая функция, которую можно применить для считывания из файла, — `fread()` — читает столько байтов из файла, сколько задано во втором аргументе, если раньше не встретится конец файла (листинг 8.5).

Листинг 8.5. Чтение из файла с помощью функции `fread()`

```
<?php
$fp = fopen("1.txt", "r+");
$contents = fread($f, 7);
echo $contents;
fclose($fp);
?>
```

Кроме описанных базовых функций для работы с файлами, в PHP есть несколько функций, объединяющих действия, выполняемые базовыми функциями. Например, функция `readfile()` открывает файл, читает все его содержимое и выводит содержимое файла в окно браузера:

```
readfile("1.txt");
```

Функция `file_get_contents()` позволяет получить содержимое файла в виде одной строки:

```
echo file_get_contents("../firma/f.sql");
```

Блокировка файла

Попытка двух сценариев одновременно внести записи в один и тот же файл может привести к непредсказуемым результатам. Идеальным решением этой проблемы в случае, когда объем данных велик, является хранение данных не в файлах, а на сервере баз данных, который имеет механизм обеспечения целостности данных, но в случае работы с небольшими файлами целесообразно использовать функции блокировки.

Применяются два вида блокировки:

- *блокировка чтения* — этот общепринятый русский термин не передает смысла действия. Суть в том, что этот вид блокировки запрещает двум и более приложениям вести одновременную запись данных, но чтение файла несколькими приложениями одновременно в этом режиме разрешено;
- *блокировка записи* — режим, при котором чтение и запись разрешены только одному приложению.

Функция `flock()` осуществляет блокировку файла, идентификатор которого указан в ее первом аргументе. Второй ее аргумент содержит код режима блокировки (табл. 8.2).

Таблица 8.2. *Допустимые значения второго аргумента функции* `flock()`

Значение	Описание
LOCK_SH	Блокировка чтения
LOCK_EX	Блокировка записи
LOCK_UN	Снятие существующей блокировки и предотвращение других попыток блокирования

Пример записи в файл с применением блокировки дан в листинге 8.6.

Листинг 8.6. Блокировка файла

```
<?php
$fp = fopen("lock.txt", "w+");
if (flock($fp, LOCK_EX))
{
  fwrite($fp, "записываемая строка\n");
  flock($fp, LOCK_UN);     // сброс блокировки
}
else
{
  echo "Ошибка при попытке блокировки!";
}
fclose($fp);
?>
```

Функции для работы с каталогами

В сценарии, созданном для администрирования сайта, наверняка придется создавать или удалять файлы и каталоги. Это можно сделать с помощью следующих функций:

- `copy("a.txt", "aa.txt")` — создание файла aa.txt, являющегося копией файла a.txt;
- `rename("aa.txt", "aaa.txt")` — переименование файла aa.txt в файл aaa.txt;
- `unlink("ab.txt")` — удаление файла ab.txt;
- `mkdir(("cat1", 0764)` — создание каталога cat1 с правами доступа к нему 764;
- `chdir("./cat1")` — переход из текущего каталога, обозначаемого символом точки, в каталог cat1.

Глава 9

Строковые функции и регулярные выражения

Строки в PHP

Функции преобразования строк делятся на несколько типов в соответствии с целями их использования:

- обработка строк, полученных от клиента;
- вывод данных на экран;

  преобразование строк для последующего анализа, например выделение фрагментов строк, объединение или разделение на части.

Преобразование данных формы

Данные, полученные от клиента, перед записью в файл или базу данных необходимо обработать так, чтобы исключить возможность нарушения работы сценария в случае ввода некорректных значений. Часто при вводе данных пользователь не замечает пробелы, которые накапливаются в начале или в конце строки. Иногда эти пробелы мешают обработке. Функция `trim()` удаляет пробелы в начале и в конце строки и возвращает результирующую строку:

```
$feedback = trim($feedback);
```

При этом символами пробелов считаются символы новой строки (`\n`), возврата каретки (`\r`), горизонтальной (`\t`) и вертикальной табуляции (`\v`), конца строки (`\0`) и обычные пробелы.

Некоторые символы, встретившиеся в строке, могут привести к проблемам, в особенности при вставке в базу данных, поскольку она может интерпретировать их как управляющие символы. Такими символами являются кавычки (одинарные и двойные), обратный слэш (`\`) и символ `NULL`.

Для того чтобы показать, что требуется буквальное представление специального символа, а не его интерпретация в качестве управляющей последовательности, перед символом добавляется обратный слэш.

В *главе 23*, посвященной работе с базами данных, мы подробнее опишем функции, которые в настоящее время рекомендуется использовать для обеспечения безопасности Web-приложений.

В HTML некоторые символы имеют специальное значение и для сохранения своего значения должны быть преобразованы в HTML-сущности. Функция `htmlspecialchars()`

возвращает строку, над которой проведены некоторые из таких преобразований. Этих преобразований достаточно для большинства задач Web-программирования. Функция выполняет следующие замены:

- `&` (амперсанд) преобразуется в `&`;
- `"` (двойная кавычка) преобразуется в `&quot;`;
- `'` (одиночная кавычка) преобразуется в `&#039;`;
- `<` (знак "меньше чем") преобразуется в `<`;
- `>` (знак "больше чем") преобразуется в `>`.

Если вам нужно преобразовать все специальные символы используйте функцию `htmlentities()`. Эта функция делает даже больше преобразований, чем хотелось бы русскоязычному Web-мастеру, поскольку заменяет даже русские буквы.

Функция `nl2br()` (от англ. *new line to break*) принимает строку в качестве параметра и заменяет в ней все символы новой строки на тег `<br>` HTML. Это полезно, например, если вы хотите разбить длинную запись в гостевой книге на несколько строк.

Форматирование строк для печати

PHP поддерживает функцию `print()`, которая выполняет ту же задачу, что и `echo`, но поскольку она является функцией, то возвращает значение (0 или 1, в зависимости от успешности выполнения).

Обе эти конструкции выводят строку как есть. Но используя функции `printf()` и `sprintf()`, можно применять более сложное форматирование. В основном эти функции работают практически одинаково, но `printf()` выводит отформатированную строку в окне браузера, а `sprintf()` возвращает отформатированную строку.

В качестве первого параметра обеим функциям передается строка формата, описывающая основную форму вывода, в которой вместо переменных используются описатели типа данных. Остальными параметрами функций являются переменные, которые будут подставляться в строку формата.

Каждый описатель типа состоит из символа процента, за которым следует один из следующих символов:

- `b` — аргумент трактуется как целое и выводится в виде двоичного числа;
- `c` — аргумент трактуется как целое и выводится в виде символа с соответствующим кодом ASCII;
- `d` — аргумент трактуется как целое и выводится в виде десятичного числа со знаком;
- `e` — аргумент трактуется как `float` и выводится в научной нотации (например, `1.2e+2`);
- `u` — аргумент трактуется как целое и выводится в виде десятичного числа без знака;
- `f` — аргумент трактуется как `float` и выводится в виде десятичного числа с плавающей точкой;
- `o` — аргумент трактуется как целое и выводится в виде восьмеричного числа;
- `s` — аргумент трактуется как строка;

- `x` — аргумент трактуется как целое и выводится в виде шестнадцатеричного числа (в нижнем регистре букв);
- `X` — аргумент трактуется как целое и выводится в виде шестнадцатеричного числа (в верхнем регистре букв).

Например:

```
<?php
$num = 5;   $txt = "Текстовая строка";
$format = "Вывод числа: %d . Вывод строки:  %s <br>";
printf($format, $num, $txt);
?>
```

В дальнейшем мы будем использовать именно этот простой синтаксис параметров, хотя из документации вы можете узнать, что возможно более гибкое задание типов данных.

Функции изменения регистра строки

Чтобы привести строки к определенному регистру, используются следующие функции:

- `strtoupper()` преобразует символы строки в прописные буквы;
- `strtolower()` преобразует символы строки в строчные буквы;
- `ucfirst()` делает первый символ строки прописным, если он является алфавитным;
- `ucwords()` делает прописным первый символ каждого слова в строке, которая начинается с алфавитного символа.

Среди функций для работы со строками встречаются такие, которые по-разному работают с текстами в различных кодировках. Например, функция `strtoupper()` корректно преобразует строки в однобайтных кодировках (windows-1251), но для работы с UTF-8 вместо этой функции следует применять `mb_strtoupper()`, как указано в следующем примере:

```
<?php
$str = "русский текст в кодировке utf-8";
$str = mb_strtoupper($str, "utf8");
echo $str;
?>
```

Причем использование второго параметра обязательно для корректной работы, хотя это и не следует из документации, выложенной на сайте **www.php.net**.

Приставка `mb` в имени этой функции означает *multi-byte*, т. е. многобайтную кодировку. Если вы помните, при установке модуля PHP мы указали необходимость подключения этого расширения в устанавливаемый модуль.

Функция `strtoupper()` не является единственным исключением, неудобства при работе с подобными функциями являются существенным недостатком современной версии языка PHP. Его обещают исправить в версии 6, так что нам остается только ждать выхода этого обновления, а до тех пор при возникновении проблем сверяться с документацией.

Объединение и разделение строк с помощью строковых функций

Функция `explode()` принимает в качестве первого аргумента строку и разделяет ее на части по разделительной строке, указанной во втором аргументе. Части строки возвращаются в виде массива, а сам разделитель вырезается и в массив не попадает.

Если передан третий аргумент *`limit`*, массив будет содержать максимум *`limit`* элементов, при этом последний элемент будет содержать весь остаток строки.

```
<?php
$stroka = "очень длинная строка просто очень длинная, конца ей не видно, какая
длинная строка";
// Подстрока является разделителем — местом разреза строки
$podstroka = "длинная";
// limit-1 — столько раз будет выполнен поиск разделителя
$limit = 3;
// Если лимит не задан, то учитываются все элементы.
// Создание массива $a, состоящего из частей строки
$a = explode($podstroka, $stroka, $limit);
print_r($a);
?>
```

В приведенном примере функция `print_r()` используется для распечатки структуры массива, она выводит на экран все ключи и значения элементов массива.

Эффекта, противоположного действию функции `explode()`, можно добиться, используя функции `implode()`. Эта функция принимает элементы из массива и объединяет их со строкой, переданной в первом параметре. Вызов этой функции во многом подобен вызову функции `explode()`, но ее действие противоположно.

```
<?php
$comp = array ("processor"=>"PIV-1600", "mboard"=>"ABIT",
               "RAM"=>"512M", "video"=>"Geforce",
               "hdd"=>"Quantum20Gb");
// Все значения элементов массива записываем
// в одну строку через тире
$str = implode(" — ", $comp);
echo $str;
?>
```

Поиск и замена подстрок

Использование функции *substr()*

Функция `substr()` ищет и возвращает подстроку, начиная с позиции, указанной во втором аргументе, длина выбираемой подстроки определяется третьи аргументом:

```
string substr(string string, int start[, int length])
```

`substr()` возвращает подстроку строки *`string`* длиной *`length`*, начинающегося с *`start`* символа по счету.

Если параметр *start* неотрицателен, возвращаемая подстрока начинается в позиции *start* от начала строки, считая от нуля. Например, в строке "abcdef" в позиции 0 находится символ "a", в позиции 1 — символ "b" и т. д.

Определение позиции подстроки: функция *strpos()*

Функция strpos() имеет следующий прототип:

```
int strpos(string substring, string string[, int offset])
```

Возвращает позицию первого вхождения подстроки *substring* в строку *string*, ищет всю строку *substring*, а не только первый символ. Необязательный аргумент *offset* позволяет указать, с какого по счету символа строки *string* начинать поиск.

Отыщем слово "иголка", причем начнем поиск с начала строки:

```
<?php
$stroka = "Стог сена, а в нем иголка";
echo $pos = strpos($stroka, "иголка", 0);
?>
```

На экране появится номер позиции — 19. Если же в качестве третьего аргумента укажем 20, то подстрока найдена не будет и функция вернет значение false.

Замена подстрок: *str_replace()*, *substr_replace()*, *strstr()*

Поиск части строки и замена найденной подстроки на другую часто используются при обработке строковых значений.

Для замены строки можно применить функцию:

```
str_replace(search, replace, subject)
```

Эта функция возвращает строку или массив *subject*, в котором все вхождения *search* заменены строкой *replace*. Если не нужны сложные правила поиска и замены, использование этой функции предпочтительнее вызова функций, оперирующих регулярными выражениями. Например:

```
<?php
$a = "ищем эту строку";
$b = "меняем на эту строку";
$d = array("ищем эту строку", "ab", "ищем эту строку", "ab");
$d = str_replace($a, $b, $d);
foreach ($d as $value)
  echo $value."<br/>";
?>
```

В результате выполнения этого сценария в браузер будет выведено:

```
меняем на эту строку
ab
меняем на эту строку
ab
```

Функция substr_replace() используется для поиска и замены конкретной подстроки в строке. Ее прототип:

```
string substr_replace(string string, string replacement,
                      int start[, int length])
```

Функция `substr_replace()` заменяет часть строки *`string`*, начинающуюся с символа с порядковым номером *`start`* и длиной *`length`*, строкой *`replacement`* и возвращает результат.

Если *`start`* — положительное число, замена начинается с символа с порядковым номером *`start`*. Если *`start`* — отрицательное число, замена начинается с символа с порядковым номером *`start`*, считая от конца строки.

Если аргумент *`length`* — положительное число, то он определяет длину заменяемой подстроки. Если этот аргумент отрицательный, он определяет количество символов от конца строки, на котором заканчивается замена. Этот аргумент необязателен и по умолчанию равен `strlen(`*`string`*`);`, т. е. замена до конца строки *`string`*.

В следующем примере происходит замена строки "Двадцать пять" на строку "25":

```
<?php
$var = "Двадцать пять";
echo "Оригинал: $var<hr />\n";
echo substr_replace($var, "25", 0)."<br />\n";
?>
```

Теперь заменим слово "букв", первый символ которого находится на 7-й позиции, на слово "цифр":

```
<?php
$var = "Строка букв";
echo "Оригинал: $var<hr />\n";
echo substr_replace($var, "цифр", 7)."<br />\n";
?>
```

В следующем сценарии из строки будут вырезаны символы "29":

```
<?php
$var = "Строка состоит из 29 символов";
echo substr_replace($var, " ", 18, 2)."<br />\n";
?>
```

Функция `strstr()` находит первое вхождение подстроки:

```
string strstr(string string, string substring)
```

Возвращает подстроку строки *`string`*, начиная с первого вхождения *`substring`* до конца строки. Если подстрока *`substring`* не найдена, возвращает `false`. Если *`substring`* не является строкой, он приводится к целому и трактуется как код символа.

В следующем примере выводится часть строки, начиная с первого пробела.

```
<?php
$stroka = "Текст с пробелами";
$res = strstr($stroka, " ");
echo $res;
?>
```

Регулярные выражения

При проверке значений строк зачастую требуется не установить точное совпадение с искомым значением, а найти соответствие некоторому шаблону, приблизительно характеризующему строку. Так, при описании человека говорят, что он высокого роста. Эта характеристика не точна, но во многих случаях позволяет исключить наверняка неподходящих людей. Такого рода шаблоны в PHP составляют с помощью регулярных выражений.

Регулярные выражения предоставляют мощный механизм для операций со строками. В его основе лежит извлечение подстроки из строки или замена подстроки на основании поиска по шаблону.

Язык регулярных выражений — это язык составления шаблонов. В PHP поддерживаются два стандарта шаблонов — POSIX-совместимые регулярные выражения и Perl-совместимые регулярные выражения (PCRE). Функции для работы с POSIX-совместимыми регулярными выражениями несколько медленнее Perl-совместимых, они также менее гибки, но синтаксис немного проще.

Рассмотрим язык PCRE. Шаблон регулярного выражения — это строка, состоящая из простого текста и метасимволов, которые позволяют находить соответствие нескольким символам одновременно. Базовые классы символов даны в табл. 9.1.

Таблица 9.1. *Базовые классы символов PCRE*

Метасимвол	Соответствие символам
`\d`	Соответствует цифре
`\D`	Соответствует нецифровому символу
`\s`	Соответствует пробельному символу (пробел, символ табуляции, символ новой строки)
`\S`	Соответствует любому непробельному символу
`\w`	Соответствует латинской или русской букве, цифре или подчеркиванию
`\W`	Соответствует любому символу, кроме латинской или русской буквы, цифры или подчеркивания
`.`	Любой символ, кроме новой строки

Примечание

Обратите внимание на то, что PHP, в отличие от JavaScript, рассматривает русские буквы как соответствующие шаблону `\w`.

Рассмотрим на примерах, как использовать эти шаблоны.

Функция `preg_match()` выполняет проверку на соответствие регулярному выражению. Иначе говоря, эта функция ищет в заданном тексте совпадения с шаблоном. Первый аргумент функции `preg_match()` содержит шаблон регулярного выражения, а второй — анализируемую строку. Функция возвращает 1, если найдет соответствие шаблону в строке хотя бы один раз, и 0, если не найдет.

Шаблон записывается как переменная строкового типа, а значение шаблона помещается в разделители — прямые слэши. В следующем примере функция `preg_match()` будет искать любую цифру в строке `$subject`.

```
<?php
$subject = "a4bcder4f";
$pattern = '/\d/';
echo preg_match($pattern, $subject);
?>
```

Базовые классы метасимволов предназначены для поиска одного-единственного символа в строке. Для какого-либо полезного их применения язык регулярных выражений надо дополнить обозначениями того, сколько раз метасимволы могут встречаться в строке. Операторы счетчиков символов приведены в табл. 9.2.

Таблица 9.2. *Счетчики PCRE*

Оператор	Описание
*	Соответствует повторению символа ноль или более раз
+	Соответствует повторению символа один или более раз
?	Соответствует повторению предыдущего символа ноль или один раз
{*n*}	*n* — неотрицательное число. Соответствует ровно *n* вхождениям символа
{*n*,}	*n* — неотрицательное число. Соответствует *n* или более вхождениям символа
{,*m*}	*m* — неотрицательное число. Соответствует *m* и менее вхождениям символа
{*n*,*m*}	*n* и *m* — неотрицательные числа. Соответствует не менее чем *n* и не более чем *m* вхождениям символа

В следующем примере будем искать в строке пять цифр, за которыми следует дефис, а затем еще четыре цифры. При этом символы, не входящие в число метасимволов, например дефис, рассматриваются просто как текстовый символ. Поставьте в строке `$subject` между пятеркой и черточкой пробел, и функция не найдет совпадения с шаблоном:

```
<?php
$subject = "12345-1234";
$pattern = '/\d{5}-\d{4}/';
echo preg_match($pattern, $subject);
?>
```

Функция `preg_match()` ищет совпадение с шаблоном по всему тексту. Если же вам надо, чтобы шаблон находился именно в начале строки, то надо добавить символ привязки к начальной позиции ^:

```
<?php
$subject = "12212345-1234";
$pattern = '/^\d{5}/';
echo preg_match($pattern, $subject);
?>
```

На конец строки указывает метасимвол `$`. Таким образом, если вы хотите найти строку, состоящую исключительно из символов, соответствующих шаблону, можно написать так:

```
<?php
$subject = "12212345-1234";
$pattern = '/^\d{5}$/';
echo preg_match($pattern, $subject);
?>
```

Можно создавать собственные классы символов, помещая символы в квадратные скобки. Допускается указывать диапазоны значений. Например, чтобы создать класс для обозначения шаблона соответствия одной из цифр от 2 до 8, можно использовать `[2-8]`. Так, номер городского телефона в Санкт-Петербурге соответствует шаблону:

```
/[2-8]{1}[0-9]{2}-[0-9]{2}-[0-9]{2}/
```

То есть номера телефонных станций (первые три цифры) в городе начинаются с двойки и не могут начинаться с девятки, зарезервированной для мобильных телефонов.

Символ вертикальной черты `|` используется для разделения альтернативных масок. Например, шаблон `"город|поселок"` означает, что мы ищем любое слово: "город" или "поселок". Допустимо указывать любое количество альтернатив. В процессе поиска соответствия просматриваются все перечисленные альтернативы слева направо до первого найденного соответствия.

Шаблоны могут работать по-разному в зависимости от модификаторов, которые добавляются после закрывающего шаблон разделителя. Список общеупотребительных модификаторов шаблонов приведен в табл. 9.3.

Таблица 9.3. *Модификаторы шаблонов PCRE*

Модификатор	**Значение**
`i` (ignore case)	Не различать строчные и прописные буквы
`m` (multiline)	Многострочный поиск
`s`	Разрешить использование символа "`.`" для обозначения новой строки
`x`	Разрешить комментарии и пробельные символы в регулярных выражениях
`u`	Трактовать данные как текст в кодировке UTF-8

Например, без модификатора `i` функция не найдет в следующем примере соответствия шаблону:

```
<?php
$subject = "ПРИМЕР СТРОКИ, СОСТОЯЩЕЙ ИЗ ЗАГЛАВНЫХ БУКВ";
$pattern = '/[а-я]/ui';
echo preg_match($pattern, $subject);
?>
```

Скорее всего, вы захотите не только узнать, есть ли в строке подстрока, соответствующая шаблону, но и получить эту подстроку. Вдобавок может возникнуть желание выявить не только всю эту подстроку, но и ее фрагменты. Для того чтобы выделить внутри шаблона интересующие нас фрагменты, нужно заключить эти фрагменты в круглые скобки. Например, чтобы из номера телефона выделить номер телефонной станции (первые три цифры) и остальную часть номера (четыре последние цифры), нужно сгруппировать подшаблоны с помощью круглых скобок так:

```
/([2-8]{1}[0-9]{2})-([0-9]{2}-[0-9]{2})/
```

После того как определены подшаблоны, можно узнать, чему они в тексте строки соответствуют. Для этого надо задать в качестве третьего параметра функции `preg_match()` имя массива. Функция создаст этот массив и заполнит его фрагментами строки, соответствующими подшаблонам. Номер элемента массива определяется по положению подшаблона: в шаблоне перебираются подшаблоны слева направо и нумеруются открывающие их скобки. Например, выделим номер телефонной станции:

```
<?php
$subject = "Мой телефон — 211-22-33 ";
$pattern = '/([2-8]{1}[0-9]{2})-([0-9]{2}-[0-9]{2})/';
preg_match($pattern, $subject, $matches);
echo $matches[1];
?>
```

Первый подшаблон содержит интересующие нас сведения. Обратите внимание, что нулевой элемент массива `$matches` содержит всю подстроку, соответствующую шаблону, а фрагменты строк, соответствующих подшаблонам, попали в элементы с номерами, под которыми идут подшаблоны.

Функция `preg_match()` ищет только первое соответствие шаблону. Если требуется провести поиск по всему тексту, то надо использовать `preg_match_all()`. Функция `preg_match_all()` выполняет глобальный поиск шаблона в строке. Она ищет в строке *`subject`* все совпадения с шаблоном *`pattern`* и помещает результат в массив *`matches`* в порядке, определяемом комбинацией флагов *`flags`*. После нахождения первого соответствия последующий поиск будет осуществляться не с начала строки, а от конца последнего найденного вхождения.

Регулярные выражения также позволяют выполнить замену фрагментов строк. Можно использовать функцию `preg_replace()` для замены подстроки, соответствующей шаблону, на другую подстроку. Второй аргумент этой функции может быть либо простой строкой, либо содержать ссылки на подшаблоны в виде `\`*`n`*, где *`n`* — номер подшаблона. Третий аргумент функции — строка, в которой ищется соответствие шаблону. Например:

```
<?php
$subject = "Здесь всего четыре слова";
$pattern = "/(всего)\s\w+\s(слова)/u";
$new_subj = preg_replace($pattern, '\1 три \2', $subject);
echo $new_subj;
?>
```

Шаблон в этом примере составлен так, что сначала в тексте ищется слово "всего" — первый подшаблон (1), потом после пробела могут идти одна или несколько букв или цифр, а затем "слова" — второй подшаблон (2). Эти подшаблоны упоминаются во втором аргументе в виде цифр, перед которым стоит обратный слэш. Функция находит соответствие шаблону в тексте и заменяет найденный фрагмент на значение второго аргумента, подставив в него найденные подшаблоны.

Обратите внимание на то, что в приведенных примерах используется модификатор `u`, что необходимо, когда мы работаем с русскими строками в кодировке UTF-8.

В следующем примере производится перестановка слов:

```
<?php
$pattern = "/(\w+)\s(\w+)\s(\w+)/u";
$subject = "Александр Иванович Привалов";
$matches = preg_replace($pattern, '\3, \1 \2', $subject);
echo $matches;
?>
```

При работе с регулярными выражениями надо быть очень внимательными — лишний пробел, добавленный и незамеченный, может нарушить работу сценария.

В следующем примере мы не просто ищем трехзначное число, но и получаем его среднюю цифру:

```
<?php
$str = "123 234 345 456 567";
$result = preg_match('/\d(\d)\d/', $str, $found);
echo "Matches: $result<br>";
print_r($found);
?>
```

Глава 10

Графика в PHP 5

Графическая библиотека PHP предназначена для генерации изображений в форматах GIF, PNG, JPEG и др. Рассмотрим кратко особенности этих форматов графических данных.

Графические форматы данных

JPEG

JPEG (Joint Picture Experts Group, объединенная группа экспертов по изображениям) — метод сжатия изображения с потерями, т. е. при уменьшении размера файла теряется качество. Формат JPEG обычно используется для хранения фотографических изображений с большим количеством цветов.

При сохранении JPEG-файла можно указать степень сжатия, которую обычно задают в некоторых условных единицах, например от 1 до 10. Большее число соответствует лучшему качеству, но при этом увеличивается размер файла.

GIF

GIF (Graphics Interchange Format) — формат обмена графическими изображениями. Формат GIF способен хранить сжатые без потерь изображения в формате до 256 цветов в одном файле и предназначен, в основном, для чертежей, графиков и т. д.

Формат GIF поддерживает анимационные изображения. Фрагменты представляют собой последовательности нескольких статичных кадров, а также информацию о том, сколько времени каждый кадр будет показан на экране. Анимация может быть закольцована, т. е. после последнего кадра будет вновь показан первый.

Один из цветов в палитре может быть объявлен прозрачным. В этом случае в программах, которые поддерживают прозрачность GIF, сквозь пикселы, окрашенные прозрачным цветом, будет виден фон.

Формат GIF в течение некоторого времени был запатентован, однако срок его патентной защиты истек. Из-за патентных осложнений поддержка GIF была исключена из первой версии графической библиотеки PHP, но восстановлена во второй.

PNG

Формат PNG (Portable Network Graphics, переносимая сетевая графика) хранит графическую информацию в сжатом виде, причем это сжатие производится без потерь.

Формат PNG спроектирован для замены формата GIF и позиционируется, прежде всего, для использования в Интернете и для редактирования графики.

Формат разрешает использование неограниченного количества цветов в изображении, полупрозрачные изображения и др.

Точнее, в PNG количество цветов все же ограничено, но в отличие от GIF, оно ограничено 2^{48} цветами. PNG использует открытый, незапатентованный алгоритм сжатия.

Существует одна особенность GIF, которую PNG не пытается воспроизвести — это поддержка множественного изображения, особенно мультипликации.

Подключение графической библиотеки

Для работы с графической библиотекой второй версии надо подключить необходимые расширения. В первую очередь, это библиотека gd2. Возможно, вы также захотите воспользоваться функциями из библиотеки exif, которые позволяют прочитать метаданные из файлов фотографий, сделанных цифровыми камерами. При работе с графикой может потребоваться еще и библиотека mbstring, которая предоставляет специальные функции для работы со строками в многобайтных кодировках. Проверьте, что эти библиотеки установлены. Они должны быть указаны в файле php.ini:

```
extension=php_mbstring.dll
extension=php_exif.dll
extension=php_gd2.dll
```

Посмотреть, поддерживается ли графическая библиотека, можно, вызвав функцию `phpinfo()` и найдя в ее выводе раздел `gd` (табл. 10.1).

Таблица 10.1. *Фрагмент данных о настройках PHPс помощью функции* `phpinfo()`

Переменная	Значение
GD Support	enabled
GD Version	bundled (2.0.28 compatible)
GIF Create Support	enabled
JPG Support	enabled
PNG Support	enabled

Создание изображений

При создании изображений надо выполнить четыре основных шага:

1. Создание холста.
2. Рисование линий, многоугольников и т. п. или вывод текста.

3. Вывод рисунка в браузер.
4. Освобождение ресурсов.

Холст для изображения можно сделать заново или воспользоваться уже имеющимся рисунком, чтобы поверх него нарисовать что-либо или просто вывести в браузер готовое изображение.

Функция `imagecreatetruecolor()` создает холст для изображения, цвет каждой точки которого определяется в координатах цветовой модели RGB (True Color), т. е. яркость каждого из трех цветов записывается двумя шестнадцатеричными числами. Функция `imagecreatetruecolor()` принимает два обязательных параметра — ширину и высоту рисунка и возвращает идентификатор ресурса. При неудаче функция возвращает `false`. Например:

```
$im = imagecreatetruecolor($width, $height);
```

Идентификатор ресурса `$im` указывает на изображение и используется для идентификации этого изображения. Переменные `$width` и `$height` определяют длину и высоту создаваемого холста. Сам холст рисуется черным.

Для использования готовых рисунков различных форматов существует набор функций, принимающих в качестве аргумента имя файла, создающих холст и возвращающих идентификатор изображения: `imagecreatefrompng()`, `imagecreatefromjpeg()` и `imagecreatefromgif()`.

Существует несколько функций рисования линий, прямоугольников, дуг и других фигур. Параметрами этих функций являются координаты начальных и конечных точек линий, цвет, размер шрифта и др.

> ***Примечание***
>
> Положение любой точки в изображении отсчитывается вниз и вправо от левого верхнего угла, координатами которого являются `x=0`, `y=0`. Правый нижний угол изображения имеет координаты `x=$width`, `y=$height`.

Сначала зальем черный холст каким-нибудь "веселеньким" цветом:

```
imagefill($im, 0, 0, 0xCCFFCC);
```

Параметрами функции `imagefill()` являются идентификатор изображения, координаты `x` и `y`, с которых начинается заливка, и шестнадцатеричное число, обозначающее номер цвета.

Теперь нарисуем прямоугольник:

```
imagefilledrectangle($im, 10, 60, 90, 120, 0x00FFFF);
```

Со второго по пятый параметры функции `imagefilledrectangle()` указывают на левую верхнюю и правую нижнюю вершины прямоугольника.

Готовое изображение можно вывести напрямую в браузер или же в файл. Вывод в браузер состоит из двух шагов. Вначале необходимо сообщить браузеру, что будет выводиться именно изображение, а не текст или HTML-код. Это достигается вызовом функции `header()`, указывающей MIME-тип передаваемого изображения:

```
header("Content-type: image/png");
```

Эта функция пересылает строки HTTP-заголовков. Важно отметить, что функция `header()` не может быть выполнена, если HTTP-заголовок уже был отправлен. PHP

посылает HTTP-заголовки автоматически всякий раз, когда происходит вывод чего-либо в браузер. Следовательно, наличие любого оператора `echo` или даже просто пробела перед дескриптором PHP приводит к отправке заголовка, а значит, и к выводу предупреждающего сообщения PHP при попытке вызова функции `header()`.

После отправки заголовка изображение выводится в браузер в результате вызова функции:

```
imagepng($im);
```

После того как работа с изображением завершена, необходимо очистить память на сервере, занятую только что созданным рисунком. Для этого следует вызвать функцию:

```
imagedestroy($im);
```

Поскольку заголовок может быть переслан лишь один раз, и это единственный способ сообщить браузеру, что передается изображение, вставлять изображения, создаваемые в сценарии, в обычные страницы не просто. Сценарий создания простого рисунка, содержащего прямоугольник, представлен в листинге 10.1.

Листинг 10.1. Создание рисунка в формате PNG

```
<?php
$width = 200;
$height = 200;
$im = imagecreatetruecolor($width, $height);
imagefill($im, 0, 0, 0xCCFFCC);
imagefilledrectangle($im, 10, 60, 90, 120, 0x00FFFF);
header("Content-type: image/png");
imagepng($im);
imagedestroy($im);
?>
```

На рисунке можно сделать надпись, используя функцию:

```
imagestring($image, 5, 0, 0, "Hello, world!", 0x000000);
```

Эта функция использует встроенный шрифт, относительный размер которого передается во втором параметре и может принимать значения от 1 до 5. Следующие два параметра — координаты начала строки.

Попытка писать по-русски с помощью этой функции приводит к появлению символов в западноевропейской, а не в кириллической кодировке. Попробуем использовать функцию `imagettftext()`, которая работает с системным шрифтом. Эта функция принимает в качестве аргументов идентификатор изображения, размер изображения, угол наклона текста, координаты начала строки `x` и `y`, цвет, имя шрифта и выводимый текст. Скопируем из каталога системных шрифтов нашей русифицированной ОС Windows (C:\Windows\Fonts) какой-нибудь файл шрифта, например, arial.ttf, в каталог, где мы храним сценарии, и напишем:

```
$text = "Отлично!";
imagettftext($im, 20, 0, 25, 25, 0xF00FFF, 'arial.ttf', $text);
```

Если этот сценарий создается, например, в кодировке windows-1251, то при использовании русифицированного шрифта проблема все равно не решается. Дело в том, что внутреннее представление текста в PHP идет в кодировке UTF-8, т. е. для корректного вывода текста надо его сначала преобразовать с помощью функции `iconv()`:

```
$text = iconv('windows-1251', 'UTF-8', 'Отлично! ');
imagettftext($im, 20, 0, 25, 25, 0xF00FFF, 'arial.ttf', $text);
```

Можно выводить в браузер рисунок, как только что рассматривалось в примере, а можно поместить имя сценария в атрибут `src` тега `<img>`, отрисовывающего изображение:

```
<img src="image.php" alt="Результат теста">
```

Теперь научимся передавать в сценарий создания рисунка координаты прямоугольника в виде параметров. Допустим, что прямоугольник на рисунке показывает результаты компьютерного теста. При 100% правильном ответе он должен занимать всю выделенную ему высоту, а при 50% правильных ответах — только половину и т. д. Пусть переменная `$grade` содержит результат теста. Этот результат можно передать методом `GET` в сценарий, указав его в качестве параметра:

```
src = "im1.php?grade=1"
```

Но значение переменной `$grade` получено в результате работы предыдущих сценариев, заранее он неизвестен, тогда вызываем скрипт im1.php так:

```
<img src="im1.php?grade=<?php echo $grade ?>" alt="Результат теста">
```

Сам сценарий создания рисунка получит результат теста из массива `$_GET` (листинг 10.2).

Листинг 10.2. Передача параметров рисунка

```
<?php
$im = imagecreatetruecolor(450, 200);
$grade = $_GET["grade"];
$grade = $grade*100;

imagefill($im, 0, 0, 0xffffcc);
$text = iconv("windows-1251", "UTF-8", "Результат — ".
              $grade."% верных ответов");
imagettftext($im, 15, 0, 100, 25, 0x000000, 'arial.ttf', $text);
imagefilledrectangle($im, 10, 0, 90, 200, 0x33CC66);
imagefilledrectangle($im, 10, round(2*(100-$grade)), 90, 200, 0x66FF33);

header('Content-type: image/png');
imagepng($im);
imagedestroy($im);
?>
```

Для определения размеров изображения можно применить функцию `getimagesize()`, которая возвращает массив, содержащий элементы, перечисленные в табл. 10.2.

Таблица 10.2. *Элементы массива, возвращаемого функцией* `getimagesize()`

Ключ элемента	Описание
0	Ширина в пикселах
1	Высота в пикселах
2	Тип изображения
3	Строка типа `height=12 width=22` для тега `<img>`
bits	Бит в выборке для формата JPEG
channels	Выборки в пикселе для формата JPEG
mime	MIME-тип

В результате получим листинг 10.3.

Листинг 10.3. Определения размеров изображения

```
<body>
  <?php
    $grade = 0.5;
    $s = getimagesize("vienna.jpg");
  ?>
  <img src="vienna.jpg" <?php echo $s[3]?> alt="Vienna">
</body>
```

Часто при создании фотогалереи требуется использовать уменьшенную копию изображения. При этом нужно уменьшить изображение на сервере и отправить клиенту уже файл рисунка меньшего размера, а не просто менять значения атрибутов `width` и `height` тега `<img>`. Во второй версии библиотеки, с появлением поддержки True Color, введена новая функция `imagecopyresampled()`, копирующая все или часть изображения и уменьшающая эту копию. При этом уменьшение производится со сглаживанием, что позволяет добиться неплохого качества уменьшенного изображения (листинг 10.4).

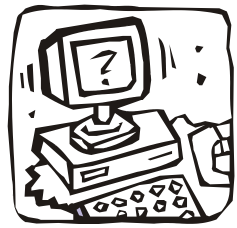
Листинг 10.4. Изменение размеров изображения

```
<?php
$filename = 'test.jpg';
$percent = 0.2;
```

```
header('Content-type: image/jpeg');

list($width, $height) = getimagesize($filename);
$new_width = $width * $percent;
$new_height = $height * $percent;

$image_p = imagecreatetruecolor($new_width, $new_height);
$image = imagecreatefromjpeg($filename);
imagecopyresampled($image_p, $image, 0, 0, 0, 0, $new_width,
                   $new_height, $width, $height);
imagejpeg($image_p, null, 100);
?>
```



Глава 11

Cookies и управление сессиями

Для разработки сайта, состоящего из многих страниц, необходимо научиться передавать данные из одной страницы в другую. Например, часто требуется передать сведения о посетителе, полученные в результате его регистрации на первой странице сайта. При этом обнаруживается проблема: протокол HTTP не сохраняет состояния. Это означает, что HTTP не имеет средств передачи какой-либо информации между страницами. Если пользователь запрашивает одну за другой две страницы, HTTP не обеспечивает возможности уведомить, что оба запроса исходят от одного и того же пользователя.

Если пользователь побывал на сайте, то надо уметь передать в следующую Web-страницу информацию о его действиях на предыдущих страницах. Для этой цели можно использовать cookie.

Cookie

Cookie — это небольшой фрагмент информации, который серверные сценарии сохраняют на клиентской машине. Операции с ним можно описать в такой последовательности:

1. Клиент отправляет HTTP-запрос серверу.
2. Сервер посылает HTTP-ответ, среди прочего включающий в себя заголовок `Set-Cookie:`*`var=value`*.
3. Клиент переходит на другую страницу того же сервера путем отправки нового запроса, включающего в себя заголовок `Cookie:`*`var=value`*.
4. Сервер отвечает клиенту.

Дополнительная информация может добавляться в cookie в виде необязательных атрибутов:

- `domain` — определяет, в каком домене будут действовать cookies. По умолчанию это домен запрошенного ресурса;
- `expires` — дата и время, после которого cookie станет считаться недействительным и будет удален. По умолчанию cookie хранится только в памяти, и его срок истекает при закрытии браузера. В качестве времени жизни cookie может использоваться:
  - дата, соответствующая давно прошедшему моменту времени. Тогда cookie окажется недействительным уже при первом предъявлении;

- ячейка памяти, выделенная под cookie, вся заполняется единицами — это указывает на бесконечно далекое время в будущем. Такие cookies останутся действительными всегда;
- текущий момент времени плюс сколько-то секунд. Вот столько секунд и будут действовать определенные таким образом cookies;

- `path` — указывает путь до ресурса, в пределах которого действует cookie. По умолчанию ограничений нет;
- `secure` — атрибут без значения, наличие которого указывает на то, что cookie может быть отправлен только по защищенному соединению (SSL).

Пример cookie, отправленного сервером:

```
Set-cookie:name=smth; domain=our.org; expires=Mon, 21 Aug 2006 12:01:20 GMT;
path=/; secure
```

Заголовок cookie, который будет передаваться в следующем запросе:

```
Cookie:name=smth
```

Атрибуты, включаемые в заголовок `Set-cookie`, используются только для определения того, будет ли cookie включен в следующий запрос. Если и будет, то в запросе появятся только имя и значение (`name=smth`).

В PHP значение cookie можно получить из суперглобального массива `$_COOKIE`.

В следующем примере происходит попытка отправить клиенту cookie и получить его обратно — этим мы проверяем, установлена ли поддержка cookie у клиента (листинг 11.1).

Листинг 11.1. Проверка поддержки cookie у клиента

```
<?php
$cookie = $_COOKIE["test"];
if (!$cookie)
{
  setcookie("test", "1", 0x7FFFFFFF);
}
else
{
  @$test = $_COOKIE["test"];
  if (!$test)
  {
    echo "Хорошо бы включить cookies";
  }
  else
  {
     echo "Работает!";
  }
}
?>
```

Каждый браузер хранит cookie в своем специально выделенном для cookie каталоге. Если вы работаете с Internet Explorer, откройте каталог Documents and Settings на системном диске, найдите там каталог с именем вашей учетной записи (то имя, под которым вы вошли в систему), а в нем каталог Cookies. В нем вы увидите тот cookie, который только что записали с помощью вышеприведенного примера. Пользователи Mozilla могут просмотреть cookie с помощью Cookie Manager (меню **Edit | Preferences | Privacy and Security | Cookies**).

Счетчик посещений

Счетчики посещений — весьма популярная категория скриптов. Основная их идея состоит в том, что при каждом запуске сценария происходит запись некоторой информации в файл или базу данных. Можно просто каждый раз добавлять единицу, а можно читать заголовки запроса клиента и писать их в базу. Конечно, лучший источник информации о посетителях сайта находится в журнальном файле Apache access.log в каталоге logs, но Web-мастер не всегда имеет доступ к журнальной статистике.

Функция `setcookie()` создает HTTP-заголовок, содержащий cookie. Ее параметрами являются имя cookie, его значение и время жизни. Время жизни cookie записывается в виде шестнадцатеричного числа.

В следующем примере (листинг 11.2) cookies используются для того, чтобы подсчитать, сколько раз пользователь заходил на наш сайт.

Листинг 11.2. Счетчик посещений с использованием cookie

```
<?php
@$counter = $_COOKIE["counter"];
if (!isSet($counter))
{
  $counter = 0;
}
$counter++;
setcookie("counter", $counter, 0x7FFFFFF);
?>
<html>
<head><title>Счетчик посещений</title></head>
<body>
   <? if ($counter == 1)
     { ?>
       <h1> Добро пожаловать! </h1>
   <?}
     else
     { ?>
        <h1> Вы зашли на эту страницу в <? echo $counter ?> раз! </h1>
   <? } ?>
</body>
</html>
```

Если же надо идентифицировать клиента, можно в качестве значения cookie отправить случайное число, сгенерированное функцией `uniqid()`. Вызванная без параметров, она возвращает строку с уникальным идентификатором:

```
<?php
$id = uniqid("");
SetCookie("id", $id, 0x7FFFFFFF);
?>
```

Можно задать время жизни cookie, например, используя функцию `time()`, которая возвращает количество секунд, прошедших с 1 января 1970 г., 00:00:00 до текущего времени. Например, пусть время жизни cookie будет 10 минут:

```
setcookie("id", $id, time()+600);
```

Cookie предоставляет весьма полезный механизм учета состояния, но пользователь может отключить возможность принимать cookie, и тогда сервер не сможет получить обратно посланные клиенту ранее данные.

Хорошо бы держать все необходимые данные в каком-то хранилище, доступ к которому разрешен только сценарию, знающему имя хранилища.

Сессии

Идея управления сессиями заключается в отслеживании перехода пользователя с одной страницы сайта на другую в течение одного сеанса связи с Web-сервером. Если это удастся осуществить, мы сможем предоставлять пользователю содержимое сайта в соответствии с его уровнем прав доступа или персональными настройками. *Сессия* представляет собой группу переменных, которые, в отличие от обычных переменных, сохраняются и после завершения выполнения PHP-сценария. Сессия идентифицируется номером, который передается сценарию в виде cookie или в виде параметра в адресной строке.

При работе с сессиями различают следующие этапы:

- открытие сессии;
- регистрация переменных сессии и их использование;
- закрытие сессии.

Открыть сессию можно с помощью функции `session_start()`, которая вызывается в начале PHP-сценария. Эта функция проверяет, существует ли идентификатор сессии, и если нет, то создает его. Функция отправляет клиенту заголовок с номером сессии, поэтому до вызова этой функции из сценария в браузер не должно передаваться ничего, даже тег HTML или пробел. Иначе вы получите сообщение о попытке повторной отсылки заголовков, и сценарий завершит работу.

Напишем несколько маленьких скриптов, демонстрирующих на примере сценария компьютерного теста, сдаваемого студентом, использование сессий.

Сначала попросим студента зарегистрироваться (листинг 11.3).

Листинг 11.3. Регистрация студента для сдачи теста

```
<html>
<head><title>Регистрация</title></head>
<body>
  <form method="GET" action="sess1.php">
    Введите имя:
    <input type="text" name="name" value="">
    <input type="submit" value="Готово!">
  </form>
</body>
</html>
```

Теперь в сценарии sess1.php создадим сессию с помощью функции `session_start()`. В сессии хранятся переменные — похоже на хранилище, куда можно положить вещь (переменную), можно забрать ее оттуда, поменять на другую, а можно выбросить вон — уничтожить переменную сессии.

Прочитаем имя студента из массива `$_GET` и запишем это имя в сессию. Для этого в нашем распоряжении имеется автоглобальный массив `$_SESSION` (листинг 11.4).

Листинг 11.4. Сценарий sess1.php — создание сессии

```
<?php
session_start();
$new = $_GET["name"];
$_SESSION['new'] = $new;
echo "Здравствуйте, ".$_SESSION['new'];
?>
<br/>Щелкните по ссылке, когда будете готовы начать тест: <br/>
<a href = "sess2.php">Начать тест!</a>
```

Щелкая по ссылке, студент переходит на сценарий sess2.php, работая с которым он отвечает на вопросы теста. В тесте много вопросов, идет запись ответов на эти вопросы. Это мы опустим, наша цель — на любой странице теста прочитать переменную из сессии (листинг 11.5) и, таким образом, узнать имя студента.

Листинг 11.5. Сценарий sess2.php — чтение переменной сессии

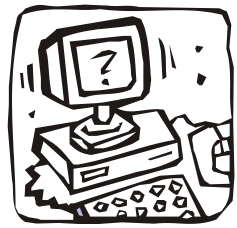
```
<?php
session_start();
$var = $_SESSION['new'];
echo $var.", ответьте, пожалуйста, на вопрос.<br>";
?>
<a href = "sess3.php">Завершить тест</a>
```

После ответов на тест надо завершить сессию (листинг 11.6). Перед закрытием сессии удалим переменные сессии.

Листинг 11.6. Сценарий sess3.php — закрытие сессии

```
<?php
session_start();
$sess = $_SESSION["new"];
echo $sess.", тест пройден!<br/>";
unset($_SESSION['new']);
echo "Ваш тест закончен.";
session_destroy();
?>
```

Функция `unset()` удаляет переменную из сессии, а функция `session_destroy()` закрывает сессию. Все, теперь сессия недоступна, все ее переменные потеряны.



Глава 12

Загрузка файлов на сервер

Представьте себе, что мы создаем сценарий фотоальбома с фотографиями, загруженными на наш сервер пользователями. Поддержка загрузки файлов на сервер по протоколу HTTP — одна из важнейших возможностей PHP. При этом пересылка файлов происходит с браузера клиента на сервер. Применим HTML-формы для передачи данных в этом направлении. Форма для загрузки файлов представлена в листинге 12.1.

Листинг 12.1. Форма для загрузки файлов на сервер

```
<h2>Загрузка файлов на сервер</h2>
<form enctype="multipart/form-data" action="upload.php" method="post">
  <input type="hidden" name="MAX_FILE_SIZE" value="1000">
  Загрузить этот файл:
  <input name="userfile" type="file">
  <input type="submit" value="Отправить файл">
</form>
```

Файл upload.php — это сценарий, в который передаются данные из формы. Атрибут `enctype` определяет MIME-тип закачиваемых данных. Тип `multipart` определен в стандарте как совокупность данных различных типов.

Скрытое поле `name="MAX_FILE_SIZE"` задает максимально допустимый размер закачиваемого файла в байтах. Опция `MAX_FILE_SIZE` при работе в системе Windows является рекомендацией браузеру, а не жестким ограничением. Можно закачать файл и существенно большего размера, чем указано в этом атрибуте. Если же ваш сервер Apache установлен в системе Linux, то эта опция работает в соответствии со своим названием.

Однако в настройках самого PHP есть параметр, накладывающий ограничение на максимальный размер файла. Это ограничение касается всех используемых сценариями файлов, и обойти его невозможно. Директивой, которая задает ограничения на размер файла, является `post_max_size`, устанавливающая максимальный допустимый размер `POST`-данных. Также влияет на закачиваемые файлы ограничение памяти `memory_limit` — максимальный объем памяти в байтах, который разрешается

использовать скрипту. Значение `memory_limit` должно быть больше чем значение `post_max_size`.

Сам процесс закачки файла не отражается ни в форме, ни в сценарии: он осуществляется по протоколу HTTP, и мы видим только результат — файл в каталоге для временного хранения.

Принятые файлы сохраняются на сервере во временном каталоге, имя которого задается при помощи директивы `upload_tmp_dir` конфигурационного файла php.ini. Поинтересуйтесь, какой каталог указан на вашей машине в этой директиве. После загрузки файл, как и все данные, получаемые от клиента, должен быть проверен. После проверки файл следует переместить в другой каталог. Если файл не переместить или не переименовать, прежде чем сценарий завершит работу, он будет уничтожен.

После успешного завершения загрузки на сервере инициализируется двумерный массив `$_FILES`. Этот массив используется для информации о загруженных файлах.

Структура массива `$_FILES` такова (`'userfile'` — имя файла, которое указал пользователь в форме):

- `$_FILES['userfile']['name']` — оригинальное имя файла на клиентской машине;
- `$_FILES['userfile']['type']` — MIME-тип файла, если браузер предоставил эту информацию;
- `$_FILES['userfile']['size']` — размер загруженного файла в байтах;
- `$_FILES['userfile']['tmp_name']` — временное имя файла, под которым загруженный файл был сохранен на сервере.

В PHP определено несколько функций, проверяющих безопасность загруженного файла. Вместо закачивания файлов со своего компьютера злоумышленник может попытаться локально скопировать файл с паролями или другой конфиденциальной информацией из какого-либо каталога нашего сервера в каталог, файлы которого предназначены для показа в клиентском браузере. Значит, нам надо проверить, действительно ли закачанный файл был загружен при помощи протокола HTTP. Для этого служит функция `is_uploaded_file($filename)`, которая возвращает `true`, если файл `$filename` был загружен при помощи HTTP методом `POST`.

Вот пример сценария проверки и сохранения загруженного файла (листинг 12.2).

Листинг 12.2. Проверка и копирование файлов при загрузке на сервер

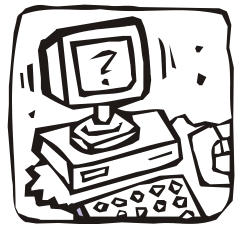
```
<html>
  <head>
    <title>Загрузка файлов на сервер</title>
  </head>
  <body>
  <?php
  $path = $_SERVER['DOCUMENT_ROOT']."/our_cat/";
```

```
  $file_name = $path.$_FILES['userfile']['name'];
  if (is_uploaded_file($_FILES['userfile']['tmp_name']))
  {
    copy($_FILES['userfile']['tmp_name'], $file_name);
    echo $_FILES['userfile']['tmp_name']."<br>";
    echo $_FILES['userfile']['name']."<br>";
    echo $_FILES['userfile']['type']."<br>";
    echo $_FILES['userfile']['size']."<br>";
  }
  else
  {
    echo "Возможна атака при загрузке файла".$_FILES['userfile']['name'];
  }
  ?>
  </body>
</html>
```

Если файл прошел проверку, то он копируется с помощью функции `copy()` в каталог с именем, определенным переменной `$path`. Каталог `our_cat` должен существовать.

Поэкспериментируйте с приведенными примерами на своей машине. Вам придется имитировать закачку с удаленной машины, совмещая клиента и сервер на одном компьютере, поэтому попытка закачать некоторые файлы будет распознана функцией `is_uploaded_file()` как попытка атаки. Попробуйте закачать файл из своих рабочих каталогов — обычно это проходит удачно. Потом попробуйте закачать файл из системного каталога Windows, вот это действие обычно воспринимается как атака, что вполне оправдано.

Попробуйте также закачать очень большой файл — вы должны увидеть, как сценарий отказывается его закачивать, сообщая, что размер закачанного файла равен нулю.



Глава 13

Объектная модель в PHP 5

В отличие от других языков, например JavaScript, PHP не является полностью объектно-ориентированным языком, и до сих пор в этой книге использовался процедурный стиль создания приложений. Объектный подход имеет смысл использовать в случае разработки крупных проектов, в которых планируется повторное использование кода, требуется систематизация и структуризация данных. Тем не менее есть немало причин, по которым основы объектно-ориентированного подхода в PHP следует осваивать с самого начала. Некоторые новые расширения ориентированы в первую очередь на применение объектно-ориентированного подхода. Примером может служить расширение mysqli, содержащее классы для работы с MySQL 5.

PHP 5 использует объектную модель, отличную от модели PHP 4. Код, написанный для PHP 4, будет, в основном, правильно работать в PHP 5, но могут потребоваться небольшие переделки. В этой главе объектный код создается в соответствии с моделью PHP 5.

Классы и объекты

Проблема процедурного программирования состоит в том, что данные и функции их обработки не связаны между собой. Классы предназначены для объединения свойств объектов и действий этих объектов.

Представьте себе класс млекопитающих. Принадлежность животного к этому классу определяется по наличию у животного некоторых характеристик — свойств. Например, кровь у млекопитающих теплая, при этом все млекопитающие могут совершать действия, например двигаться или кормить детенышей молоком.

На языке программирования свойство или атрибут — это переменная, имеющая некоторое значение. Действие, совершаемое объектом, — это функция, которую в объектном программировании принято называть *методом*. Мы можем объявить свойство:

```
public $blood;
```

Ключевое слово `public` указывает, что после него идет объявление элемента класса (свойства или метода). Кроме того, это ключевое слово определяет механизм доступа к элементу, о чем пойдет речь далее в этой главе. Присвоим свойству значение:

```
$blood =  "теплая";
```

Определим метод `move()`, который принимает один аргумент `$legs` — количество лап у животного:

```
public function move($legs)
{
  if ($legs) echo "$this->name двигается на $legs ногах <br>";
  else echo "Животное плавает";
}
```

Значение переменной `$legs` надо указать при вызове метода — ничего необычного в этом нет, так мы делали при вызове функции.

Специальный указатель `$this` применяется для обозначения объекта. Аналогично в процедурном программировании в определении функции указывается формальный аргумент. В приведенном здесь фрагменте кода распечатывается значение свойства `name` объекта, для которого вызывается метод `move()`.

Можно сказать, что класс млекопитающих характеризуется совокупностью свойств и методов. Какое-либо животное, относящееся к этому классу, будет обладать этими свойствами и методами, но возможно, что методы будут реализовываться по-разному (ног-то может быть разное количество).

Конструктор класса

При создании объекта — экземпляра класса — вызывается функция, которая инициализирует все требуемые переменные, выполнит все действия, нужные для полного определения объекта. Эта функция называется *конструктором*.

В PHP 5 конструктор — это метод, имеющий зарезервированное имя `__construct`, который может быть определен так:

```
function __construct($name)
{
  $this->name = $name;
  $this ->blood = 'теплая';
  echo "Запущен конструктор класса mammal <br>";
}
```

Свойства класса `name` и `blood` получают значения при вызове конструктора.

Код класса и создание объекта

Напишем определение класса млекопитающих (листинг 13.1).

Листинг 13.1. Класс `mammal`

```
<?php
class mammal
{
  public $blood;
  public function __construct($name)
```

```
  {
    $this->name = $name;
    $this->blood="теплая";
    echo "Запущен конструктор класса mammal <br>";
  }
  public function move($legs)
  {
    if ($legs) echo "$this->name двигается на $legs ногах <br>";
    else echo "Животное плавает";
  }
}
?>
```

Объект — экземпляр класса можно создать с помощью оператора `new`, после которого указывается имя класса и параметры, передаваемые конструктору:

```
$cat = new mammal("кошка");
```

При создании объекта `$cat` для него устанавливаются значения свойств. Получить доступ к ним можно так:

```
echo $cat->name;
```

Вызвать метод объекта `$cat` можно таким образом:

```
$cat->move(4);
```

Деструктор объекта

Объект можно уничтожить в ходе выполнения сценария, вызвав функцию `unset()` и передав ей в качестве параметра имя объекта. Но в любом случае при завершении работы сценария память, занимаемая объектом, высвобождается, и объект из нее удаляется. В объектной модели PHP 5 определена функция `__destruct()`, которая вызывается автоматически при уничтожении объекта.

В листинге 13.2 вы можете увидеть определение конструктора и деструктора класса, а также создание объекта как экземпляра созданного класса.

Листинг 13.2. Класс `mammal` и создание объекта

```
<?php
class mammal
{
  public $blood;
  public function __construct($name)
  {
    $this->name = $name;
    $this->blood = "теплая";
    echo "Запущен конструктор класса mammal <br>";
```

```
  }
  public function move($legs)
  {
    if ($legs) echo "$this->name двигается на $legs ногах <br>";
    else echo "Животное плавает";
  }
  function __destruct() {
    echo "Вызван деструктор объекта <br>";
  }
}
$cat = new mammal("кошка");
echo $cat->name."<br>";
$cat->move(4);
unset($cat);
echo "А теперь завершается работа сценария";
?>
```

При выполнении этого примера обратите внимание на то, что деструктор выполняется именно при вызове функции `unset()`. Если же вы удалите из сценария строку с этой функцией, то деструктор будет вызван в самом конце работы после выполнения всех остальных операторов. Деструктор — это подходящее место для действий, которые наводят порядок после выполнения различных работ: закрывают соединения с серверами баз данных, очищают память для предотвращения утечек памяти и т. п.

Вложенные объекты

Свойства объектов сами могут быть объектами. Тогда говорят, что объект вложен в другой объект (листинг 13.3).

Листинг 13.3. Вложенные объекты

```
<?php
class Room
{
  public $name;
  function __construct($name = "безымянная")
  {
    $this->name = $name;
  }
}
class House
{
```

```
  public $room;
}

$home = new House;
$home->room[] = new Room("спальня");
$home->room[] = new Room("кухня");
print($home->room[1]->name);
?>
```

В этом примере объект `$home` содержит массив вложенных объектов `room`.

Конструктор класса `Room` имеет один входной параметр, при этом у него есть значение по умолчанию, которое подставляется в случае, если конструктор был вызван без параметра.

Копирование и клонирование объектов

При копировании объектов не происходит копирование данных — создается только ссылка на область данных так же, как при создании ссылки на переменную.

Попробуем создать два объекта немного измененного класса млекопитающих — кошку и кита. Дадим кошке 4 лапы, а про кита укажем, что у него нет лап (`$legs = 0`). Посмотрим, что получилось (листинг 13.4).

Листинг 13.4. Копирование объектов

```
<?php
class simple_mammal
{
  public $legs;
}
$cat = new simple_mammal;
$cat->legs = 4;
$whale = $cat;
$whale->legs = 0;
echo $cat->legs;
echo $whale->legs;
?>
```

А ничего хорошего не вышло! Поскольку при копировании объекта не копируется область данных, то и у кошки лап не оказалось.

Чтобы скопировать свойства и методы объекта, надо применить клонирование (листинг 13.5).

Листинг 13.5. Клонирование объектов

```
<?php
class mammal
{
   public $legs;
}
$cat = new mammal;
$cat->legs = 4;
$whale = clone $cat;
$whale->legs = 0;
echo $cat->legs;
echo $whale->legs;
?>
```

Вот теперь все работает так, как задумывалось. Последние примеры призваны пояснить отличительные черты работы с объектами в PHP 5: при создании копии объекта с помощью оператора присваивания (`$whale = $cat`) создается ссылка на объект `$cat`, а не копия всех свойств и методов объекта `$cat`.

Наследование

Можно определить класс `beast`, являющийся наследником ранее определенного класса `mammal`:

```
class beast extends mammal
```

Класс-наследник получает все свойства и методы класса-родителя, но может дополнительно объявить и свои:

```
public $fur;  // объявляем новое свойство
```

Кроме того, класс-наследник может переопределить свойства и методы родителя (это называется *перегрузкой*). Например, перегрузим метод `move()`:

```
function move($legs)
{
  if ($legs) echo "$this->name бегает, лазает по деревьям на ".
                  $legs." лапах <br>";
}
```

Создадим новый метод, присущий только этому классу-наследнику:

```
function description()
{
  $this->fur = "мягкая и пушистая";
  echo $this->name, " ", $this->fur, " . ";
  echo "Кровь - ", $this->blood, "<br>";
}
```

Конструктор родительского класса не вызывается автоматически при создании объекта — экземпляра класса-наследника. Его следует вызвать явно, используя символ двойного двоеточия. Конструктор класса `beast` определяется следующим образом:

```
function __construct($name)
{
  parent::__construct($name);
  echo "Запущен конструктор класса beast <br>";
}
```

В итоге определение класса `beast` выглядит так (листинг 13.6). Вначале подключим код родительского класса, сохраненный в файле 13-2.php.

Листинг 13.6. Класс `beast`

```
<?php
include("13-2.php");
class beast extends mammal
{
  public $fur;
  function __construct($name)
  {
    parent::__construct($name);
    echo "запущен конструктор класса beast <br>";
  }
  function move($legs)
  {
    if ($legs) echo "$this->name бегает, лазает по деревьям на ".
                    $legs." лапах <br>";
  }
  function description()
  {
    $this->fur = "мягкая и пушистая";
    echo $this->name, " ", $this->fur, " . ";
    echo "Кровь - ", $this->blood, "<br>";
  }
}
?>
```

Можем теперь создать объекты класса-наследника и посмотреть, как это все работает:

```
$Murka = new beast("кошка");
$Murka->move(4);
$Murka->description();
```

Финальные классы

Теперь мы хотим создать класс `cat` — потомок класса `beast`. Класс `cat` является также потомком класса `mammal`. Мы хотим указать, что у этого класса `cat` никаких наследников быть не может. Для этого используется ключевое слово `final`:

```
final class cat extends beast
```

Из-за применения этого ключевого слова попытки создать классы-наследники класса `cat` будут вызывать сообщение об ошибке.

Объявим новое свойство `$sound` и определим его значение в конструкторе:

```
public $sound;
function __construct($name)
{
  parent::__construct($name);
  echo "Запущен конструктор класса cat <br>";
  $this->sound = "мурр";
}
```

Объявим и новый метод:

```
function speak()
{
  echo $this->name, " говорит ", $this->sound."<br>";
}
```

Объявление класса и создание объекта, экземпляра этого класса, будет выглядеть так, как показано в листинге 13.7.

Листинг 13.7. Класс `cat`

```
<?php
class mammal
{
  public $blood;
  public function __construct($name)
  {
    $this->name = $name;
    $this->blood = "теплая";
    echo "Запущен конструктор класса mammal <br>";
  }
  public function move($legs)
  {
    if ($legs) echo "$this->name двигается на $legs ногах <br>";
    else echo "Животное плавает";
  }
  function __destruct() {
    echo "Вызван деструктор объекта <br>";
```

```
  }
}
class beast extends mammal
{
  public $fur;
  function __construct($name)
  {
    parent::__construct($name);
    echo "запущен конструктор класса beast <br>";
  }
  function move($legs)
  {
    if ($legs) echo "$this->name бегает, лазает по деревьям на ".
                    $legs." лапах <br>";
  }
  function description()
  {
    $this->fur = "мягкая и пушистая";
    echo $this->name, " ", $this->fur, " . ";
    echo "Кровь - ", $this->blood, "<br>";
  }
}

final class cat extends beast
{
  public $sound;
  function __construct($name)
  {
    parent::__construct($name);
    echo "Запущен конструктор класса cat <br>";
    $this->sound = "мурр";
  }
  function speak()
  {
    echo $this->name, " говорит ", $this->sound."<br>";
  }
}
// Теперь создадим объект этого класса и вызовем его методы:
$Murka = new cat("кошка");
$Murka->move(4);
$Murka->description();
$Murka->speak();
?>
```

Объект `$Murka` создается, и все методы работают, но попробуйте добавить в конец этого сценария строки:

```
class ChildClass extends cat {
   public function moreTesting() {
       echo "Вызван метод ChildClass::moreTesting()\n";
   }
}
```

и вы увидите на экране сообщение об ошибке:

```
Fatal error: Class ChildClass may not inherit from final class (cat) in
C:\Program Files\Apache Group\Apache2\htdocs\13-6.php on line 64
```

Из приведенных примеров видно, когда стоит использовать объектно-ориентированный подход. Если бы зверей было раз, два и обчелся, никто бы не стал заводить их классификацию, упорядочивать перечень их свойств и методов. Но животных много, потребовалось структурировать информацию о них, вот и завели классы и семейства. Так и в программировании, затраты времени на создание классов окупятся, если надо писать большой проект.

Доступ к свойствам и методам класса

В объектно-ориентированном программировании доступ к свойствам класса осуществляется обычно с определенными ограничениями. Рекомендуется, чтобы работа со свойствами велась через вызов методов — это называется *инкапсуляцией*. Смысл инкапсуляции состоит в том, что внешний пользователь не знает детали реализации объекта, работая с ним путем предоставленного объектом интерфейса.

Для реализации инкапсуляции надо ввести ограничения на доступ к свойствам и методам класса (то и другое можно называть элементом класса) с помощью следующих спецификаторов.

- `public` — общедоступный или открытый. К свойствам и методам, помеченным этим спецификатором, можно получить доступ без каких-либо ограничений. В PHP 4 все свойства и методы объявлялись только как общедоступные, причем перед именем свойства писалось ключевое слово `var`. Объектный код, созданный на языке PHP 4, будет работать и с модулем 5-й версии, причем слово `var` будет трактоваться как определение свойства с открытым доступом.
- `protected` — защищенный или с ограничением доступа. Элементы этого типа доступны внутри класса, в котором они объявлены, и в его классах-наследниках. Оборот "внутри класса" означает, что прочесть защищенные свойства можно только с помощью метода, определенного в этом же классе.
- `private` — закрытый или приватный. Элементы этого типа доступны только внутри класса, в котором они объявлены.

Если не указать ни один из спецификаторов, то по умолчанию элемент будет общедоступным. Посмотрим, как все это работает, на примере. Создадим два класса (листинг 13.8).

Листинг 13.8. Типы доступа к свойствам класса

```
<?php
class mammal
{
  public $blood = "теплая";
  protected $legs = "4";
  private $eat = "молоко";
  public function printPrivate()
  {
     echo $this->eat;
  }
}

class cat extends mammal
{
   public function printProtected()
   {
     echo $this->legs;
   }
}
?>
```

Попробуем теперь создать экземпляр класса `mammal` и распечатать значения его свойств:

```
$Murka = new mammal;
echo $Murka->blood;
echo $Murka->classname;
```

Работает. А если так?

```
echo $Murka->legs;
```

Не выходит, получаем сообщение об ошибке:

```
Fatal error: Cannot access protected property mammal::$legs
```

А так?

```
echo $Murka->eat;
```

Нет, нельзя:

```
Fatal error: Cannot access private property mammal::$eat
```

Получить значение приватного свойства можно только через вызов метода:

```
$Murka->printPrivate();
```

Определим этот метод так:

```
public function printPrivate()
{
  echo $this->eat;
  echo $this->legs;
}
```

Вызов такого метода предоставит нам значения и приватного, и защищенного свойств. А в чем же между ними разница? Посмотрим, как обстоит дело с классом-наследником `cat`. Создадим новый экземпляр класса `cat` и вызовем его метод:

```
$Barsik = new cat();
$Barsik->printProtected();
```

Все отлично, значение защищенного свойства `$legs` читается. А теперь попробуем с помощью этого метода получить значение приватного свойства `$eat`, добавив его вывод в определение метода `printProtected()`:

```
public function printProtected()
{
  echo $this->legs;
  echo $this->eat;
}
```

Нет, видим в браузере:

```
Notice: Undefined property: cat::$eat
```

Теперь перечитайте определения спецификаторов доступа, и они станут вам понятнее.

Статические свойства и методы класса

В *главах 30* и *31*, изучая классы для работы с XML-документами, вы обнаружите в них статические свойства. Мы с вами видели, как определяются статические переменные в процедурном программировании на PHP. Можно объявить статические свойства класса так, как представлено в листинге 13.9.

Листинг 13.9. Статическое свойство класса

```
<?php
class mammal
{
  static $feeding = "молоко";
}
?>
```

Статические свойства едины для всего класса и не могут принадлежать ни одному из объектов класса. Кроме этого, можно обратиться к такому свойству, не создавая объекта:

```
echo mammal::$feeding;
```

Аналогично можно определить статический метод и использовать его без создания объекта такого класса (листинг 13.10).

Листинг 13.10. Статический метод класса

```
<?php
class mammal
{
  static function moving()
  {
    echo "Все звери двигаются ";
  }
}
echo mammal::moving();
?>
```

Однако в статическом методе становится невозможным использовать указатель `$this`, т. к. при вызове статического метода неизвестно, в контексте какого объекта он вызывается.

Абстрактные классы и интерфейсы

Представьте себе, что вы создаете сетевую игру, участники которой ведут борьбу с противниками. Каждый игрок представлен фигуркой того или иного существа, обладающего разными видами оружия, защиты и пр. Для описания этих разнообразных и многочисленных героев вам придется создавать много классов — родителей и наследников, описывать в них методы, которыми могут воспользоваться участники игры. Возможно, в одном из базовых классов вы захотите только указать наличие некоторого метода, а конкретную его реализацию отложить до перегрузки его в классе-наследнике. Это можно сделать с помощью абстрактного метода.

Абстрактным называется метод, который имеет только объявление, но не имеет реализации. Реализация же выполняется в классе-наследнике. Класс, который содержит абстрактный метод, объявляется как абстрактный. При этом в нем могут содержаться и обычные, неабстрактные, методы.

Создать объект, являющийся экземпляром абстрактного класса, невозможно. Но вот после того, как в классе-наследнике абстрактный метод будет переопределен, можно уже создавать экземпляры этого класса-наследника (см. листинг 13.11).

Листинг 13.11. Абстрактный класс

```
<?php
abstract class voin {
  abstract public function deistvie();
}
class super_voin extends voin {
  public function deistvie() {
    echo "Появилась реализация метода ";
```

```
  }
}
$obj = new super_voin;
$obj->deistvie();
?>
```

В PHP невозможно описать класс, являющийся наследником сразу двух классов, даже абстрактных. Для решения этой проблемы существуют *интерфейсы*, представляющие собой абстрактные классы, не содержащие ни одного неабстрактного метода. Класс может наследоваться от двух интерфейсов одновременно, переопределяя их методы. Можно создавать объекты — экземпляры такого класса-наследника. В листинге 13.12 представлено объявление двух интерфейсов и класса — их общего наследника.

Листинг 13.12. Интерфейсы

```
<?php
interface voin{
  function shoot();
}
interface artist{
  function paint();
}
class hero implements voin, artist {
  public function shoot() {
    echo "Герой умеет стрелять. ";
  }
  public function paint() {
    echo "Герой умеет рисовать. ";
  }
}
$obj = new hero;
$obj->shoot();
$obj->paint();
?>
```

Константа класса

В PHP 5 введен такой элемент класса, как *константа* (листинг 13.13).

Листинг 13.13. Константа класса

```
<?php
class baza
{
```

```
  const DBNAME = "taxi";
}
echo baza::DBNAME;
?>
```

Из приведенного листинга ясно, что использовать константу можно без создания объекта. Константы класса пригодятся нам для хранения параметров подключения к серверу баз данных или других данных, не подлежащих изменениям в ходе выполнения сценария.

Ключевое слово *instanceof*

Ключевое слово `instanceof` позволяет определить, является ли объект экземпляром определенного класса (назовем класс `country`) или экземпляром класса-наследника (класс-наследник назовем `city`) определенного класса (листинг 13.14).

Листинг 13.14. Ключевое слово `instanceof`

```
<?php
class country { }
$obj1 = new country();
if ($obj1 instanceof country) {
  echo "\$obj1 - объект класса country";
}
class city extends country{ }
$obj2 = new city();
if ($obj2 instanceof country) {
   echo "\$obj2 - объект класса, производного от country";
}
?>
```

Обработка ошибок

Посмотрим теперь, как в объектном коде реализована обработка ошибок. Мы видели, что при написании процедурного кода не рекомендуется выводить сообщения о возможных ошибках, генерируемые самим модулем PHP, на экран. Лучше предусмотреть возможность ошибки и создать свой код ее обработки. Например, может возникнуть проблема с открытием файла, так не будем заставлять пользователя созерцать неудобочитаемые сообщения, а создадим свое, подходящее к случаю.

Для обработки ошибок в ядро PHP 5 встроен класс `Exception` (в листинге 13.15 представлена структура класса с комментариями, но не описана реализация, это не рабочий пример, а демонстрация идеи). Конечно, нельзя разрешать пользователю переопределять методы этого класса, поэтому они объявлены как финальные.

Листинг 13.15. Класс `Exception`

```
<?php
class Exception
{
  protected $message = 'Unknown exception';  // сообщение об исключении
  protected $code = 0;   // код исключения, определяемый пользователем
  protected $file;       // имя файла-источника исключения
  protected $line;       // строка, в которой произошло исключение
  function __construct($message = null, $code = 0);
  final function getMessage();       // сообщение об исключении
  final function getCode();          // код исключения
  final function getFile();          // файл источника
  final function getLine();          // строка источника
  final function getTrace();         // массив backtrace()
  final function getTraceAsString(); // форматированная строка трассировки

  /* Overrideable — можно перегружать */
  function __toString(); // Создание строки для отображения на экране
?>
```

В PHP, как и во многих объектно-ориентированных языках, реализована схема обработки исключений с помощью конструкции `try...catch...throw`. Она позволяет весь код обработки ошибок локализовать в одном месте сценария. Код, при выполнении которого может появиться ошибка, пишется внутри блока `try`.

Попытаемся открыть несуществующий файл:

```
$fp = fopen("file.txt", "r");
```

Получаем сообщение об ошибке. Скроем ее от пользователя:

```
@$fp = fopen("file.txt", "r");
```

Теперь напишем объектный код так, чтобы при возникновении ошибки, т. е. в случае, когда переменная `$fp` получила значение `false`, создавался новый экземпляр `$exception` класса `Exception`. Создавать его будем, конечно, оператором `new`. Передадим в конструктор класса `Exception` в качестве значения свойства `$message` сообщение об исключении "Невозможно открыть файл!" Если ошибки не случится, ну и хорошо — запишем все, что нужно, в файл. Но если возникнет ошибка, то она будет перехвачена конструкцией `catch`, которая напечатает "Ошибка в строке ", затем вызовет метод `getLine()` для нашего объекта `$exception` и метод `getMessage()`. Эти методы вернут номер строки скрипта, в которой произошла ошибка, и значение свойства `$message`. Синтаксис обработки ошибок представлен в листинге 13.16.

Листинг 13.16. Обработка ошибок с помощью `try...catch...throw`

```
<?php
try
{
  @$fp = fopen("file.txt", "w");
  if (!$fp) throw new Exception("Невозможно открыть файл!");
  // Запись данных в файл при отсутствии ошибок
  fclose($fp);
}
catch (Exception $exception)
{
  echo "Ошибка в строке ", $exception->getLine();
  echo $exception->getMessage();
}
?>
```

Автозагрузка класса

Итак, пусть у нас готовится большой проект, при реализации которого мы решили использовать объектный подход. Определили классы и записали каждый класс в отдельный файл — так мы сможем соблюсти принцип модульности программного обеспечения и сможем подключать объявления классов к любому скрипту, используя, например, инструкцию `include`. Теперь каждый из описанных ранее классов хранится в файлах mammal.php, beast.php и cat.php. Для их подключения в начале скрипта пишем:

```
include ("mammal.php");
```

и т. д. для всех классов. Если мы забудем подключить какой-то файл с классом, то получим ошибку при создании объекта.

В PHP 5 есть возможность загрузки классов с помощью глобальной функции `__autoload()`, которая принимает один параметр — имя класса. Она вызывается при попытке создать экземпляр класса, не определенного в текущем сценарии. Эта функция встроена в ядро PHP, при ее вызове ядро передает ей в качестве параметра имя вызываемого класса. Переопределим действия, которые функция выполняет по умолчанию так, чтобы при попытке создать объект непределенного класса происходил поиск и подключение файла с кодом класса. Для этого нам потребуется ввести некоторое правило относительно имен файлов: имя файла должно совпадать с именем класса, а сам файл должен быть сохранен в том же каталоге, где хранится сценарий, создающий экземпляр класса. Вот теперь инструкция `include()` сможет выполнить все требуемые действия (листинг 13.17).

Листинг 13.17. Автозагрузка класса

```
<?php
function __autoload($class)
{
  include($class.".php");
}
$Barsik = new mammal("кошка");
echo $Barsik->move(4);
?>
```

При создании классов и файлов надо помнить о чувствительности к регистру в именах файлов и при необходимости выполнять соответствующие преобразования имен.

Объекты также могут сохраняться в сессии, но при чтении объекта из сессии для корректной работы необходимо подключить описание класса к тому скрипту, в котором происходит такое чтение. Функция `__autoload()` поможет и в этом случае, позволяя загружать коды тех классов, экземплярами которых являются объекты, хранимые в сессии.

Итераторы: просмотр всех общедоступных свойств объекта

В объектной модели определена возможность перебора всех открытых свойств объекта с помощью оператора цикла `foreach()`. Это и выполнено в листинге 13.18.

Листинг 13.18. Использование итератора

```
<?php
class mammal
{
  public $blood, $legs;
  public function __construct($name)
  {
    $this->name = $name;
    $this->blood = "теплая";
    $this->legs = 4;
  }
}
```

```
$Murka = new mammal("Кошка");
foreach ($Murka as $svoistvo => $znachenie) {
  echo "$svoistvo => $znachenie";
}
?>
```

Цикл `foreach` перебирает все свойства объекта `$Murka`, имеющие спецификатор `public`. Каждый раз имя свойства записывается в переменную `$svoistvo`, а его значение — в переменную `$znachenie`.

В PHP 5 создан мощный механизм работы с объектами, в настоящей главе он описан только кратко, за полной информацией следует обращаться к сайту **php.net**.

ЧАСТЬ II

PHP и MySQL

Глава 14

Реляционные базы данных

База данных — это централизованное хранилище данных, обеспечивающее доступ, первичную обработку и поиск информации. В настоящее время наибольшее распространение получили *реляционные базы данных*. В реляционных базах данных все данные представлены в виде таблиц, разбитых на строки и столбцы. Каждая строка (запись) имеет фиксированный набор полей. База данных может включать в себя несколько таблиц, связанных друг с другом.

Для работы с базой данных необходима система управления базами данных (СУБД), т. е. программа, которая берет на себя все заботы, связанные с доступом к данным. Она содержит команды, позволяющие создавать таблицы, вставлять в них записи, осуществлять поиск данных, а также их изменение и удаление.

СУБД обеспечивает безопасность данных. Пользователям системы предоставляются определенные права доступа к информации. Некоторым пользователям разрешено лишь просматривать данные, тогда как другие пользователи могут менять содержимое таблиц.

СУБД поддерживает параллельный доступ к базе данных. Приложения могут обращаться к базе данных одновременно, что повышает общую производительность системы.

Наконец, СУБД помогает восстанавливать информацию в случае непредвиденного сбоя, незаметно для пользователей создавая резервные копии данных. Все изменения, вносимые в базу данных, регистрируются, поэтому многие операции можно отменять и выполнять повторно.

По некоторым оценкам Web-мастер тратит около 90% своего рабочего времени на работу с базами данных.

MySQL представляет собой сервер реляционных баз данных, отличающийся высокой надежностью и скоростью. MySQL функционирует по модели "клиент-сервер".

По умолчанию запрос от клиентской программы на подключение поступает через порт 3306 сервера MySQL. Этот порт постоянно прослушивается сервером. При ответе на запрос сервер создает сеанс связи с клиентом. За сеансом закрепляются два порта: один используется для отправки данных, а другой — для их приема.

Язык SQL (Structured Query Language, язык структурированных запросов) является общепринятым стандартом языка работы с реляционными базами данных. В ходе сеанса клиент посылает серверу команды, имеющие вид инструкций SQL. В ответ на некоторые инструкции сервер возвращает данные, а клиентская программа форматирует их для отображения на экране.

Таблицы, записи, столбцы

Рассмотрим работу фирмы, владеющей несколькими такси и нанимающей водителей для работы на них. Руководство фирмы решило хранить информацию об автомобилях и водителях, работающих в компании, так, чтобы информация была полной, не повторялась многократно в разных файлах, и легко было найти нужные данные.

Такую информацию целесообразно хранить в базе данных. Реляционная база данных состоит из именованных таблиц. Таблица состоит из строк (записей) и столбцов (полей). В столбце хранятся соответствующие значения каждой строки, каких-либо пропусков или коротких столбцов быть не может. Запись является отдельной сущностью, а поля представляют собой атрибуты записей. У каждого столбца есть название и тип. Столбцы располагаются в порядке, который определяется при создании таблицы, в отличие от строк, которые располагаются в произвольном порядке. В таблице может не иметься ни одной строчки, но обязательно должен быть хотя бы один столбец.

Первая таблица `cars` содержит сведения об автомобилях:

- модель;
- год выпуска;
- государственный регистрационный номер;
- цвет;
- учетный номер автомобиля в автопарке.

Вторая таблица, назовем ее `drivers`, предназначена для хранения информации о водителях:

- имя;
- отчество;
- фамилия;
- дата рождения;
- домашний адрес;
- дата приема на работу;
- учетный номер водителя.

Таблица `timetable` содержит данные о расписании использования того или иного автомобиля:

- дата;
- учетный номер автомобиля в автопарке;
- учетный номер водителя.

Каждая таблица будет состоять из нескольких строк — записей о водителях, автомобилях и расписании использования автомобиля. Строки таблицы не хранятся в определенном порядке. Упорядочивать их пользователи могут по своему усмотрению. Порядок строк в таблице отражает очередность записи строк на диск. Каждая строка состоит из набора значений.

С другой стороны, таблица состоит из нескольких столбцов, каждый из которых имеет имя и может содержать данные только определенного типа.

Каждое значение имеет определенный тип данных, задаваемый для всего столбца в целом.

Для того чтобы отыскать данные о водителе, надо указать его фамилию. Но фамилия, а иногда даже имя и отчество одновременно, не позволяют однозначно определить водителя. Читатели, имеющие распространенные сочетания имен и фамилий, сразу поймут почему. Принимая человека на работу, ему присваивают некоторый учетный номер, однозначно идентифицирующий сотрудника фирмы. Этот номер пишут на пропуске и в бухгалтерских документах. Будем называть столбец в таблице, однозначно идентифицирующий запись, *первичным ключом (primary key)*. Сервер баз данных не допустит, чтобы две записи имели одинаковые значения в столбцах первичного ключа.

Отношения и ключи

Между таблицами могут существовать отношения трех типов: "один-ко-многим", "один-к-одному" и "многие-ко-многим".

Таблица `cars` связана с таблицей `timetable`. Одна и та же машина, описанная в таблице `cars`, может несколько раз упоминаться в таблице `timetable`. Такая связь называется отношением "один-ко-многим" (1 : *N*). Аналогично, каждый водитель из таблицы `drivers` может несколько раз упоминаться в таблице `timetable`. Отношения 1 : *N* являются основными отношениями, поддерживаемыми серверами баз данных.

Кроме этого, может встречаться отношение "один-к-одному" (1 : 1). Например, при поступлении на работу водитель представил рекомендации. Их тоже можно занести в базу, заведя для этой цели новый столбец. Но рекомендации — это длинный текст, и есть они не у всех, значит, несколько полей останутся пустыми, это будет необоснованной тратой памяти. Можно завести отдельную таблицу `driver_ref`, содержащую два столбца — учетный номер водителя и сам столбец с текстом рекомендации. Отношение между таблицами `drivers` и `driver_ref` будет отношением "один-к-одному", ведь у каждого водителя есть только один текст рекомендаций. Обычно, если вы проектируете базу и обнаруживаете, что между таблицами существует связь "один-к-одному", это означает, что таблицы можно объединить. Проверяйте, не напрасно ли вы создаете дополнительные таблицы, но в данном примере создание таблицы оправдано.

В реляционной базе данных нельзя напрямую создать отношение "многие-ко-многим" (*M* : *N*). Его необходимо преобразовать в два отношения 1 : *N*, устанавливаемых с промежуточной таблицей.

Если значения одного поля таблицы представлены в поле другой таблицы, мы говорим, что первое поле ссылается на второе. Когда один столбец в таблице ссылается на другой, он называется *внешним ключом*; а столбец, на который он ссылается, называется *родительским ключом*. Ключи служат связующим звеном между таблицами. Внешний ключ (foreign key) — это столбец, который соответствует первичному ключу другой таблицы.

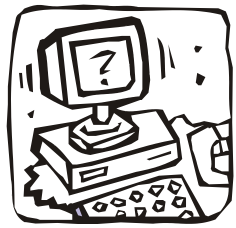
В нашем примере учетный номер автомобиля в автопарке и учетный номер водителя являются первичными ключами в таблицах `cars` и `drivers` и внешними ключами в таблице `timetable`. Сервер проверяет соответствие значений в столбцах, по которым происходит связь. Вы не сможете ввести такое значение в поле

внешнего ключа, которое отсутствует в поле первичного ключа. То есть надо сначала принять водителя на работу и присвоить ему номер, а уж потом использовать этот номер в расписании автопарка.

Значения в поле первичного ключа должны быть уникальными.

Полный набор таблиц для базы данных называется *схемой базы данных*. Схема должна показывать таблицы вместе с их столбцами, типы данных столбцов, указывать первичный ключ каждой таблицы и все внешние ключи.

Теория баз данных основана на теории множеств, которая определяет, какие операции с данными допустимы, из нее следуют и рекомендации по проектированию баз данных. Проектируя таблицу, можно предусмотреть одно поле для записи имени, отчества и фамилии водителя, но в этом случае искать человека только по фамилии окажется сложно — потребуется не просто считывать значение поля с фамилией, но и анализировать его. Следовательно, в каждом поле таблицы должна находиться запись только об одной сущности.



Глава 15

Установка сервера MySQL 5 в Windows

Для создания баз данных и работы с ними нам потребуется установить сервер MySQL и утилиты, которые поставляются вместе с сервером. Дистрибутив сервера можно взять с сайта **dev.mysl.com**. Это должен быть zip-архив из раздела **Windows downloads** (Windows (x86)). Далее в этой главе описывается установка, настройка и работа с MySQL 5.1.

Разархивируйте пакет и запустите на выполнение файл с расширением exe, затем выберите типичный способ установки. По возможности зарегистрируйте свою копию сервера через Интернет или откажитесь от регистрации, выбрав соответствующий переключатель. В следующем окне обратите внимание на расположение файлов устанавливаемого сервера (рис. 15.1): в каталог, указанный как **Destination Folder**, будут помещены исполняемые и конфигурационные файлы сервера. Сами же базы данных будут храниться в каталоге, помеченном как **Data Folder**.

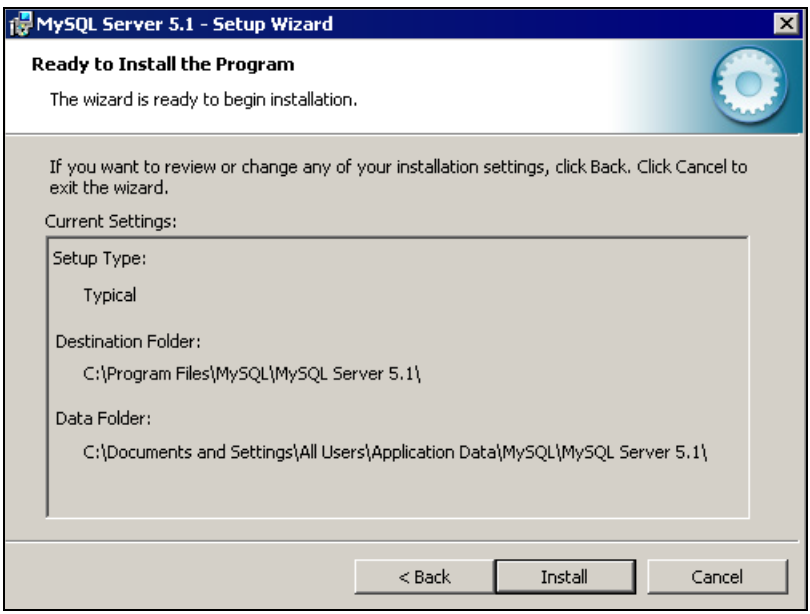


Рис. 15.1. Расположение файлов сервера MySQL

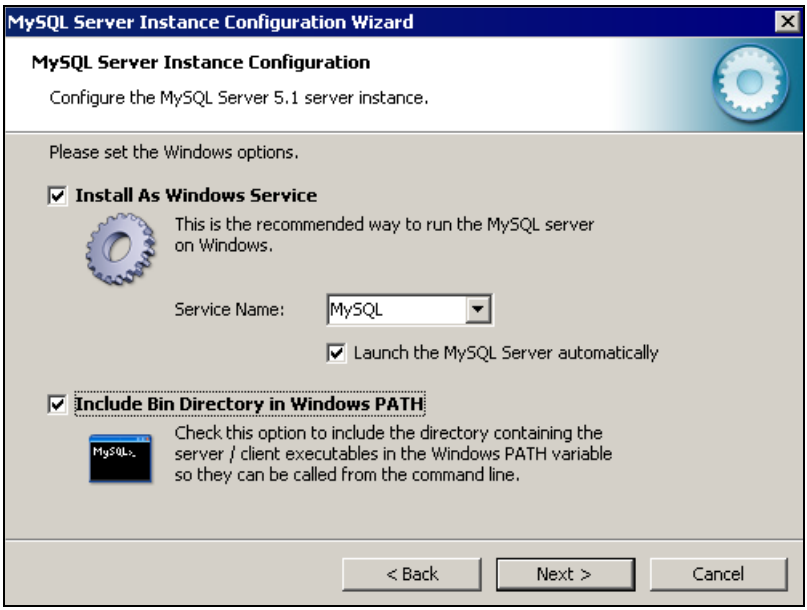


Рис. 15.2. Выбор режима конфигурации сервера MySQL

Затем сконфигурируйте сервер с помощью мастера (wizard), выбрав стандартный вариант конфигурации сервера (рис. 15.2).

Выберите установку сервера MySQL как сервиса Windows, тогда MySQL будет запускаться при старте системы автоматически. Выберите имя для сервиса, например MySQL. Под этим именем вы в дальнейшем найдете его в списке сервисов в окне **Панель управления | Администрирование | Службы**.

Укажите также, что необходимо включить путь к каталогу bin в переменную окружения Windows `PATH`. Переменная `PATH` содержит имена всех каталогов, в которых система ищет программу, запускаемую пользователем из командной строки. Если окажется, что программа находится в каталоге, не указанном в переменной `PATH`, то Windows не сможет ее найти и сообщит о невозможности запуска приложения.

В каталоге bin сервера MySQL содержатся те файлы, которые требуется запускать при старте сервера, а также утилиты, которые позволят нам связываться с сервером напрямую из командной строки.

На следующем этапе задайте пароль для суперпользователя `root` (рис. 15.3).

> ***Примечание***
>
> Суперпользователь `root` — это владелец системы, он может выполнять любые действия по управлению сервером и администрированию баз данных.

Требования к качеству пароля здесь нежесткие, можете выбрать любое словарное слово, записав его латиницей. Не разрешайте удаленный доступ к серверу для суперпользователя.

Рис. 15.3. Конфигурирование доступа к серверу MySQL

Примечание

Разрешение удаленного доступа к серверу существенно ослабит его безопасность и может быть выдано, только если доступ осуществляется с шифрацией передаваемых данных.

В том же окне укажите, что требуется создать учетную запись для анонимного пользователя.

Примечание

Анонимный пользователь заходит в систему и при этом получает ограниченные права, в частности он не имеет доступа к системным базам данных.

Завершите установку и убедитесь, что сервис MySQL запущен. Найдите среди установленных на вашем компьютере программ группу **MySQL** и запустите клиентскую программу **MySQL Command Line Client** для доступа к серверу из командной строки.

Если установка завершилась благополучно, то сервер у вас уже запущен, клиентская утилита `mysql` установит соединение с сервером и предложит ввести пароль. Введите пароль суперпользователя. Таким образом, вы соединитесь с сервером MySQL под именем суперпользователя `root`, т. е. будете иметь все возможные права для работы. Для завершения сеанса работы с сервером MySQL достаточно дать команду `quit` в окне клиентского приложения, и его окно закроется.

Но не торопитесь создавать базы с данными, записанными в какой-либо кодировке русского языка. Дело в том, что, начиная с версии MySQL 4.1, задать кодировку можно не только для всего сервера, но и для отдельной базы, таблицы и даже столбца. Причем в вашей базе данных столбцы могут иметь как различную кодировку, так и разные правила сравнения данных.

Сравнение или сопоставление (от англ. *collation*) — это способ упорядочивания строк, например, по алфавиту. При этом можно учитывать регистр (строчная или заглавная попалась буква), а можно игнорировать его. При сопоставлении можно задать порядок в соответствии с кодировкой символов. Кроме того, вы можете указать, что сравнивать придется двоичный код или текст. Это все вместе и называется сопоставлением.

При установке сервера MySQL в Windows необходимо выполнить настройку сервера с учетом кодировки и способа сравнения данных в базе, иначе русские слова будут отображаться в виде строк из вопросительных знаков.

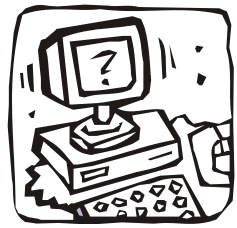
Итак, для настройки MySQL откройте каталог, в который был установлен сервер, и найдите файл my.ini. В этот конфигурационный файл и необходимо внести приведенные далее директивы.

Найдите в файле my.ini раздел, касающийся настройки самого сервера, он помечен символами `[mysqld]`. Строки, начинающиеся с символа решетки (`#`), являются комментариями. Найдите директиву `default-character-set` (задается кодировка по умолчанию), измените ее и добавьте еще 3 директивы так, чтобы этот фрагмент файла имел следующий вид:

```
default-character-set=utf8
character-set-server=utf8
init-connect="SET NAMES utf8"
skip-character-set-client-handshake
```

Две последние строки принудительно устанавливают кодировку utf8 для всех запросов. Эти директивы позволяют создавать таблицы в кодировке utf8 и благополучно выполнять запросы, используя русскоязычные строки. Но хотелось бы предостеречь от создания столбцов, таблиц и баз с русскими именами. Иначе в самых неожиданных местах вас могут подстерегать ошибки и некорректная обработка данных.

После внесения этих изменений перезапустите сервер MySQL, используя соответствующий пункт в окне служб, запущенных в операционной системе.



Глава 16

Создание баз данных

Перед созданием базы данных надо подумать о том, какие типы данных будут использованы.

Типы данных MySQL

Строковые типы

В MySQL определены следующие строковые типы (табл. 16.1).

Таблица 16.1. *Строковые типы данных и требуемый объем памяти*

Тип данных	Необходимый объем памяти
CHAR(*M*)	*M* байтов, $0 \le M \le 255$
VARCHAR(*M*)	*L* + 1 байтов, где $L \le M$ и $0 \le M \le 65\,535$ (было $0 \le M \le 255$ до версии MySQL 5.0.3)
BINARY(*M*)	*M* байтов, $0 \le M \le 255$
VARBINARY(*M*)	*L* + 1 байтов, где $L \le M$ и $0 \le M \le 255$
TINYBLOB, TINYTEXT	*L* + 1 байтов, где $L < 2^8$
BLOB, TEXT	*L* + 2 байтов, где $L < 2^{16}$
MEDIUMBLOB, MEDIUMTEXT	*L* + 3 байтов, где $L < 2^{24}$
LONGBLOB, LONGTEXT	*L* + 4 байтов, где $L < 2^{32}$
ENUM('*value1*','*value2*',...)	1 или 2 байта в зависимости от числа значений (65 535 значений максимум)
SET('*value1*','*value2*',...)	1, 2, 3, 4 или 8 байтов в зависимости от числа значений (64 значения максимум)

Типы CHAR (строки фиксированной длины) и VARCHAR (строки произвольной длины) — это типы, предназначенные для записи коротких фрагментов текста. Длину каждого из них можно регулировать числом *M*. Записи в столбцы типа CHAR будут дополняться пробелами до максимальной длины. При этом длина строки не зависит

от размеров данных, в то время как в столбцах с типом VARCHAR размер поля зависит от размеров данных. MySQL удаляет пробелы в конце текстовых строк у CHAR во время извлечения и у VARCHAR во время сохранения.

Тип VARCHAR позволяет экономить память, но при хранении данных приходится указывать, сколько памяти занимает введенное значение. Это значение сервер использует для того, чтобы определить, с какой позиции начнется запись следующего значения. При записи данных типа CHAR серверу не приходится указывать в каждом поле реальную длину строки, а при извлечении данных считывать это значение, чтобы определить, где начнется следующее значение поля, поэтому сервер работает с такими строками быстрее, зато используется несколько больший объем памяти.

Если мы при определении таблицы зададим тип данных в поле фамилии водителя как CHAR(10), а потом попробуем ввести более длинную фамилию, то вводимый текст будет усечен до указанной в определении таблицы длины.

Тип BLOB (Binary Large Object, большой двоичный объект) используется для хранения двоичных данных, например файлов фотографий.

Данные типа ENUM (перечисление) могут принимать одно из нескольких заранее заданных значений: ENUM('*value1*','*value2*',...). Например, предусмотрено, что в некотором столбце должен храниться ответ на вопрос, а допустимыми значениями ответа являются "да" и "нет". Тогда '*value1*'будет "да", а '*value2*' — "нет".

Тип SET (множество) предусматривает, что в поле одновременно может содержаться несколько значений из заранее заданного списка.

Форматы записи даты и времени

Дату и время можно хранить в одном из следующих форматов (табл. 16.2).

Таблица 16.2. *Типы данных даты и времени*

Тип данных	Описание	Диапазон значений
DATETIME	YYYY-MM-DDHH:MM:SS	'1000-01-01 00:00:00' — '9999-12-31 23:59:59'
DATE	YYYY-MM-DD	'1000-01-01' — '9999-12-31'
TIMESTAMP	YYYY-MM-DD HH:MM:SS	'1970-01-01 00:00:00' — 'Э 2037-12-31 23:59:59'
TIME	HH:MM:SS	'-838:59:59' — '838:59:59'
YEAR	YY или YYYY	1970 — 2069 или 1901 — 2155

Отсчет времени в MySQL осуществляется с начала эпохи UNIX — 1 января 1970 г.

Хранение числовых значений

Целые числа могут храниться со знаком (signed, по умолчанию) и без (unsigned). Для указания чисел без знака после целочисленного типа применяется ключевое слово UNSIGNED и означает, что его значения могут быть только положительными либо нулевыми. Допустимые форматы целых чисел представлены в табл. 16.3.

Таблица 16.3. *Целые числа*

Тип данных	Память в байтах	Диапазон значений со знаком	Диапазон значений без знака
TINYINT	1	–128 — 127	0 — 255
SMALLINT	2	–32 768 — 32 767	0 — 65 535
MEDIUMINT	3	–8 388 608 — 8 388 607	0 — 16 777 215
INT	4	–2 147 483 648 — 2 147 483 647	0 — 4 294 967 295
BIGINT	8	-2^{63} — $2^{63}-1$	0 — $2^{64}-1$

Таблица 16.4. *Числа с плавающей точкой*

Тип данных	Память в байтах
FLOAT(*p*)	4, если $0 \le p \le 24$; 8, если $25 \le p \le 53$
FLOAT	4
DOUBLE [*M*]	8
DECIMAL(*M*,*D*), NUMERIC(*M*,*D*)	Переменная

Числа с плавающей точкой могут храниться в столбцах следующих типов (табл. 16.4).

Данные типа FLOAT или DOUBLE округляются при записи до указанной в определении столбца точности.

Тип DECIMAL предназначен для хранения данных, которые не могут быть округлены, например денежных величин. В MySQL версии 5.0.2 и более ранних данные этого типа хранятся в виде текста. Начиная с версии 5.0.3, данные типа DECIMAL хранятся в двоичном виде, в котором десять двоичных цифр записываются в 4-х байтах.

Работа с клиентской программой *mysql*

Утилита mysql позволяет осуществить соединение с сервером MySQL и выполнять команды SQL для создания баз, записи в них данных и выборки данных из баз. В версии MySQL 5 реализован оконный интерфейс для работы с mysql. Его можно вызвать, выбрав пункт системного меню **MySQL | MySQL Command Line Client**.

При запуске утилита mysql установит соединение с сервером и предложит ввести пароль. Введите пароль суперпользователя root.

Команда на языке SQL обычно состоит из выражения, за которым следует точка с запятой. Когда пользователь вводит команду, mysql отправляет ее серверу для выполнения и выводит на экран сначала результаты, а затем — новую строку mysql>, что означает готовность к выполнению новых команд.

Строка mysql> называется *приглашением командной строки*.

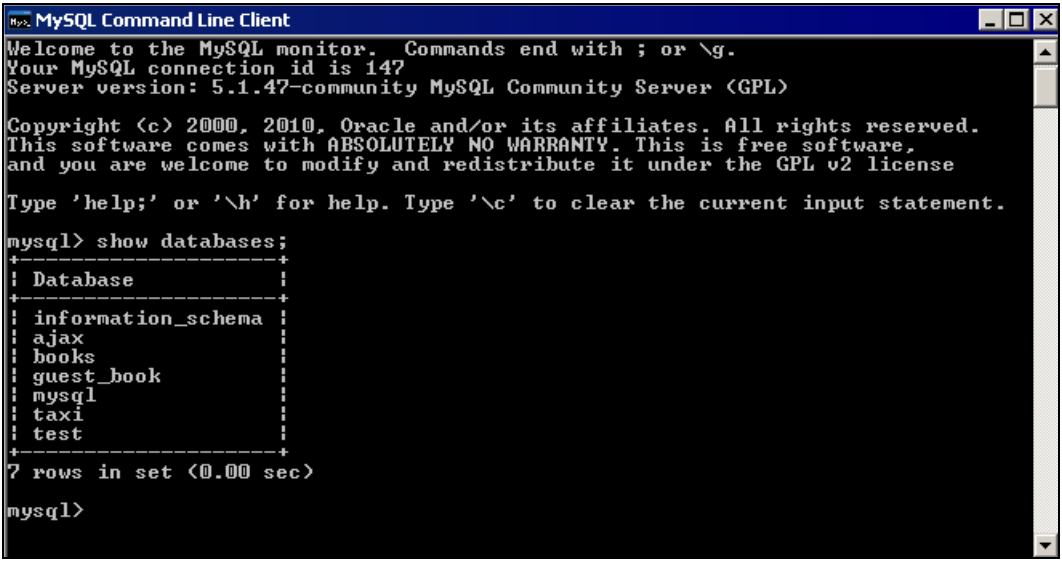


Рис. 16.1. Окно утилиты `mysql`

Утилита `mysql` выводит результаты работы запроса, если таковые есть, в виде таблицы (строк и столбцов) (рис. 16.1). В первой строке этой таблицы содержатся заголовки столбцов, а в следующих строках — сами результаты. Обычно заголовками столбцов становятся имена, полученные из таблиц базы.

Утилита `mysql` также сообщает количество возвращаемых строк и время выполнения запроса, что позволяет в некоторой степени составить представление о производительности сервера.

Для `mysql` признаком завершения команды является точка с запятой, а не конец строки. Команды собираются, но не исполняются до тех пор, пока программа не обнаружит точку с запятой. Вот пример команды, требующей вывести список баз данных сервера и занимающей несколько строк:

```
mysql> SHOW
    -> DATABASES;
```

Обратите внимание на то, как изменилась метка командной строки (с `mysql>` на `->`) после ввода первой строки этого запроса. Таким образом `mysql` показывает, что завершенного выражения она пока что не получила и ожидает его полного ввода. Эта метка очень полезна, т. к. предоставляет весьма ценную информацию о состоянии программы. С ее помощью всегда можно узнать, чего ждет `mysql`.

Команды, указание на тип данных и прочие зарезервированные слова принято вводить заглавными буквами, хотя чувствительности к регистру здесь нет. Иное дело имена таблиц и баз данных: здесь необходимо соблюдать выбранный регистр, а вот имена столбцов нечувствительны к регистру, но их принято писать строчными буквами.

В табл. 16.5 приведены все возможные варианты вида метки командной строки и соответствующие им состояния утилиты `mysql`.

Таблица 16.5. *Метки командной строки MySQL*

Метка	Значение
`mysql>`	Ожидание новой команды
`->`	Ожидание следующей строки многострочной команды
`'>`	Ожидание следующей строки, сбор строкового выражения, начинающегося с одиночной кавычки
`">`	Ожидание следующей строки, сбор строкового выражения, начинающегося с двойной кавычки

Создание базы данных *taxi*

Вспомним о фирме, предлагающей услуги такси. В этой фирме на нескольких автомобилях работает посменно несколько водителей. Мы хотим иметь базу данных, в которой хранились бы сведения о водителях, данные об автомобилях и расписание их работы с указанием фамилий работавших водителей. Создадим базу данных `taxi` таксопарка командой `CREATE DATABASE`:

```
mysql> CREATE DATABASE taxi;
```

Команда должна завершаться точкой с запятой (этот символ называется разделителем — delimiter). Получив команду, сервер выполняет ее и выдает сообщение (при успехе — `Query OK, 1 row affected`).

При создании базы данных она автоматически не выбирается, выбирать ее нужно отдельно. Для этого надо дать команду использовать только что созданную базу данных:

```
mysql> USE taxi;
```

Создавать базу нужно только однажды, но выбирать ее приходится в начале каждого сеанса работы с `mysql`.

Таблицы создаются командой `CREATE TABLE`. При создании нужно указать не только имя таблицы, но и ее полное определение, состоящее из определений отдельных полей. Таблица `cars` должна содержать поля:

- `model` — с названием модели автомобиля;
- `madein` — года ее выпуска;
- `reg_number` — государственного регистрационного номера;
- `color` — цвета.

Кроме того, нам понадобится столбец `id`, однозначно идентифицирующий запись об автомобиле — первичный ключ. Значения этого поля не должны повторяться и не могут быть пустыми, желательно, чтобы эти значения были целыми числами, увеличивающимися на единицу в каждой новой записи. Полностью команда создания таблицы `cars` будет выглядеть так:

```
mysql> CREATE TABLE  cars (
    -> model CHAR(50) NOT NULL,
    -> madein YEAR(4) NOT NULL,
```

```
    -> reg_number CHAR(12) NOT NULL,
    -> color CHAR(15) NOT NULL,
    -> id SMALLINT AUTO_INCREMENT,
    -> PRIMARY KEY(id)
    -> );
```

Атрибут `NOT NULL` означает, что все строки таблицы должны иметь значение в этом столбце. Если `NOT NULL` не указано, поле может быть пустым (`NULL`).

`PRIMARY KEY` после имени столбца определяет, что этот столбец является первичным ключом для таблицы. Данные в этом столбце должны быть уникальными.

`AUTO_INCREMENT` — атрибут, используемый для создания столбца с уникальными значениями. Если при вставке строк в таблицу оставлять такое поле пустым, MySQL автоматически генерирует уникальное значение идентификатора. Это значение будет на единицу больше максимального значения, уже существующего в столбце. В каждой таблице может быть не больше одного такого поля.

MySQL использует следующий алгоритм для инициализации счетчика для столбца `id`, имеющего атрибут `AUTO_INCREMENT`: после запуска сервера MySQL при первом запросе на добавление данных в таблицу `cars` сервер высчитывает максимальное значение в столбце `id`. Полученное значение увеличивается на единицу, заносится в новую запись и в счетчик. Если таблица была пуста, то счетчик устанавливается в единицу.

Таблица `drivers` содержит столбцы для хранения имени, отчества и фамилии водителя — `name`, `second_name`, `family_name`; даты его рождения `birth`, домашнего адреса `address` и даты поступления на работу `startdate`.

```
mysql> CREATE TABLE  drivers (
    -> name CHAR(30) NOT NULL,
    -> second_name CHAR(30),
    -> family_name CHAR(30) NOT NULL,
    -> birth DATE NOT NULL,
    -> address CHAR(150) NOT NULL DEFAULT 'unknown',
    -> startdate DATE    NOT NULL DEFAULT '2002-01-01',
    -> id SMALLINT AUTO_INCREMENT,
    -> PRIMARY KEY(id)
    -> );
```

Кроме этого, при создании таблицы для некоторых полей могут применяться дополнительные ключевые слова, уточняющие диапазон возможных значений. После целочисленного типа может указываться ключевое слово `UNSIGNED` для хранения чисел без знака.

После ключевого слова `DEFAULT` указывается значение по умолчанию для данного столбца. Например:

```
DEFAULT 'no description'
```

Таблица `timetable` состоит из полей с датой `use_on`, номером машины `car_number` и идентификатором водителя `driver_number`.

```
mysql> CREATE TABLE  timetable (
    -> use_on DATE DEFAULT NULL,
```

```
-> car_number SMALLINT,
-> driver_number SMALLINT,
-> id SMALLINT AUTO_INCREMENT,
-> PRIMARY KEY(id),
-> INDEX (car_number),
-> INDEX (driver_number),
-> FOREIGN KEY (car_number) REFERENCES cars (id),
-> FOREIGN KEY (driver_number) REFERENCES drivers (id)
-> );
```

MySQL поддерживает ссылочную целостность базы с помощью ограничения внешнего ключа `FOREIGN KEY`. Назначение этого ключа — проверять соответствие значений в столбцах родительского и внешнего ключей. Эти столбцы должны иметь одинаковый тип данных, и только те значения, которые встречаются в родительском ключе, могут использоваться во внешнем ключе. Родительский ключ должен содержать неповторяющиеся значения и не содержать значений `NULL`.

Последние два столбца таблицы `timetable` являются внешними ключами, и MySQL 5 требует, чтобы перед созданием внешних ключей на этих столбцах были созданы индексы.

Индекс формируется из значений одного или нескольких столбцов таблицы и позволяет находить нужную строку по заданному значению. Для ускорения запросов индексы обычно создаются на тех столбцах таблицы, которые часто используются в запросах.

После создания таблиц можно просмотреть их список и структуру. Список таблиц в базе можно просмотреть с помощью команды:

```
mysql> SHOW TABLES;
```

а структуру таблицы:

```
mysql> DESCRIBE table_name;
```

Здесь *`table_name`* — имя таблицы. Эта команда позволит посмотреть типы данных столбцов и дополнительные атрибуты, указанные при создании таблицы.

Схема базы данных `taxi` представлена на рис. 16.2.

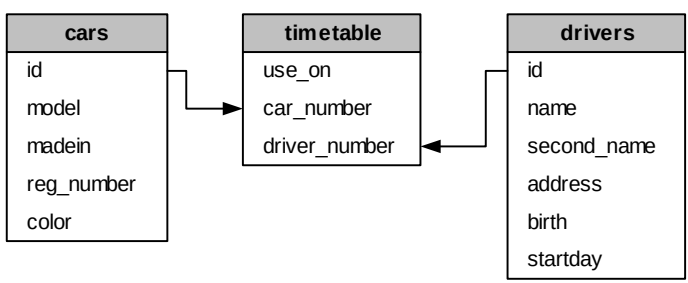


Рис. 16.2. Схема базы данных таксопарка

Запись данных в таблицы

Теперь необходимо внести данные в таблицы. Для внесения данных в базу можно использовать оператор `INSERT`. Самый простой вариант — внесение данных во все поля таблицы:

```
mysql> INSERT INTO cars VALUES ('Volga','2003','A786YC78','white','','1');
```

Аналогично можно добавлять данные только в избранные поля, при этом надо указать имена полей явно:

```
mysql> INSERT INTO cars (model, madein, reg_number, color) VALUES
    -> ('Volga','2002','A788YC78','yellow');
mysql> INSERT INTO cars (model, madein, reg_number, color) VALUES
    -> ('Volkswagen','2003','A789YC78','red');
mysql> INSERT INTO cars (model, madein, reg_number, color) VALUES
    -> ('Renault','2005','A790YC78','white');
```

Добавим еще несколько значений в наши таблицы:

```
mysql> INSERT INTO drivers VALUES (
    -> 'Stanislav', 'Petrovitch', 'Nikolaev',
    -> '1970-01-22', 'Nevsky prospect 11, flat 54', '2004-02-30', 1);
mysql> INSERT INTO drivers (name, second_name, family_name, birth,
    -> address, startdate)
    -> VALUES ('Fedor', 'Sergeevitch', 'Sentsov',
    -> '1960-11-02', 'Moskovsky prospect 92, flat 112', '2003-12-30');
mysql> INSERT INTO timetable VALUES ('2006-08-09', 3,1,1),
    -> ('2006-07-21', 2,2,2), ('2006-07-20', 2,2,3);
```

Последняя запись наверняка делается диспетчером автопарка каждый день с утра. Чтобы упростить его работу, можно использовать встроенную в MySQL функцию `curdate()`, которая возвращает текущую дату как раз в нужном формате:

```
mysql> INSERT INTO timetable VALUES (curdate(), 2,2,4);
```

Если вы попытаетесь оставить пустыми поля, отмеченные `NOT NULL` и не имеющие значения по умолчанию, то сервер MySQL выдаст сообщение об ошибке и не выполнит команду.

Кроме того, сервер MySQL 5 проверяет связи между таблицами при внесении записей. То есть вам не удастся внести в поле, являющееся внешним ключом, значение, отсутствующее в связанной таблице.

Утилита `mysql` может создавать таблицы и заполнять их данными и в пакетном режиме. Для этого необходимо создать текстовый сценарий с командами SQL, обычно такой сценарий сохраняют в файле с расширением sql. Затем необходимо запустить утилиту `mysql` в пакетном режиме и передать ей имя сценария. Рассмотрим эту процедуру подробнее в следующем разделе.

Клиентские утилиты

Интерфейс MySQL Command Line Client, с которым мы работали в предыдущем разделе, удобен, но не позволяет использовать функциональность утилиты `mysql` полностью. Кроме того, в состав MySQL входит еще ряд полезных утилит, с одной из которых, `mysqldump`, необходимо познакомиться даже начинающим.

Утилита командной строки *mysql*

Клиентская утилита `mysql` запускается в интерактивном режиме в DOS-окне, когда вы выбираете в меню пункт **MySQL | MySQL Command line client**. До MySQL 5 в комплекте поставки сервера не было такой возможности запуска `mysql`, и приходилось самим вызывать окно для работы с командной строкой. Как всегда в Windows, за удобство работы с готовым интерфейсом приходится платить непониманием сущности происходящего, при этом еще и лишаясь некоторых возможностей, благополучно использовавшихся при работе со старым интерфейсом. Поэтому, налюбовавшись на сервис MySQL Command Line Client из MySQL 5, вернемся в юность информационных технологий и откроем DOS-окно для ввода команд. В Windows это делается так: щелкаем по кнопке **Пуск**, выбираем пункт меню **Выполнить** и в появившемся окне **Запуск программы** набираем `cmd`. Появляется черное DOS-окно со строкой, содержащей имя текущего каталога, — это приглашение ввести команду. Курсор мигает в конце этой строки.

Наш сервер mysqld-nt.exe, клиентские утилиты `mysql`, `mysqldump` и `mysqlimport` находятся в том каталоге, в который мы установили сервер, а именно в C:\Program Files\MySQL\MySQL Server 5.0\bin. Поскольку при установке MySQL мы добавили путь до этого каталога в переменную `PATH`, то смело можем запускать утилиту сразу после открытия DOS-окна.

Утилиты предназначены для работы с сервером и при запуске потребуют указать имя пользователя, под которым мы хотим подключиться к серверу, и пароль. При конфигурировании сервера MySQL мы указали пароль для суперпользователя `root`, теперь снова пора им воспользоваться. Обычно для получения доступа к серверу MySQL необходимо сообщить клиентской программе и хост, с которым вы хотите соединиться. Например, клиент `mysql` можно запустить следующим образом:

```
>mysql -h host_name -u user_name -pyour_pas
```

Здесь ключ `-h` указывает на то, что дальше последует имя хоста, к которому мы хотим подключиться. По умолчанию подключение осуществляется к серверу localhost. Ключ `-u` означает, что за ним следует имя пользователя, от имени которого осуществляется соединение, а после ключа `-p` указывается пароль. Обратите внимание, что пробела перед паролем нет.

Формат вызова утилиты `mysql` в нашем случае будет таким:

```
>mysql -u root -p
```

После ключа `-p` не указан пароль, мы знаем, что он нужен, но не хотим давать его в командной строке, а дождемся приглашения и введем так, что его у нас из-за спины никто не прочитает. Приглашение появится, если сервер MySQL запущен, а если сервер не доступен, будет выведено сообщение о невозможности к нему

подключиться. Если ключи набраны неверно, например, пропущены пробелы, вы получите справку по утилите. После ввода пароля мы должны увидеть приветствие и приглашение командной строки уже самой утилиты `mysql` (рис. 16.3).

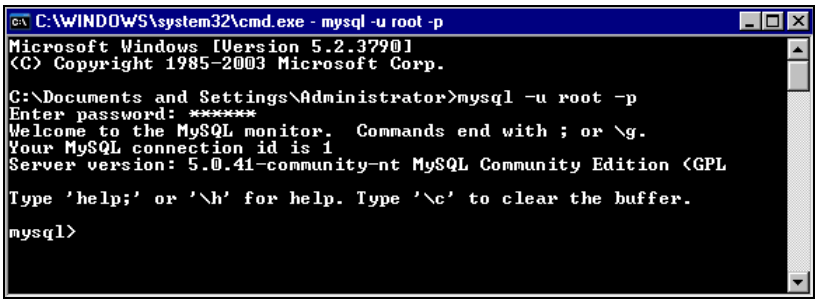


Рис. 16.3. Запуск утилиты `mysql`

Похоже на MySQL Command Line Client? Что и требовалось доказать.

Получив приглашение утилиты `mysql`, можно давать команды MySQL для создания баз и таблиц. Завершение сеанса работы происходит по команде:

```
mysql> quit
```

После этого в DOS-окне снова появится приглашение операционной системы. Учтите, что и при работе с `mysql`, и с операционной системой существует возможность повтора ранее данных команд. Просмотреть историю команд можно нажатием клавиш со стрелками вверх и вниз. При нажатии клавиши после приглашения командной строки появляются команды, запустить их можно, нажав клавишу <Enter>.

Утилита `mysql` может работать в двух режимах: интерактивном, который мы только что инициировали, и пакетном.

Для работы с утилитой в пакетном режиме следует написать сценарий, состоящий из последовательности команд SQL, которые предстоит исполнить для создания базы и таблиц, входящих в базу. Сценарий можно сохранить в тот же каталог bin сервера MySQL, в котором находится утилита `mysql`. Принято хранить сценарии в файлах с расширением sql. Кроме уже рассмотренных команд в скрипте надо предусмотреть возможность того, что в сценарии создания таблицы возможны ошибки. При исправлении ошибок в сценарии вы запускаете его несколько раз, но уже при повторном запуске получаете сообщение о том, что нельзя создать уже существующую таблицу. Значит, надо предварять создание таблицы командой:

```
mysql> DROP TABLE IF EXISTS cars;
```

и удалить таблицу, если она уже существует.

Тогда сценарий создания и заполнения таблицы `cars` будет выглядеть так:

```
USE taxi;
DROP TABLE IF EXISTS cars;
CREATE TABLE  cars (
  model CHAR(50) NOT NULL,
  madein YEAR(4) NOT NULL,
  reg_number CHAR(12) NOT NULL,
```

```
  color CHAR(15) NOT NULL,
  id SMALLINT AUTO_INCREMENT PRIMARY KEY
);

INSERT INTO cars (model, madein, reg_number, color) VALUES
 ('Volga','2002','A788YC78','yellow'),
 ('Volkswagen','2003','A789YC78','red'),
 ('Renault','2005','A790YC78','white');
```

Скрипт создается в любом текстовом редакторе в кодировке Windows-1251. В этой кодировке данные и записываются. И хотя в окне утилиты `mysql` русские буквы будут нечитаемыми, но скрипты PHP передадут символы правильно.

При использовании в пакетном режиме сценарии с запросами для создания базы надо скопировать в каталог bin и передать их на вход утилите `mysql` при запуске:

```
>mysql -u root -p < script.sql
```

Здесь script.sql — это сценарий, содержащий команды SQL для создания таблиц, записи в них данных и т. п.

Утилита *mysqldump*

Утилита `mysqldump` позволяет получить дамп ("моментальный снимок") содержимого базы данных или совокупности баз для создания резервной копии или пересылки данных на другой SQL-сервер. Дамп будет содержать набор команд SQL для создания и заполнения таблиц. Если вы из командной строки дадите такую команду:

```
>mysqldump имя_базы
```

то выведете на экран весь этот набор команд. Чтобы сохранить дамп в файле, дайте команду перенаправить вывод не на экран, а в файл backup-file.sql:

```
>mysqldump имя_базы > backup-file.sql
```

Если не указывать имена таблиц или использовать параметр `--databases` или `--all-databases`, то будет получен дамп базы данных в целом или всех баз данных сервера соответственно.

Перенеся файл дампа на другой сервер MySQL, можно этот файл передать на вход утилиты MySQL для создания таблиц в базах данных.

Команда:

```
>mysqldump -u root -p taxi cars > cars.sql
```

приведет к созданию в текущем каталоге текстового файла cities.sql следующего содержания (листинг 16.1). Выглядеть в DOS-окне это должно так, как показано на рис. 16.4.

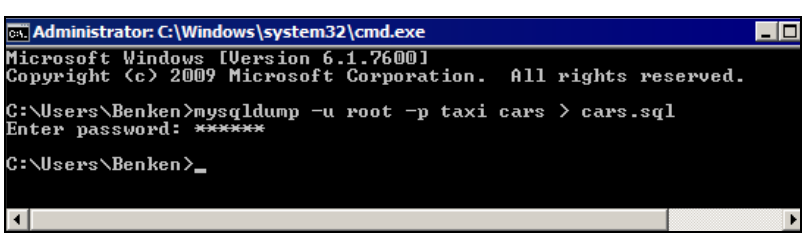


Рис. 16.4. Утилита `mysqldump`

Листинг 16.1. Дамп таблицы `cars`

```
-- MySQL dump 10.13  Distrib 5.1.47, for Win32 (ia32)
--
-- Host: localhost    Database: taxi
-- ------------------------------------------------------
-- Server version    5.1.47-community

-- Table structure for table 'cars'
DROP TABLE IF EXISTS 'cars';
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE 'cars' (
  'model' char(50) NOT NULL,
  'madein' year(4) NOT NULL,
  'reg_number' char(12) NOT NULL,
  'color' char(15) NOT NULL,
  'id' smallint(6) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY ('id')
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

-- Dumping data for table 'cars'
LOCK TABLES 'cars' WRITE;
/*!40000 ALTER TABLE 'cars' DISABLE KEYS */;
INSERT INTO 'cars' VALUES
('Volga',2002,'A788YC78','yellow',1),('Volga',2001,'A418YC78','red',2),('Volga
',2003,'A799YC78','white',3),('Volga',2004,'A218YC78','yellow',4),('Volga',200
1,'A798YC78','yellow',5),('Volkswagen',2003,'A411YC78','red',6),('Renault',200
5,'A790YC78','white',7);
/*!40000 ALTER TABLE 'cars' ENABLE KEYS */;
UNLOCK TABLES;
/* Здесь пропущены указанные в дампе установки для различных
   системных параметров */
```

Обратите внимание, что команда `CREATE TABLE` заканчивается описанием параметров таблицы:

```
ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Первый из них, `ENGINE`, указывает тип таблиц, к которому относится наша таблица `cities`. Тип `InnoDB` применяется в сервере MySQL 5 под Windows по умолчанию. Если создается таблица такого типа, то сервер производит проверку данных, вносимых в столбцы, помеченные как внешний ключ. Проверяется, соответствует ли тип данных этого столбца типу столбца другой таблицы, с которой осуществляется связь. Кроме того, тип `InnoDB` поддерживает транзакции, описание которых будет дано в *главе 18*.

Утилита *mysqlimport*

Утилита `mysqlimport` импортирует текстовые записи в таблицы. На основании имени файла утилита определяет имя таблицы, в которую импортируются данные.

```
>mysqlimport параметры имя_базы_данных имя_текстового_файла
```

Для каждого текстового файла, указанного в командной строке, `mysqlimport` удаляет расширение в имени файла и использует его, чтобы определить, в какую таблицу занести содержимое. Например, файлы с именами patient.txt, patient.text и patient должны быть все занесены в таблицу с именем `patient`.

Например, для того чтобы импортировать данные из файла new.txt (одноименного с таблицей) в базу данных `our_base`, надо дать следующую команду, а затем ввести пароль:

```
>mysqlimport –u root –p –local our_base new.txt
password:******
```

Каждая строка в файле образует запись в таблице и должна содержать значения полей, разделенные запятыми. Конец записи распознается по символу конца строки текстового файла.

Графический интерфейс *phpMyAdmin*

Помните, что утилиты командной строки — это средства, которые вас никогда не подведут: они точно соответствуют версии сервера MySQL и потому корректно выполнят любой запрос. Они не выводят на экран лишней информации и непосредственно отражают то, что происходит на сервере.

Но, возможно, вы нашли работу с этими утилитами не слишком удобной, слишком слабо автоматизированной. Для обучения это прекрасно, ведь чем меньше автоматизации, тем больше приходится думать самому. Но если вы хотите использовать какое-либо средство, предлагающее графический интерфейс для работы с базами данных, то можно предложить `phpMyAdmin`.

Это приложение представляет собой набор сценариев, написанных на PHP, и, следовательно, требует установки с подключенным расширением для работы с MySQL. Мы с вами при установке модуля PHP подключили расширение mysqli, чем теперь и воспользуемся.

Вы можете загрузить `phpMyAdmin` с сайта **http://sourceforge.net**, а документацию на русском языке найти на сайте **http://php-myadmin.ru** или взять с приложенного к настоящей книге компакт-диска.

Распакуйте архив в каталог htdocs и настройте конфигурационный массив `$cfg`, который должен находиться в файле config.inc.php в том же каталоге, где находится индексный файл этого интерфейса. Сначала необходимо указать, что при подключении интерфейса к серверу для аутентификации клиента должны использоваться имя и пароль пользователя из этого файла, что задается значением элемента массива с ключом `'auth_type'`. Интерфейс может работать с несколькими серверами MySQL, потому в качестве первого индекса массива `$cfg` указывается номер сервера `i`, но нам достаточно и одного сервера, поэтому просто укажем по порядку

имя хоста, к которому будем подключаться, имя пользователя и пароль. При подключении сервер будет проверять права доступа этого пользователя по своим системным таблицам. Наконец, следует указать, какое расширение для MySQL вы собираетесь использовать. Получиться должно примерно следующее:

```
$cfg['Servers'][$i]['auth_type'] = 'config';
$cfg['Servers'][$i]['host'] = 'localhost';
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = 'secret';
$cfg['Servers'][$i]['extension'] = 'mysqli';
```

Если вам удалось настроить `phpMyAdmin`, то при обращении к индексной странице этого интерфейса вы должны увидеть в браузере то, что представлено на рис. 16.5.

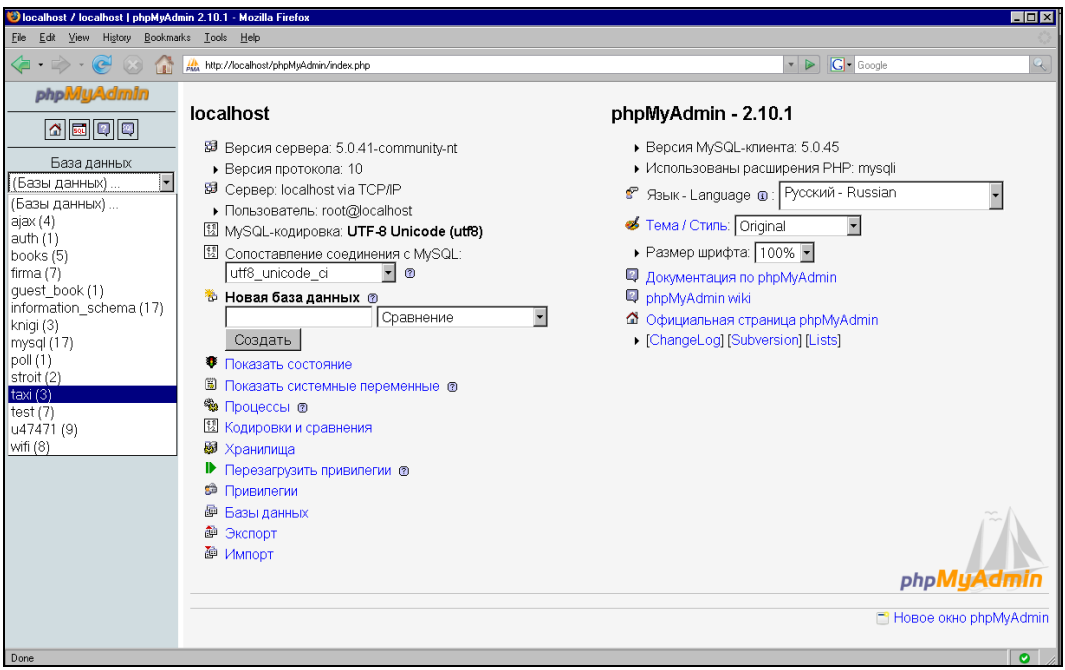


Рис. 16.5. Интерфейс `phpMyAdmin`

Интерфейс достаточно прост, чтобы читатель мог самостоятельно пробраться сквозь толщу пояснений на кнопках, подписях к полям и т. п. в процессе создания и модификации баз данных. Только обратите внимание, что при создании баз, таблиц и столбцов вас все время будут просить указать вариант сравнения данных. Так перевели здесь на русский язык слово collation. Вполне разумно выбрать cp1251_general_ci, что означает кириллическую кодировку Windows, суффикс ci означает, что упорядочивание строк по алфавиту будет производиться без учета регистра символов (от англ. *case insensitive*, нечувствительный к регистру). Еще более перспективный

вариант — выбор сравнения utf8_unicode_ci. Хотя в этом случае надо отдавать себе отчет в том, что строки в кодировке UTF8 занимают в два раза больше места, чем строки в однобайтной кодировке cp1251. На рис. 16.6 представлен вид одной из обсуждавшихся таблиц с указанием варианта сравнения данных.

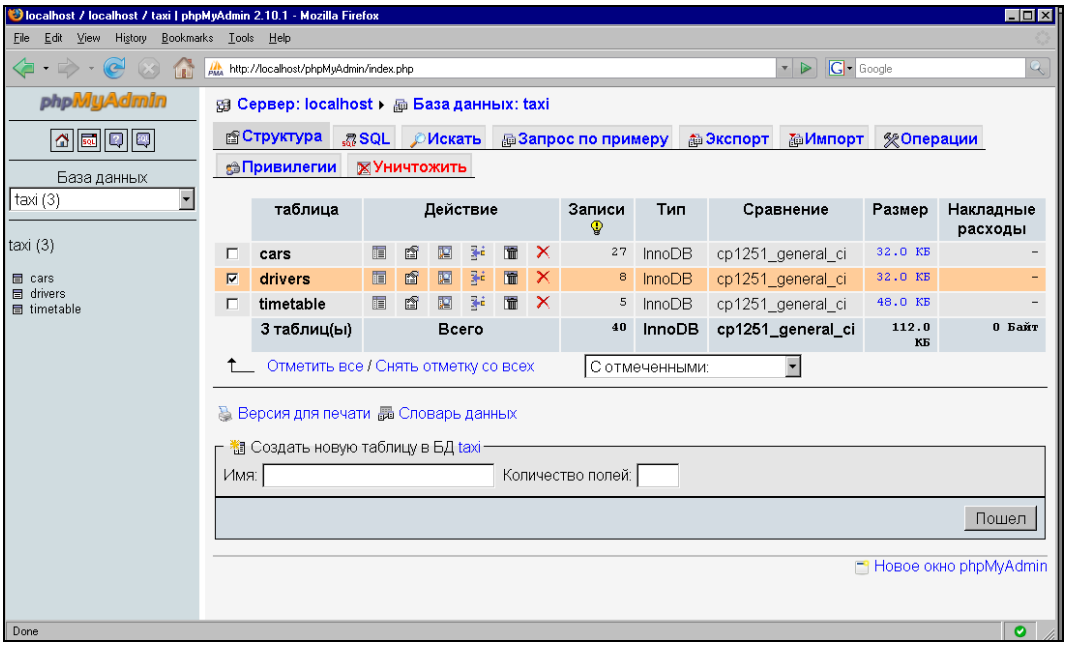


Рис. 16.6. Таблица `cars` в `phpMyAdmin`

Глава 17

Запросы к базе данных

На языке баз данных команды, адресованные базе, называются *инструкциями* либо *запросами*. Запросом считается такая инструкция, которая возвращает информацию из таблиц в память. Эта информация обычно посылается непосредственно на экран.

Запросы часто отправляются с помощью команды `SELECT`, которая дает инструкцию базе данных извлечь информацию из одной или нескольких таблиц.

Вначале необходимо соединиться с сервером MySQL и выбрать базу данных.

Для этого запустим клиентскую программу MySQL Command Line Client, а затем дадим команду выбора базы данных:

```
mysql> USE taxi
```

Команда *SELECT*

Запросим данные о модели и цвете автомобилей автопарка:

```
mysql> SELECT model, color FROM cars;
```

Результат представлен на рис. 17.1.

`SELECT` — ключевое слово, которое сообщает базе данных, что эта команда — запрос. Все запросы начинаются этим словом, за которым следует пробел.

`model, color` — список столбцов, которые выбираются из таблицы и помещаются в результирующую выборку в памяти.

`FROM` — ключевое слово, которое должно быть в каждом запросе. За ним должен идти пробел, а затем имя таблицы, используемой в качестве источника информации.

Если данных в таблице немного, то можно запросить их все:

```
mysql> SELECT * FROM table_name;
```

Здесь символ звездочки означает, что следует выбрать значения из всех столбцов. Но чаще запрашивают значения отдельных полей, поскольку обработка такого запроса требует меньших ресурсов.

Выясним, какие водители работали за то время, пока ведется учет с помощью базы. Нам нужен только список водителей из таблицы `timetable`, без повторов.

`DISTINCT` — параметр в запросе `SELECT`, дающий возможность исключить повторяющиеся данные из результатов запроса.

```
mysql> SELECT DISTINCT driver_number FROM timetable;
```

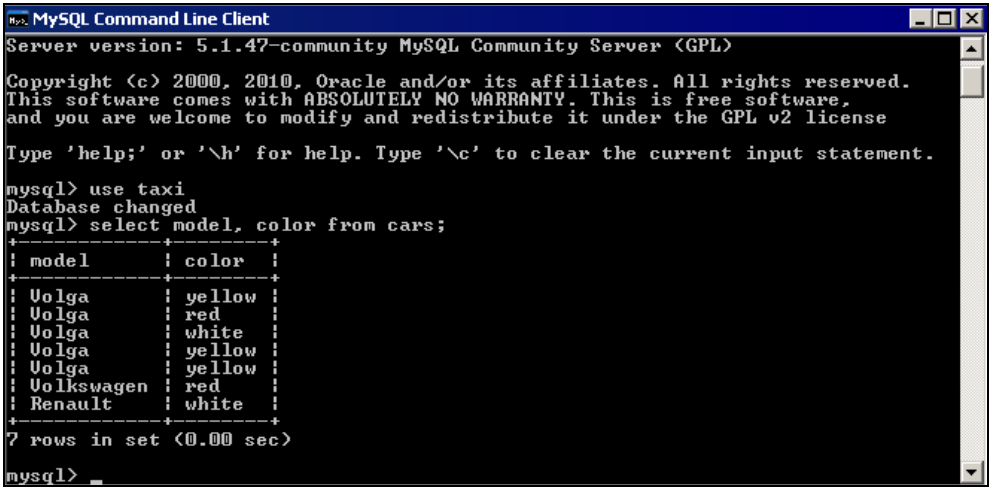


Рис. 17.1. Запрос к базе с помощью команды `SELECT`

Результат запроса при условии, что были выполнены описанные ранее действия, будет таким:

```
+---------------+
| driver_number |
+---------------+
|       1       |
|       2       |
+---------------+
```

Результат получился действительно без повторений, но не слишком понятный — указаны номера, а не фамилии водителей. Но фамилии находятся в другой таблице. Как выбрать данные из двух и более таблиц, мы узнаем, рассмотрев способы объединения таблиц.

Запросы с указанием критерия отбора данных

Таблицы бывают очень большими, а нас могут интересовать только определенные строки. SQL дает возможность устанавливать критерии для отбора нужных нам строк. Для этого используется предложение `WHERE`. Операции, допустимые в предложении `WHERE`, даны в табл. 17.1.

***Таблица 17.1.** Знаки операции сравнения для конструкции `WHERE`*

Оператор	Название	Описание
=	Равенство	Проверяет, являются ли два значения равными
>	Больше	Проверяет, больше ли одно значение другого
<	Меньше	Проверяет, меньше ли одно значение другого

Таблица 17.1 *(окончание)*

Оператор	Название	Описание
>=	Больше или равно	Проверяет, больше или равно одно значение по отношению к другому
<=	Меньше или равно	Проверяет, меньше или равно одно значение по отношению к другому
!= или <>	Не равно	Проверяет, не равны ли два значения
IS NOT NULL		Проверяет, имеет ли поле значение
IS NULL		Проверяет, не имеет ли поле значение
BETWEEN	Величина между	Проверяет, значение больше или равно минимальному и меньше или равно максимальному
IN	Значение содержится	Проверяет, содержится ли значение в определенном множестве
NOT IN	Значение не содержится	Проверяет, не содержится ли значение в определенном множестве
LIKE	Соответствие	Проверяет, отвечает ли значение образцу, используя простые механизмы соответствия SQL
NOT LIKE	Соответствие	Проверяет, не соответствует ли значение образцу
REGEXP	Регулярное выражение	Проверяет, соответствует ли значение регулярному выражению

Узнаем цвет всех автомобилей "Волга":

```
mysql> SELECT color FROM cars WHERE model = 'Volga';
```

Допустим, нам надо узнать, когда выходил на работу водитель с учетным номером 1:

```
mysql> SELECT use_on FROM timetable WHERE driver_number = 1;
```

Можно проверить несколько критериев сразу, объединяя их операциями `AND` или `OR`. Определим номера всех автомобилей "Волга" желтого цвета:

```
mysql> SELECT reg_number FROM cars
    -> WHERE model = 'Volga' AND color = 'yellow';
```

Оператор `IN` определяет множество значений, которому данное значение может принадлежать или не принадлежать. Чтобы найти номерные знаки всех автомобилей "Волга" и "Volkswagen", дайте такую команду:

```
mysql> SELECT reg_number FROM cars
    -> WHERE model IN ('Volga', 'Volkswagen');
```

Оператор `BETWEEN` задает границы, в которые должно попасть значение, чтобы условие запроса выполнялось. Вы должны ввести ключевое слово `BETWEEN` с начальным значением, затем ключевое слово `AND` и конечное значение. Выясним, кто из водителей устроился на работу в период с 1 января 2001 г. по 1 января 2004 г.:

```
mysql> SELECT family_name FROM drivers
    -> WHERE startdate BETWEEN '2001-01-01' AND '2004-01-01';
```

Оператор LIKE использует простой механизм соответствия SQL. Образец может состоять из обычного текста и знака процента (%) для указания совпадения с любым количеством символов. В MySQL соответствия не чувствительны к регистру. Например, шаблон 'Fed%' соответствует любой строке, которая начинается с 'Fed':

```
mysql> SELECT * FROM drivers WHERE name LIKE 'Fed%';
```

Ключевое слово REGEXP служит для указания регулярных выражений. MySQL использует регулярные выражения в стиле POSIX.

Найдем данные об автомобиле, в номере которого встречается число 788:

```
mysql> SELECT * FROM cars WHERE reg_number = '[A-Z]+788[A-Z0-9]+';
```

Группировка данных и агрегатные функции

Группировка и агрегирование данных совершаются с помощью функций, определенных в MySQL. Описание некоторых агрегатных функций дано в табл. 17.2.

Таблица 17.2. *Агрегатные функции MySQL*

Название	Описание
AVG(*столбец*)	Средняя величина значений в определенном столбце
COUNT(*элементы*)	При указании столбца выдается число ненулевых значений в этом столбце. Если перед именем столбца указать DISTINCT, то выдается количество уникальных значений
MIN(*столбец*)	Минимальное значение в столбце
MAX(*столбец*)	Максимальное значение в столбце
SUM(*столбец*)	Сумма значений в столбце

Агрегатные функции указываются подобно именам полей в запросе SELECT, но сами они используют имена полей в качестве аргументов. Функции SUM и AVG могут обрабатывать только числовые поля. С функциями COUNT, MAX и MIN могут использоваться и числовые, и символьные поля. Например, чтобы определить количество водителей в автопарке, можно запросить:

```
mysql> SELECT COUNT(id) FROM drivers;
```

Если надо узнать, сколько водителей работало за отчетное время, то подойдет команда:

```
mysql> SELECT COUNT(DISTINCT driver_number) FROM timetable;
```

Функция COUNT(*) посчитает число всех строк в таблице, но если указать COUNT(*field*), то при подсчете значений в поле *field*, в котором есть значения NULL, пустые поля учтены не будут.

Предложение GROUP BY позволяет определять подмножество значений в особом поле в терминах другого поля и применять функцию агрегата к подмножеству. Это дает вам возможность объединять поля и агрегатные функции в едином предложении SELECT.

Посчитаем, сколько автомобилей есть в таксопарке, сгруппировав данные по моделям:

```
mysql> SELECT model, COUNT(model) FROM cars GROUP BY model;
```

Каждая группа состоит из строк с одинаковым значением поля `model`. Результат запроса представлен на рис. 17.2.

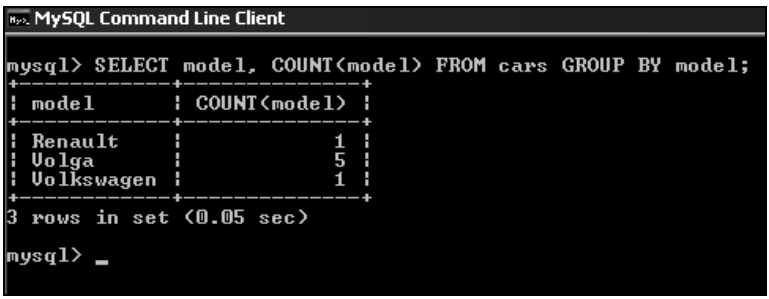


Рис. 17.2. Группировка результатов запроса

Таблицы — это неупорядоченные наборы данных, и выборка из них не обязательно появляется в какой-то определенной последовательности. Для извлечения данных в определенном порядке используется конструкция `ORDER BY`.

Упорядочим сведения о водителях по дате рождения:

```
mysql> SELECT * FROM drivers ORDER BY birth;
```

В полученной выборке список откроют старшие водители. Если требуется упорядочить данные по убыванию, следует использовать ключевое слово `DESC` (от англ. *descending* — по убыванию):

```
mysql> SELECT * FROM drivers ORDER BY birth DESC;
```

Конструкция `LIMIT` служит для указания, сколько строк результата следует отображать. Если надо найти двух самых пожилых водителей, то можно применить такую команду:

```
mysql> SELECT * FROM drivers ORDER BY birth LIMIT 2;
```

Можно использовать два параметра: номер строки, с которой следует начать, и количество строк.

```
mysql> SELECT family_name FROM drivers LIMIT 2,3;
```

Последний запрос можно интерпретировать так: "Выбрать фамилии водителей, в результате отобразить три строки, начиная со строки 2". Нумерация строк начинается с нуля.

Запросы к двум и более таблицам

Запрос, который обращен к нескольким записям одной таблицы или к нескольким таблицам одновременно, называется *объединением*. В теории баз данных определено несколько типов запросов, чаще других применяется внутреннее объединение (inner join), которое и будет описано в этом разделе.

Рассмотрим запрос, при котором надо выбрать данные из двух таблиц так, чтобы выбирались только те строки, значения которых по соединяемым одноименным столбцам совпадают. Например, нам требуется узнать фамилию водителя, который позавчера водил синюю машину. Для того чтобы получить все необходимые данные, нам потребуется запросить все три таблицы нашей базы, только тогда мы соберем все нужные сведения.

Нам придется оперировать именами столбцов из разных таблиц. Следует учесть, что полное имя столбца состоит из имени базы данных, таблицы и собственно имени столбца, например `taxi.cars.color`.

При работе с одной базой данных из запросов можно исключить упоминание имени этой базы. Если запросы адресуются к одной таблице, то имя таблицы также опускается. Но при объединении необходимо указывать полностью имена таблиц и столбцов.

Часто имена таблиц заменяют короткими псевдонимами (alias), указывая их в запросе после ключевого слова `FROM`. Псевдонимы для таблиц являются стандартной частью языка SQL и обычно состоят из одной буквы. Задать псевдоним можно с помощью ключевого слова `AS`:

```
mysql> SELECT color FROM cars AS c;
```

Псевдонимы удобно использовать в случае, когда имя результирующего столбца слишком длинное или выводятся данные из двух таблиц, в которых есть одноименные столбцы.

Между таблицами существуют связи по определенным столбцам. Столбец `cars.id` связывает таблицу `cars` с таблицей `timetable` через столбец `timetable.car_number`. Аналогично строится связь между таблицей `drivers` и `timetable`.

Объединение осуществляется по столбцам, образующим связь. При этом выбираются одинаковые значения полей, по которым осуществляется связь между таблицами:

```
timetable.car_number = cars.id
```

и

```
timetable.driver_number = drivers.id
```

Определим дату (данные из столбца `timetable.use_on`) и фамилию водителя (`drivers.family_name`), который водил машину определенного цвета (`cars.color`):

```
mysql> SELECT t.use_on, d.family_name, c.color
    -> FROM timetable AS t, cars AS c, drivers AS d
    -> WHERE t.car_number = c.id AND t.driver_number = d.id;
```

Выборка данных происходит из таблиц `timetable`, `cars` и `drivers`, для которых определены псевдонимы, сокращающие запись запроса и облегчающие его редактирование. Выбираются только те строки из таблиц `timetable` и `cars`, в которых указан одинаковый учетный номер автомобиля. А из таблиц `timetable` и `drivers` выбираются строки, в которых поля в столбце, содержащем учетный номер водителя, одинаковы.

Команды обновления и удаления данных в таблицах

Если необходимо изменить значение поля таблицы, то следует использовать команду UPDATE. В команде надо указать имя используемой таблицы и предложение SET, определяющее изменение, которое нужно сделать для некоторого столбца. Исправим дату выпуска автомобиля "Renault":

```
mysql> UPDATE cars SET madein = '2004' WHERE model = 'Renault';
```

Можно изменить значение всех полей столбца, например, при переоценке товаров. Увеличим цены на все товары на 10% с учетом инфляции:

```
mysql> UPDATE goods SET price = price * 1.1;
```

Удалить целые строки из таблицы можно с помощью команды DELETE. Удалим некоторые строки из таблицы cars в связи с продажей старых машин, изготовленных до 2006 г.:

```
mysql> DELETE FROM cars WHERE madein <= 2005;
```

Можно удалить все записи из таблицы:

```
mysql> DELETE FROM cars;
```

При этом счетчик автомобилей в столбце id не сбрасывается, и при вставке новых данных MySQL автоматически присвоит новой записи следующий после последнего удаленного номер.

Для того чтобы очистить таблицу и сбросить счетчик, можно применить команду:

```
mysql> TRUNCATE cars;
```

Удалить таблицу можно командой DROP TABLE:

```
mysql> DROP TABLE timetable;
```

При этом надо иметь права на удаление, а также учитывать, что удалить таблицу, имеющую первичный ключ, по которому осуществляется связь с другой таблицей, MySQL не позволит, пока не будет удалена таблица с соответствующим внешним ключом. Удалить можно и всю базу целиком:

```
mysql> DROP DATABASE taxi;
```

Изменение структуры таблицы

Оператор ALTER TABLE используется для изменения структуры таблицы. Преобразования, осуществляемые с помощью этого оператора, представлены в табл. 17.3.

Таблица 17.3. *Преобразования таблицы с помощью оператора* `ALTER TABLE`

Синтаксис	Описание
`ADD [COLUMN] column_description [FIRST | AFTER column]`	Добавить новый столбец в указанное место
`ADD INDEX [index] (column, ...)`	Добавить индекс в указанный столбец (столбцы) таблицы

Таблица 17.3 *(окончание)*

Синтаксис	Описание
`ADD PRIMARY KEY (column, ...)`	Сделать указанный столбец (столбцы) первичным ключом таблицы
`ALTER [COLUMN] column {SET DEFAULT value | DROP DEFAULT}`	Добавить или удалить значение по умолчанию определенного столбца
`CHANGE [COLUMN] column new_column_description`	Изменить столбец с именем `column` так, чтобы он получил указанное описание. Это можно использовать для изменения имени столбца, поскольку `column_description` включает в себя имя
`MODIFY [COLUMN] column_description`	Похоже на `CHANGE`. Используется для изменения типов столбцов, но не имен
`DROP [COLUMN] column`	Удалить указанный столбец
`DROP PRIMARY KEY`	Удалить первичный индекс (не столбец!)
`DROP INDEX index`	Удалить указанный индекс
`RENAME table tbl to new_tbl`	Переименовать таблицу

Допустим, мы решили добавить отсканированные фотографии водителей в базу, чтобы легче было печатать пропуска или другие документы. Ссылки на файлы с фотографиями можно поместить в новый столбец, добавив его в таблицу командой `ALTER TABLE`:

```
mysql> ALTER TABLE drivers ADD COLUMN photo CHAR(50);
```

В данном примере столбец добавляется в конец таблицы, но можно было бы добавить ключевое слово `FIRST`, тогда столбец стал бы первым, а можно указать `AFTER birth`, поместив его тем самым после указанного столбца.

Допустим, вы обнаружили, что поле для указания адреса оказалось мало для вводимых данных: последние символы обрезаются, т. к. адрес длиннее 150 выделенных для него символов. Изменим определение столбца:

```
mysql> ALTER TABLE drivers MODIFY address char(200);
```

Создание индексов

Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному значению путем последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет находить нужную строку по заданному значению.

Для оптимальной производительности запросов индексы обычно создаются на тех столбцах таблицы, которые часто используются в запросах. Для одной таблицы может быть создано несколько индексов.

Когда столбец помечается как ключ, создается индекс. Индекс хранит список значений ключа и указатели на записи, содержащие эти значения. При создании индекса таблица уже должна содержать столбцы, по которым он создается.

В базе таксопарка в таблице `cars` первичным ключом является учетный номер машины. Запросы же к базе могут строиться на поиске по государственному регистрационному номеру. Для того чтобы повысить скорость выполнения таких запросов, следует создать индекс по столбцу `reg_number`, в котором записан регистрационный номер:

```
mysql> CREATE INDEX nomer ON cars(reg_number);
```

При создании индекса на сервере формируется упорядоченный список значений этого поля, который MySQL использует при выполнении запросов. Если для создания индекса используется несколько полей, то значения второго поля упорядочиваются внутри первого, третьего поля — внутри второго и т. д.

Имя индекса должно быть уникальным в базе данных. После создания индекса MySQL использует его для поиска данных автоматически, т. е. в запросах не требуется указывать индекс каким-либо образом.

Вложенные запросы

Поддержка вложенных запросов появилась в MySQL относительно недавно, начиная с версии 4.1. Вложенный запрос создает результирующую таблицу, данные которой используются внешним запросом. Вложенный запрос помещается в скобках после выражения `WHERE`.

Создадим вложенный запрос с целью выяснить, на каком автомобиле работал определенный водитель 30 мая 2006 г.:

```
mysql> SELECT model, reg_number FROM cars
    -> WHERE id = (SELECT id
    -> FROM timetable
    -> WHERE date = '2006-05-30' AND driver_number = 1);
```

Сначала выполняется команда `SELECT` внутри скобок. MySQL получает результат вложенного запроса и использует его в условии в выражении `WHERE` во внешнем запросе `SELECT`. При этом результаты внутреннего запроса не отображаются. Конструкция запроса в приведенном примере подразумевает, что результат внутреннего запроса должен представлять собой единственное значение. В противном случае в результате выполнения этого запроса появится сообщение об ошибке: "Error 1241: Subquery returns more than 1 row". Вложенные запросы могут содержать команды `SELECT`, `INSERT`, `UPDATE` и `DELETE`.

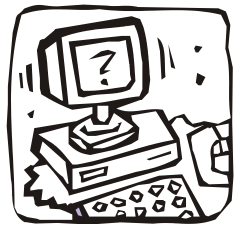
Табличные вложенные запросы

В случае, когда результатом вложенного запроса является не одно значение, а таблица, синтаксис запроса несколько меняется. Например, определим, на каких автомобилях работали водители 8 и 12 августа 2006 г.:

```
mysql> SELECT model, reg_number
    ->    FROM cars
```

```
->    WHERE id IN
->    (
->       (SELECT car_number FROM timetable
->       WHERE date = '2006-08-08'),
->       (SELECT car_number FROM timetable
->       WHERE date = '2006-08-12'),
->    );
```

Внешний запрос выбирает модель и регистрационный номер тех автомобилей, учетные номера которых указаны в таблице `timetable` в записях, относящихся к 8 и 12 августа 2006 г. Перечень условий для выражения `IN` пишется через запятую и помещается во внешние скобки. Вначале выполняются вложенные запросы, результаты которых формируют набор значений для выражения `IN`.



Глава 18

Обеспечение безопасности данных

Привилегии в MySQL

При подключении к серверу MySQL пользователь указывает имя и пароль, а сервер проверяет, имеет ли клиент право получить доступ к серверу, и в случае успеха наделяет пользователя привилегиями, исходя из данных о привилегиях конкретного пользователя. Эти данные хранятся в системной базе `mysql`.

Проверка пользователя осуществляется с помощью трех полей таблицы `user` (`host`, `user` и `password`) базы `mysql`. Сервер устанавливает соединение только в том случае, если находит в таблице `user` запись, в которой имя пользователя и пароль совпадают с введенными значениями. Поле `password` может быть пустым. Это не означает, что в данном случае подходит любой пароль. Если поле пароля пусто, пользователь должен быть подсоединен без указания какого-либо пароля.

Непустые значения в поле `password` представляют собой зашифрованные пароли. В MySQL пароли не хранятся в виде открытого текста, который может прочитать кто угодно. Пароль, который вводится пользователем при попытке подсоединения, шифруется (с помощью функции `PASSWORD()`). В дальнейшем зашифрованный пароль используется клиентом и сервером в процессе проверки его правильности.

При переходе к версии MySQL 4.1.3 произошло очень важное изменение. Дело в том, что зашифрованный пароль может быть разной длины. Длина зашифрованного пароля зависит от алгоритма шифрования, и чем больше длина, тем труднее отгадать пароль. Шифрацию пароля осуществляет сам сервер при создании новой учетной записи. Пользователь вводит пароль при соединении с сервером, клиентская программа его шифрует и отправляет на сервер в зашифрованном виде. Сервер не расшифровывает пароль, а сравнивает присланную ему зашифрованную строку с хранящейся у него строкой пароля. Алгоритм шифрования таков, что шифрация одинаковых слов всегда даст один и тот же результат.

В версии MySQL 4.0 длина шифра составляла 16 символов, пароль шифровали клиентская утилита `mysql` и другие клиентские утилиты, а также функции PHP для доступа к MySQL.

Начиная с MySQL 4.1.3, в реализации этого сервера под Windows был изменен алгоритм шифрации — длина шифра теперь составляет 41 символ. Вы можете убедиться в этом, выполнив команды:

```
mysql> USE mysql
mysql> SELECT password FROM user;
```

Вообще полезно посмотреть, что содержится в этой таблице. Поупражняйтесь в этом, давая команды SELECT для выборки различных полей. Сначала дайте команду:

```
mysql> SHOW TABLES;
```

Потом постарайтесь разобраться, какую информацию содержат таблицы базы mysql.

Что из всего этого следует? Во-первых, старые клиентские утилиты не годятся для работы с новым сервером MySQL 5. Поэтому работайте только с клиентскими утилитами из того пакета, из которого вы ставили сервер.

Во-вторых, для работы с сервером MySQL 4.1 и более новыми версиями, в частности 5.0, требуется новая библиотека функций PHP, это расширение mysqli — новое семейство функций для работы с сервером MySQL.

После установления соединения сервер приступает к выполнению второго этапа. Для каждого поступающего запроса сервер проверяет, имеется ли у вас достаточно привилегий для выполнения запроса. Информация о привилегиях находится в таблицах привилегий базы данных mysql — user, db, host, tables_priv или columns_priv.

Список привилегий представлен в табл. 18.1. Сервер MySQL считывает содержимое этих таблиц во время запуска и в случаях, когда изменения в привилегиях вступают в силу.

Привилегии SELECT, INSERT, UPDATE и DELETE позволяют выполнять операции над строками таблиц баз данных. Привилегия INDEX обеспечивает создание или уничтожение индексов. Привилегия ALTER позволяет использовать команду ALTER TABLE. Привилегии CREATE и DROP позволяют создавать новые или уничтожать существующие базы данных и таблицы. Привилегия GRANT позволяет вам предоставлять другим пользователям привилегии, которыми обладаете вы сами.

Таблица 18.1. *Привилегии MySQL*

Привилегия	**К чему применяется**
ALTER	Таблицы
DELETE	Таблицы
INDEX	Таблицы
INSERT	Таблицы
SELECT	Таблицы
UPDATE	Таблицы
CREATE	Базы данных, таблицы или индексы
DROP	Базы данных или таблицы
GRANT	Базы данных или таблицы
SHOW DATABASES	Администрирование сервера
SHUTDOWN	Администрирование сервера

Остальные привилегии используются для операций администрирования.

Команда SHUTDOWN завершает работу сервера.

При запуске сервера MySQL все таблицы назначения привилегий загружаются в память, и с этого момента привилегии вступают в силу.

Команды GRANT и REVOKE позволяют системным администраторам создавать пользователей MySQL, а также предоставлять права пользователям или лишать их прав на четырех уровнях:

- глобальный уровень. Глобальные привилегии применяются ко всем базам данных на указанном сервере. Эти привилегии хранятся в таблице mysql.user;
- уровень базы данных. Привилегии базы данных применяются ко всем таблицам указанной базы данных. Эти привилегии хранятся в таблицах mysql.db и mysql.host;
- уровень таблицы. Привилегии таблицы применяются ко всем столбцам указанной таблицы. Эти привилегии хранятся в таблице mysql.tables_priv;
- уровень столбца. Привилегии столбца применяются к отдельным столбцам указанной таблицы. Эти привилегии хранятся в таблице mysql.columns_priv.

Команда GRANT позволяет добавлять новых пользователей сервера:

```
mysql> GRANT SELECT, INSERT ON taxi.* TO Mike@localhost
    -> IDENTIFIED BY 'secret';
```

Эта команда дает права INSERT и SELECT на базу taxi (на все ее таблицы — это задается символом звездочки) пользователю Mike, который подключается локально (localhost), причем при подключении к серверу пользователь должен указать пароль 'secret'. Пароль хранится на сервере в зашифрованном виде. Дайте такую команду, а потом посмотрите, что изменилось в таблице user базы mysql.

Вот еще пример использования команды GRANT:

```
mysql> GRANT ALL PRIVILEGES ON *.* TO Anna@"%"
    -> IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
```

Здесь Anna получает все права (ALL PRIVILEGES) на все таблицы всех баз сервера (*.*), причем подключаться она может как локально, так и удаленно — через сеть (это указано символом % после Anna@). Кроме того, Anna может передавать свои права другим пользователям (WITH GRANT OPTION), т. е. сама может давать команду GRANT и создавать новых пользователей системы.

```
mysql> GRANT USAGE ON *.* TO dummy@localhost;
```

Пользователь dummy, созданный в этом примере, может подсоединяться к серверу без пароля, но только с локального компьютера. Привилегия USAGE означает, что пользователя в системе завели, но прав у него никаких нет. Такое может потребоваться, когда человек принят на работу, но обязанности его еще не определены. Предполагается, что относящиеся к базам данных привилегии будут назначены позже.

Команда SET может устанавливать различные опции, влияющие на работу сервера или клиента. Пароль для пользователя root также можно задать с помощью этой команды:

```
mysql> SET PASSWORD FOR root@localhost = PASSWORD('new_password');
```

Пользователи могут работать и непосредственно с таблицами назначения привилегий:

```
mysql> USE mysql;
mysql> UPDATE user SET Password = PASSWORD('new_password')
    -> WHERE user = 'root';
```

После этого следует дать команду, заставляющую сервер перечитать таблицы привилегий — только тогда они вступят в силу:

```
mysql> FLUSH PRIVILEGES;
```

Если пароль задается при помощи оператора `GRANT...IDENTIFIED BY`, нет необходимости использовать функцию `PASSWORD()`. Эта команда самостоятельно выполнит шифрацию пароля, поэтому пароль следует указывать в простом текстовом виде, например, так:

```
mysql> GRANT USAGE ON *.* TO jeffrey@"%" IDENTIFIED BY 'biscuit';
```

Изменения, которые вносятся в таблицы назначения привилегий при помощи команд `GRANT`, `REVOKE` или `SET PASSWORD`, учитываются сервером немедленно.

Примечание

При создании нового пользователя в MySQL 5 не удастся создать пользователя с пустым паролем — система не позволит сделать это из соображений безопасности.

Транзакции

На сайте могут работать одновременно несколько пользователей, отправляя запросы к одной и той же странице. Система должна обслуживать такие запросы одновременно — это называется *параллелизмом*. Легко представить себе, что может произойти с файлами данных, не имеющих механизма защиты, если два сотрудника фирмы начнут обновлять сведения об оплате или доставке товаров. Первый, более медлительный сотрудник откроет файл с данными, при этом данные будут скопированы во временную область хранения, и начнет вносить в эту копию данные о произведенной оплате. В то же время второй, более энергичный сотрудник, откроет файл с данными, быстро внесет в него данные о доставке товара и закроет файл. После того как первый сотрудник закончит работу и сохранит свою копию рабочего файла, данные о доставке будут потеряны. В операционных системах такая ситуация называется *гонками*. В *главе 8* мы видели, что при работе с файлами эта проблема решается блокировкой файла на время работы с ним.

В реляционных базах данных защита целостности данных реализована в виде блокировки таблиц или транзакций. Блокировка производится способом, весьма напоминающим блокировку файлов при работе с ними с помощью функций PHP.

Транзакция — это группа операций, которая может быть выполнена либо полностью успешно, с обеспечением целостности данных и независимо от параллельно идущих других транзакций, либо не выполнена вообще, и тогда она не должна произвести никакого эффекта. Транзакция похожа на беспосадочный перелет через океан тем, что она не может быть выполнена наполовину. Если видим, что не доле-

теть, то поворачиваем назад. Сервер изолирует одновременные потоки друг от друга, ограничивая их доступ к модифицируемым данным. Сервер пытается выполнять транзакции так, будто они происходят последовательно, а не параллельно.

В конце транзакции происходит либо ее отмена, либо подтверждение. Отмена транзакции называется *откатом* (rollback). Подтверждение транзакции называется *фиксацией* (commit). Фиксация транзакции происходит в специальных журнальных файлах, информация из которых считывается при сбоях и используется для восстановления состояния базы до сбоя.

В MySQL всех версий можно создавать базы данных на основе таблиц нескольких типов. Транзакции применимы лишь к некоторым типам таблиц.

В MySQL 3 по умолчанию создаются таблицы типа MYISAM, в которых обеспечение целостности данных при одновременных запросах происходит за счет блокировки таблиц целиком в случае необходимости.

В MySQL 5 по умолчанию создаются таблицы типа InnoDB (при работе в Windows), в которых реализован механизм транзакций. В этом вы можете убедиться, просмотрев дамп базы, сделанный с помощью утилиты `mysqldump`.

Чтобы создать таблицу типа InnoDB, в SQL-запросе на создание таблицы можно указать `ENGINE = InnoDB` или `TYPE = InnoDB`:

```
mysql> CREATE TABLE customers(a INT, b CHAR (20), INDEX (a))
    -> ENGINE = InnoDB;
mysql> CREATE TABLE customers(a INT, b CHAR (20), INDEX (a))
    -> TYPE = InnoDB;
```

По умолчанию каждый клиент соединяется с сервером MySQL в режиме `AUTOCOMMIT`, в котором автоматически фиксируется каждый одиночный SQL-запрос клиента. Для того чтобы использовать транзакции, состоящие из нескольких запросов, надо дать команду:

```
mysql> SET AUTOCOMMIT = 0;
```

а затем использовать команды `COMMIT` или `ROLLBACK` для фиксации или отмены транзакции. Два примера транзакций в листинге 18.1 продемонстрируют вам их фиксацию и отмену.

Листинг 18.1. Фиксация и отмена транзаций

```
mysql> CREATE TABLE cities(id INT, name CHAR (20), INDEX (id))
    -> ENGINE = InnoDB;
Query OK, 0 rows affected (0.00 sec)
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO cities VALUES (1, 'SPb');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> SET AUTOCOMMIT = 0;
```

```
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO cities VALUES (2, 'Paris');
Query OK, 1 row affected (0.00 sec)
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM cities;
+------+--------+
| id   |  name  |
+------+--------+
|   1  |   SPb  |
+------+--------+
1 row in set (0.00 sec)
mysql>
```

Вставка записи о Париже прошла успешно, в этом можно убедиться, запросив выборку из таблицы до отката. Но в результате отката эти данные не были зафиксированы в базе и не были сохранены, что и показывает последняя выборка. То есть мы можем посмотреть, какой эффект произведет то или иное действие, а потом отменить его — произвести откат.

Глава 19

Расширение *mysqli* для работы с базами данных

Ядро PHP 4 включало в себя функции для работы с MySQL, но в PHP 5 они вынесены в отдельную библиотеку, которую надо подключать в файле php.ini. Мы установили сервер MySQL версии 5. Расширение, предназначенное для работы с MySQL 5, называется `mysqli`. Его необходимо подключить на этапе установки модуля PHP (что мы и сделали в *главе2*). Если подключение было выполнено, то в файле php.ini вы найдете следующую директиву:

```
extension=php_mysqli.dll
```

Проверить, что расширение подключено, можно, найдя в выводе функции `phpinfo()` строки, говорящие о поддержке `mysqli`. Если же выяснится, что расширение не подключено, то есть два варианта поправить ситуацию. Первый — просто переставить модуль PHP с правильным выбором подключаемых библиотек. Второй, более экономный, но требующий более глубокого понимания сути дела, состоит в добавлении файла библиотеки php_mysqli.dll из архива php-5.3.5-win32-VC6-x86.zip в каталог ext, расположенный внутри каталога PHP. После добавления надо открыть конфигурационный файл php.ini и добавить в его конец строки:

```
[PHP_MYSQLI]
extension=php_mysqli.dll
```

Сохраните файл и перезапустите сервер Apache.

Рассмотрим функции из этого расширения и напишем сценарии для работы с MySQL. Порядок обращения к серверу MySQL в них должен остаться таким же, как и при работе с клиентским интерфейсом:

1. Соединение с сервером MySQL с указанием имени сервера, имени пользователя и пароля.
2. Выбор базы данных.
3. Создание запросов к базе данных.
4. Закрытие соединения с базой данных.

Соединение с сервером MySQL до версии 4.1 с использованием более старой PHP-библиотеки `mysql` могло быть двух типов:

- *непостоянное*, при котором программа каждый раз создает новое соединение с сервером, после окончания работы требуется отсоединение от сервера;
- *постоянное*, при установлении которого программа проверяет, нет ли уже созданного и свободного канала связи с MySQL, и если такой есть, то программа

использует это соединение. Если нет, то создает новое соединение. При завершении работы нет необходимости завершать постоянное соединение принудительно. Постоянные соединения не поддерживаются в библиотеке `mysqli`.

Еще одно важное обстоятельство работы с `mysqli` состоит в том, что это расширение предоставляет возможность применения как объектно-ориентированного подхода, так и процедурного стиля программирования.

Процедурный стиль создания скрипта для работы с MySQL

Подключение к серверу и выбор базы данных

Для установления соединения с сервером MySQL и выбора базы данных вызывается функция `mysqli_connect()`, принимающая в качестве параметров имя сервера, с которым надо соединиться, имя пользователя и его пароль и, наконец, имя базы данных. Эта функция действует аналогично командам запуска утилиты `mysql`, которой мы указывали те же параметры.

Функция `mysqli_connect()` при успехе возвращает идентификатор соединения, а при неудаче — `false`.

До сих пор, работая с сервером MySQL, мы использовали учетную запись суперпользователя `root` в первую очередь для того, чтобы познакомиться со структурой системных баз данных. Но ни один администратор сервера не позволит Web-мастеру использовать имя и пароль суперпользователя в сценариях PHP для подключения к рабочему серверу. Обычно Web-мастер узнает от администратора имя пользователя и пароль для своих сценариев, а часто и единственное имя базы данных, которое он может использовать. Последнее вовсе его не ограничивает — мы видели, что в базе можно создавать сколько угодно таблиц, и не обязательно, чтобы они были связаны друг с другом.

Итак, когда вы заключаете с хостером договор о размещении своего сайта с поддержкой PHP и MySQL, администратор сервера баз данных дает такие команды:

```
mysql> CREATE DATABASE db45;
mysql> GRANT ALL PRIVILEGES ON db45.* TO user45 identified by 'pass45';
```

Это означает, что для вас создана база данных и заведена учетная запись. Параметры соединения с сервером MySQL могут использоваться в большом количестве сценариев. С учетом возможных изменений этих параметров целесообразно создать сценарий, содержащий определения констант соединения, и подключать его с помощью инструкции `include()` в каждый сценарий работы с MySQL:

```
<?php
define("USERNAME", "user45");
define("PASSWORD", "pass45");
define("DBNAME", "db45");
define("SERVER", "localhost");
?>
```

Далее в сценариях будем использовать эти константы для указания соответствующих значений. Функция `mysqli_connect()` осуществляет соединение с сервером и выбор базы:

```
$link = mysqli_connect(SERVER, USERNAME, PASSWORD, DBNAME);
```

Как мы уже говорили, эта функция возвращает идентификатор соединения в случае успеха и `false` в случае неудачи. В последнем случае надо вывести сообщение об этом с помощью функции `mysqli_connect_error()`, возвращающей описание ошибки соединения:

```
if (!$link) {
   printf("Соединение установить не удалось : %s\n",
          mysqli_connect_error());
   exit();
}
```

Можно поменять базу (аналогично SQL-команде `USE`), вызвав функцию `mysqli_select_db()`, которой надо передать идентификатор соединения с сервером и имя базы данных:

```
mysqli_select_db($link, DBNAME);
```

Эта функция возвращает `true` или `false` в зависимости от того, удалось подключиться к базе или нет.

Запросы к базе данных

Теперь можно отправлять SQL-запросы к базе данных. Общий вид запросов, отправляемых на сервер с помощью функции `mysqli_query()`, такой:

```
$result = mysqli_query($link, 'SQL-запрос');
```

Запросим список моделей любого года выпуска и регистрационных номеров автомобилей из базы данных `taxi`:

```
$result=mysqli_query($link, 'SELECT model, madein, reg_number FROM cars');
```

Функция возвращает `true` в случае успешного завершения (при запросе `INSERT` или `UPDATE`, когда запрос не возвращает результирующую выборку) или `false` в случае возникновения ошибки. Кроме того, в результате запроса из базы могут быть получены данные, которые помещаются во временную таблицу в памяти сервера. Тогда функция вернет идентификатор этой таблицы. Переменная `$result` получит этот идентификатор в качестве значения.

Подсчитать количество строк в таблице, содержащей результаты запроса, можно с помощью функции `mysqli_num_rows()`, которой в качестве аргумента нужен только идентификатор результата выборки:

```
$rows = mysqli_num_rows($result);
```

Наш запрос к базе данных вернул временную результирующую таблицу из трех столбцов: `model`, `madein` и `reg_number`. В таблице может быть несколько записей. Можно считывать одну запись за другой из этой выборки и из каждой записи создавать массив с ключами от нуля до двух. Для этого предусмотрена функция `mysqli_fetch_row($result)`, которая принимает в качестве аргумента идентифика-

тор результата выборки. Функция выбирает одну строку из временной результирующей таблицы и возвращает численно-индексированный массив, каждый элемент которого содержит значение одного из столбцов. При этом указатель сдвигается на следующую запись в результирующей таблице.

```
while ($row = mysqli_fetch_row($result))
{
  printf("%s, %s, %s\n", $row[0], $row[1], $row[2]);
}
```

В случае численно-индексированного массива программисту придется самому запоминать, какой элемент массива содержит значение того или иного поля. Удобнее создать ассоциативный массив, ключи которого — имена столбцов в результирующей таблице. Придется создавать такой массив несколько раз — по одному для каждой строки результирующей выборки.

Функция `mysqli_fetch_assoc($result)` возвращает ассоциативный массив, ключами которого будут `model`, `madein` и `reg_number`.

Теперь нарисуем таблицу, в которую запишем данные из выборки:

```
<?php
echo '<table >\n<tr>\n';
echo '<th>Модель</th><th>Год выпуска</th><th>Рег. номер</th></tr>\n';
for ($i = 0; $i < $rows; $i++)
{
  $r = mysqli_fetch_assoc($result);
  echo '<tr><td>'.$r["model"].'</td>';
  echo '<td>'.$r["madein"].'</td>';
  echo '<td>'.$r["reg_number"].'</td></tr>';
}
echo '</table>';
```

В этом примере в цикле, повторяющемся столько раз, сколько записей в результирующей таблице, функция `mysqli_fetch_assoc($result)` считывает одну запись из таблицы и из ее полей формирует ассоциативный массив. Функция `mysqli_fetch_assoc()` возвращает этот массив в переменную `$r`, а потом мы распечатываем элементы массива `$r`.

Если результат запроса больше не потребуется, следует освободить память от выборки:

```
mysqli_free_result($result);
```

Для закрытия соединения следует вызвать функцию `mysqli_close($link)`, в качестве параметра принимающую идентификатор соединения с сервером.

Соберем теперь все вместе (листинг 19.1).

Листинг 19.1. Чтение данных из базы

```
<?php
define("USERNAME", "user45");
define("PASSWORD", "pass45");
```

```
define("DBNAME", "db45");
define("SERVER", "localhost");

$link = mysqli_connect(SERVER, USERNAME, PASSWORD, DBNAME);
if (!$link) {
   printf("Соединение установить не удалось: %s\n",
          mysqli_connect_error());
   exit();
}
$result = mysqli_query($link,
                       'SELECT model, madein, regnumber FROM cars');
if ($result)
{
  $rows = mysqli_num_rows($result);
  echo '<table >\n<tr>\n';
  echo '<th>Модель</th><th>Год выпуска</th><th>Рег. номер</th></tr>\n';
  for ($i = 0; $i < $rows; $i++)
  {
     $r = mysqli_fetch_assoc($result);
     echo '<tr><td>'. $r["model"].'</td>';
     echo '<td>'. $r["madein"] .'</td>';
     echo '<td>'. $r["reg_number"] .'</td></tr>';
  }
  echo '</table>';
  mysqli_free_result($result);
}
mysqli_close($link);
?>
```

Если надо записать новые данные в таблицу или внести иные изменения, может оказаться полезной функция `mysqli_affected_rows()`, возвращающая количество записей, из которых выбирались данные при последнем SQL-запросе. Эта функция посчитает строки таблицы, из которых происходила выборка данных при запросах `SELECT`, `INSERT`, `UPDATE`, `ALTER TABLE`, причем параметром ей служит идентификатор соединения, а вовсе не идентификатор результата запроса.

```
$result = mysqli_query($db, 'SELECT * FROM cities');
printf("Affected rows (SELECT): %d\n", mysqli_affected_rows($db));
```

Обратите внимание! Начиная с PHP версии 5.2.4, появилось важное изменение, касающееся запросов, содержащих функции агрегирования (`count()`, `sum()` и т. п.). В запросах к базе данных эти функции необходимо заключать в кавычки:

```
$query = "SELECT 'count(*)' FROM cars WHERE city = 3";
```

Объектный подход

Объектно-ориентированный подход позволяет уменьшить размер сценария и улучшить устойчивость кода к ошибкам. В расширении `mysqli` предопределены следующие классы: `mysqli` , `mysqli_result` и `mysqli_stmt`.

Класс *mysqli*

Класс `mysqli` определяет соединения между PHP и базой данных MySQL. Конструктор этого класса вызывается при создании объекта — экземпляра класса:

```
$mysqli = new mysqli(SERVER, USERNAME, PASSWORD, DBNAME);
```

В качестве параметров конструктору передаются описанные ранее константы. В результате создается объект `$mysqli` — соединение с сервером, причем база данных уже выбрана. В классе определено большое количество методов, например метод `query()`, выполняющий запрос к базе данных:

```
$mysqli->query("CREATE TABLE cities(id INT PRIMARY KEY, name CHAR(30))");
```

С учетом возможных ошибок этот метод используется так:

```
if ($mysqli->query("CREATE TABLE cities (id INT PRIMARY KEY,
    name CHAR(30)) ") = = true)
{
   printf("Таблица cities создана.\n");
}
```

Ряд методов и свойств этого класса предназначены для сбора информации о сервере. Определены, например, такие свойства класса `mysqli`:

- `affected_rows` — число строк базы, использованных в предыдущем SQL-запросе;
- `errno` — код ошибки последнего вызова функции;
- `field_count` — число столбцов в последней результирующей выборке;
- `server_info` — строка, содержащая информацию о версии сервера MySQL;
- `get_host_info` — информация о типе соединения с MySQL-сервером.

Например, следующий код при выполнении на сервере под управлением Windows:

```
printf("Host information: %s\n", $mysqli->host_info);
```

должен вывести такую строку:

```
Host information: localhost via TCP/IP
```

А свойство `server_info` сообщит вам версию вашего сервера, например:

```
5.0.15-nt
```

Как вы, наверное, уже догадались, определен и метод закрытия соединения `close()`:

```
$mysqli->close();
```

Класс *mysqli_result*

Если запрос возвращает результирующую выборку, то создается новый объект — экземпляр класса `mysqli_result`:

```
if ($result = $mysqli->query("SELECT name FROM cities LIMIT 10"))
   printf("результат запроса - %d строк.\n", $result->num_rows);
```

В приведенном примере используется и свойство `num_rows` класса `mysqli_result`, значение которого равно числу строк в результирующей выборке.

Метод `fetch_row()` позволяет получить результат в виде численно-индексированного массива (листинг 19.2).

Листинг 19.2. Объектно-ориентированный код запроса данных из базы

```
<?php
$query = "SELECT name, description FROM cities LIMIT 5";
if ($result = $mysqli->query($query))
{
   /* Вначале проверили, что запрос успешен.
      Теперь создадим массив $row, он будет состоять из двух элементов,
      т. к. каждая строка результата содержит два значения.
   */
   while ($row = $result->fetch_row())
   {
      printf ("%s (%s)\n", $row[0], $row[1]);
   }
?>
```

При каждом проходе цикла `while` массиву `$row` будут присваиваться новые значения, так что после выхода из цикла в нем останется только последний пятый набор значений.

Как вы уже видели, класс `mysqli_result` описывает объекты, являющиеся результатами запроса к базе данных. Его метод `fetch_array()` представляет строку результирующей выборки в виде ассоциативного или численно-индексированного массива. Тип массива определяется тем, какая из определенных в этом классе констант указывается в качестве параметра метода: `MYSQLI_ASSOC`, `MYSQLI_NUM` или `MYSQLI_BOTH`. Эти константы определены как константы класса — в соответствии с объектной моделью PHP 5.

Получить ассоциативный массив результатов запроса можно так:

```
$row = $result->fetch_array(MYSQLI_ASSOC);
printf("%s (%s)\n", $row["name"]);
```

Есть методы, которые возвращают результат запроса в виде объекта — `fetch_object()`, и в виде только ассоциативного массива — `fetch_assoc()`. Кроме того, есть методы, предоставляющие информацию о результатах запроса — число строк, имена столбцов и т. д.

В конце работы с запросом надо вызвать метод, очищающий память сервера от результатов запроса:

```
$result->close();
```

Класс *mysqli_stmt*

Класс `mysqli_stmt` предназначен для создания подготовленных выражений. Представьте себе, что на сайте нужно выводить результаты запросов к базе. Запросы обычно формируются сложные, требующие выборки данных из нескольких таблиц. Значения выбираемых полей разные, а структура запросов — одинаковая. Хочется сформировать запросы раз и навсегда, отправить их на сервер, а потом только отправлять запрашиваемые значения полей с указанием вставить эти значения в сформированные запросы. Вот это и есть суть того, что называется *подготовленными* запросами.

Есть для их создания и процедурный интерфейс, но едва ли он будет широко применяться.

Рассмотрим пример подготовленных запросов с заданными параметрами. Создадим Web-страницу, используемую сотрудниками для пополнения базы данных в случае, когда надо добавить информацию о некоторых странах в таблицу с именем `countries`. На такой странице сотрудник должен ввести название и описание страны. Допустим, что для этого есть форма, куда и вводятся нужные значения. Они передаются в сценарий, где становятся значениями переменных:

```
$name = 'Portugal';
$description = 'The country is near the Atlantic ocean';
```

Подключимся к серверу:

```
$mysqli = new mysqli(SERVER, USERNAME, PASSWORD, DBNAME);
if (mysqli_connect_errno()) {
    printf("Подключение невозможно: %s\n", mysqli_connect_error());
    exit();
}
```

Метод `prepare()` класса `mysqli` позволяет создать подготовленное выражение:

```
$stmt = $mysqli->prepare("INSERT INTO countries VALUES (?, ?)");
```

В команде `INSERT` вместо значений столбцов, добавляемых в таблицу, стоят знаки вопроса. Они являются шаблонами, которые при отправке запроса заполнятся реальными значениями, вопросительные знаки как клеточки в кроссвордах — их надо заполнить.

В результате работы метода `prepare()` мы получаем объект `$stmt` класса `mysqli_stmt`. Метод `bind_param()` этого класса позволяет указать имена переменных, чьи значения заполнят шаблоны:

```
$stmt->bind_param('ss', $name, $description);
```

Обратите внимание на то, что первым параметром метода `bind_param()` является короткая строка. Это строка формата, используемая для определения того, как объявленные параметры должны быть интерпретированы. В данном случае эта

строка указывает, что оба требуемых параметра должны быть строковыми. Допустимые типы, используемые в качестве первого параметра функции, перечислены в табл. 19.1.

Таблица 19.1. *Допустимые типы параметров*

Символ	Описание типа
`I`	Все `int`-типы
`d`	`double` и `float`
`s`	Строки
`b`	BLOB

Теперь надо выполнить подготовленное выражение:

```
$stmt->execute();
printf("%d Row inserted.\n", $stmt->affected_rows);
```

Закрытие выражения и соединения:

```
$stmt->close();
$mysqli->close();
```

В листинге 19.3 приведен этот сценарий целиком.

Листинг 19.3. Запрос с использованием подготовленного выражения

```
<?php
$mysqli = new mysqli('localhost', 'root', 'secret', 'firma');
if (mysqli_connect_errno()) {
    printf("Подключение невозможно: %s\n", mysqli_connect_error());
    exit();
}
$stmt = $mysqli->prepare("INSERT INTO countries VALUES (?, ?)");
$stmt->bind_param('ss', $name, $description);
$name = 'Portugal';
$description = 'The country is near the Atlantic ocean';
$stmt->execute();
printf("%d Row inserted.\n", $stmt->affected_rows);
$stmt->close();
$mysqli->close();
?>
```

Но запросы часто возвращают результат, к которому необходимо получить доступ, а для этого надо назначить результату имя. Если пользоваться подготовленными выражениями, то тут следует создавать так называемые подготовленные выражения с объявленными результатами. Сформировав с помощью метода `prepare()`

подготовленный запрос, мы запускаем его на исполнение методом `execute()` класса `mysqli_stmt`, а вот результаты, которые вернет нам этот запрос, должны представлять собой данные из столбцов, например, `name` и `description`. Назовем их `$col1` и `$col2` и вызовем метод `bind_result()`, способный привязать имена этих столбцов к нашему результату выборки.

```
if ($stmt = $mysqli->prepare("SELECT name, description
                              FROM countries ORDER BY name LIMIT 5"))
{
   $stmt->execute();
   $stmt->bind_result($col1, $col2);
   // Теперь применим метод fetch() для выборки результата
   while ($stmt->fetch())
   {
     printf("%s %s\n", $col1, $col2);
   }
}
/* Закрытие выражения и соединения */
$stmt->close();
$mysqli->close();
```

ЧАСТЬ III

РАЗРАБОТКА ПРИЛОЖЕНИЯ

Глава 20

Проектирование сайта электронной коммерции

В этой главе мы начинаем разрабатывать практическое приложение, реализованное с помощью объектно-ориентированного кода на PHP 5. Приложение использует базу данных на сервере MySQL 5.

Задача

Требуется создать сайт туристической фирмы "Магазин Путешествий", посетители которого смогут посмотреть предложения фирмы по разным странам, городам и отелям, выбрать вариант тура и дату поездки, оставить заказ на выбранный тур. При оформлении заказа должна подсчитываться его стоимость. На сайте также должна быть представлена информация о странах, городах и отелях, включающая в себя описания и фотографии.

Информация о сделанных заказах и списке заказчиков должна быть доступна администратору, он должен иметь возможность внести исправления в данные о заказах и заказчиках, добавить или изменить сведения о городах, отелях и турах.

Структура сайта

Общая структура сайта представлена на рис. 20.1. На рисунке показаны ссылки между сценариями в пользовательской части сайта, предназначенной для выбора тура и оформления заказа клиента. Сценарии, доступные из меню администрирования, позволяют постранично просмотреть списки заказов и данные о клиентах, а также добавить или изменить сведения о заказчиках, городах, странах и отелях.

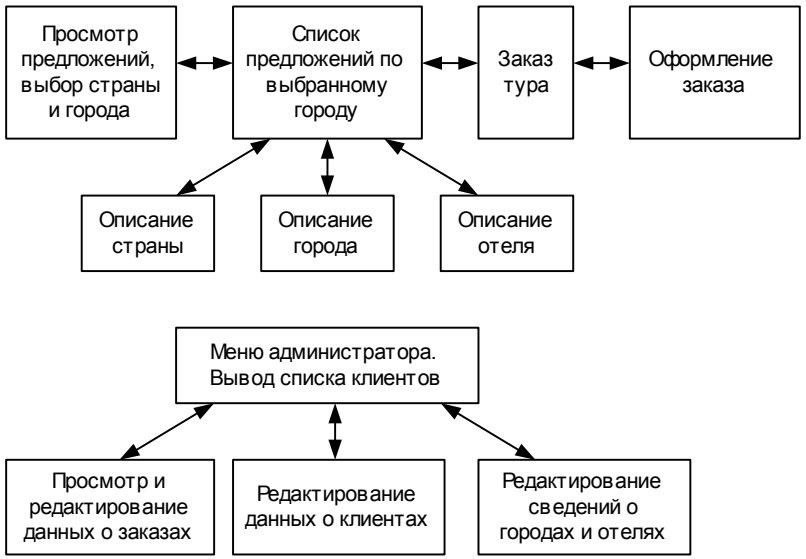


Рис. 20.1. Структура сайта туристической фирмы "Магазин Путешествий"

Файлы приложения электронной коммерции

Домашняя страница сайта генерируется через сценарий index.php. После загрузки сценария на странице выводится информация о предложениях по различным странам и форма для выбора страны. После выбора страны появляется форма с перечнем городов в выбранной стране. Вывод этого сценария показан на рис. 20.2.

После выбора города посетитель попадает на страницу select_tour.php, на которой выводится список предложений по выбранному городу. С этой же страницы можно попасть на страницы с описаниями стран, городов и отелей.

При такой организации страниц нарушается правило разработки сайта, которое гласит: на хорошо организованном сайте расстояние между любыми двумя страницами не должно превышать двух щелчков по ссылкам. В рассматриваемом примере можно добавить ссылки на описания городов и отелей на другие страницы, но в приведенных листингах такие ссылки удалены для облегчения понимания кода.

Если посетитель сайта решит заказать тур, то он переходит на страницу order.php, где может сообщить данные о себе и указать, какой тур он выбрал.

Объявления классов содержатся в отдельных сценариях, подключаемых с помощью функции `__autoload()`.

Интерфейс администратора имеет свою титульную страницу — admin.php, на ней выводится список клиентов и стоит ссылка на страницу редактирования данных базы.

Приложение содержит следующие сценарии (табл. 20.1).

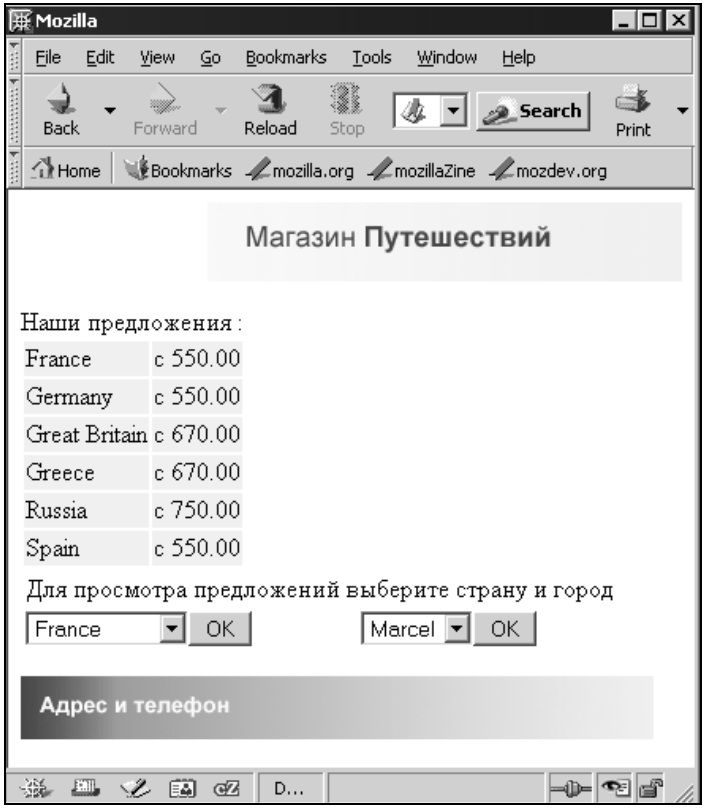


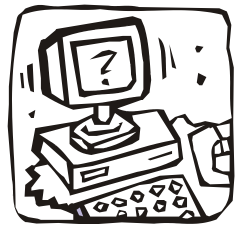
Рис. 20.2. Титульная страница сайта

Таблица 20.1. *Файлы приложения*

Название сценария	**Описание**
index.php	Домашняя страница сайта. Вывод самых дешевых предложений по всем странам, форма для выбора страны и города для поездки
select_tour.php	Вывод сведений о турах в выбранный город. Ссылки на страницы, содержащие описания стран, городов и отелей
order.php	Форма заказа тура
process_order.php	Запись данных о заказчике и сделанном заказе в базу
admin.php	Постраничный вывод информации о заказчиках и ссылки на страницы, на которых можно внести изменения в базу данных
admin_insert.php	Постраничный вывод данных о заказах, формы для внесения изменения в базу данных

Таблица 20.1 *(окончание)*

Название сценария	Описание
insert_country.php, insert_city.php, insert_hotel.php, edit_customer.php, edit_order.php	Редактирование данных базы в соответствующих таблицах на основании данных, указанных администратором
display_country.php, display_city.php, display_hotel.php	Вывод информации о странах, городах и отелях
class_baza.php	Родительский класс для всех остальных используемых классов. Содержит конструктор с функциями подключения к базе данных
class_hat_foot.php	Класс для подключения к страницам статического содержимого: заголовочных рисунков, сведений о фирме и т. п.
class_country.php, class_city.php, class_hotel.php	Классы для вывода информации из базы данных о странах, городах и отелях
class_customer.php, class_order.php	Классы для записи заказов



Глава 21

Реализация базы данных

Схема базы данных

Схема базы данных туристической фирмы представлена на рис. 21.1. База `firma` содержит таблицы с описаниями стран (`countries`), городов (`cities`) и отелей (`hotels`). Перечень туров находится в таблице `tours`, где хранится информация о стоимости поездки в тот или иной отель в зависимости от даты и типа размещения в отеле. Сведения о клиентах хранятся в таблице `customers`, таблица `orders` содержит сведения о сделанных заказах, а таблица `order_items` — детализированную информацию о заказах.

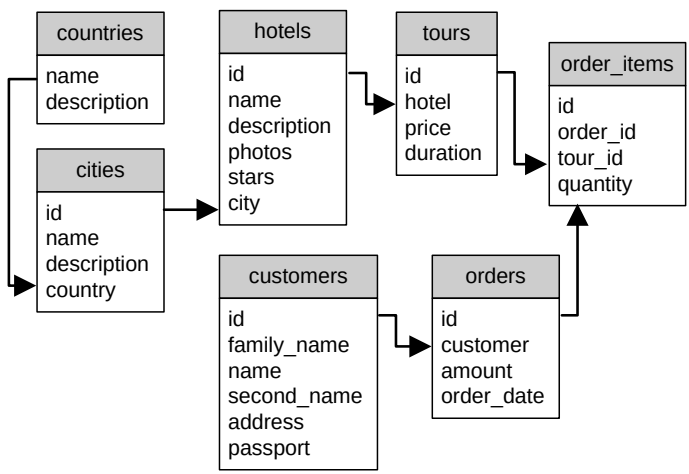


Рис. 21.1. Схема базы данных туристической фирмы

Создание и заполнение базы данных

Для создания базы данных необходимо запустить клиентскую утилиту `mysql` и выполнить команду создания базы:

```
mysql> CREATE DATABASE firma;
```

Сценарий, представленный в листинге 21.1, позволяет создать необходимые таблицы. Команды, его составляющие, можно набрать интерактивно в командной строке клиентской программы, а можно сохранить в файле с расширением sql, например firma.sql. Удобнее всего поместить этот файл в каталог bin сервера MySQL, а затем вызвать утилиту `mysql` в пакетном режиме и передать ей на исполнение сохраненный сценарий:

```
>mysql –u root –p <firma.sql
Enter password: ******
```

Если в сценарии ошибок нет, то программа создаст все таблицы, не выведя на экран никакого сообщения.

Листинг 21.1. Сценарий создания базы данных

```
USE firma;
/*
Перед созданием каждой таблицы не забудем удалить возможные результаты предыдущего творчества. Создадим таблицу с описаниями стран. Первичным ключом выбрано название страны – уникальное значение столбца.
*/
DROP TABLE IF EXISTS  countries;
CREATE TABLE  countries  (
   name  CHAR(40) NOT NULL,
   description  TEXT,
   PRIMARY KEY(name)
);
/*
Города. Столбец country является внешним ключом.
*/
DROP TABLE IF EXISTS  cities;
CREATE TABLE  cities(
   name  CHAR(30) NOT NULL,
   description  TEXT,
   id  SMALLINT NOT NULL AUTO_INCREMENT,
   country  CHAR(40) NOT NULL,
   PRIMARY KEY(id),
   INDEX (country),
   FOREIGN KEY (country) REFERENCES countries(name)
);
/*
Описание отелей. В столбце stars указывается категория отеля, единица измерения – звездочка (*).
*/
```

```
DROP TABLE IF EXISTS  hotels;
CREATE TABLE  hotels  (
   id  INT NOT NULL AUTO_INCREMENT,
   name  CHAR(50) NOT NULL,
   description  TEXT,
   photos  CHAR(30) DEFAULT NULL,
   stars  CHAR(5) NOT NULL,
   city SMALLINT NOT NULL,
   PRIMARY KEY(id),
   INDEX (city),
   FOREIGN KEY (city) REFERENCES cities(id)
);
/*
Описание туров.
startdate - дата начала поездки;
price - стоимость поездки;
duration - продолжительность поездки в днях.
*/
DROP TABLE IF EXISTS  tours;
CREATE TABLE  tours  (
   id  INT NOT NULL  AUTO_INCREMENT,
   hotel INT NOT NULL,
   startdate  DATE NOT NULL,
   price  DECIMAL(10,2) DEFAULT NULL,
   duration TINYINT NOT NULL,
   PRIMARY KEY(id),
   INDEX (hotel),
   FOREIGN KEY (hotel) REFERENCES hotels(id)
);
/*
Список заказчиков с указанием фамилии, имени и отчества, адреса и паспортных
данных.
*/
DROP TABLE IF EXISTS  customers;
CREATE TABLE  customers  (
   id  INT  NOT NULL AUTO_INCREMENT,
   family_name  CHAR(50) NOT NULL,
   name  CHAR(50) NOT NULL,
   second_name  CHAR(50) DEFAULT NULL,
   address  CHAR(100) NOT NULL,
   passport  TEXT,
```

```
   PRIMARY KEY(id)
);
/*
Список заказов с указанием суммы заказа в столбце price.
*/
DROP TABLE IF EXISTS  orders;
CREATE TABLE  orders  (
   id INT NOT NULL AUTO_INCREMENT,
   customer  INT NOT NULL,
   amount DECIMAL(12,2) NOT NULL,
   order_date  DATE NOT NULL,
   PRIMARY KEY(id),
   INDEX (customer),
   FOREIGN KEY (customer) REFERENCES customers(id)
);
/*
Описание заказанных туров.
*/
DROP TABLE IF EXISTS  order_items;
CREATE TABLE  order_items  (
   id  INT NOT NULL AUTO_INCREMENT,
   order_id  INT NOT NULL,
   tour_id  INT NOT NULL,
   quantity  SMALLINT,
   PRIMARY KEY(id),
   INDEX (order_id),
   INDEX (tour_id),
   FOREIGN KEY (order_id) REFERENCES orders(id),
   FOREIGN KEY (tour_id) REFERENCES tours(id)
);
```

Несколько туров может быть заказано в нескольких заказах — это отношение "многие-ко-многим" между таблицами `orders` и `tours`, которое нельзя реализовать в реляционной базе данных. Таблица `order_items` служит промежуточной таблицей между этими таблицами, позволяя преобразовать отношение "многие-ко-многим" между таблицами `orders` и `tours` в два отношения "один-ко-многим" между таблицами `order_items` и `orders` и между таблицами `order_items` и `tours`.

В листинге 21.2 приведен сокращенный сценарий заполнения базы данных туристической фирмы. Передать его на исполнение клиентской утилите `mysql` можно точно так же, как и предыдущий сценарий. Последующие обновления в базу будут вноситься интерактивно с помощью соответствующих HTML-форм и сценариев PHP.

Листинг 21.2. Запись данных в таблицы

```
USE firma;
INSERT INTO  countries(name, description)  VALUES
  ('Russia', 'country description'),
  ('France', 'country description');

INSERT INTO  cities(name, country)  VALUES
  ('Moscow', 'Russia'), ('Sankt-Petersburg', 'Russia'),
  ('Paris', 'France'), ('Marcel', 'France');

INSERT INTO  hotels(name, stars, city)  VALUES
  ('Europe', '5', 2), ('Astoria', '4', 2);

INSERT INTO tours(startdate, hotel, price, duration)  VALUES
  ('2011-09-01', 1, '35000.00', 5);
```

Примеры запросов к базе данных

Рассмотрим, какие запросы к базе потребуются для того, чтобы получить информацию о турах, отелях и городах, а также чтобы записать заказы клиентов.

Для начала надо будет представить список стран, а когда пользователь выберет страну, то представить список городов этой страны.

Первый запрос очень прост:

```
mysql> SELECT name FROM countries;
```

Результат этого запроса надо поместить в тег `<select>` формы на домашней странице сайта. Пользователь выбирает страну, а серверный сценарий получает это значение в качестве значения переменной и может использовать его для запроса списка городов:

```
mysql> SELECT name, id FROM cities WHERE country = "Great Britain";
```

Список городов выбранной страны формирует второй тег `<select>` — на этот раз со списком городов. Теперь пользователь выбирает город, передает это значение через форму, а мы получаем возможность вывести предложения по отелям этого города. Поле `id` даст нам небольшой выигрыш в простоте запроса — дело в том, что в таблице `hotels` содержится не имя, а только номер отеля. Этот запрос оказывается несколько сложнее, поскольку в таблице `tours` у нас есть список гостиниц, но только в виде внешних ключей, связывающих эту таблицу с таблицей `hotels`. То есть чтобы вывести на странице название гостиницы и все предложения по ней, придется создать объединение таблиц:

```
mysql> SELECT h.name, t.startdate, t.price, t.duration, t.id
    -> FROM hotels AS h, tours AS t
    -> WHERE h.id = t.hotel
```

В результате этого запроса вы должны увидеть все предложения по Лондону. Объединение таблиц `tours` и `hotels` сделано по столбцам, в которых записаны номера отелей. Во второй строчке запроса кроме имен таблиц, из которых производится выборка, указаны их псевдонимы. Рекомендуется использовать в качестве псевдонимов только одну букву. Такие псевдонимы действительно сильно сокращают строки запросов, но вовсе не облегчают жизнь начинающим. Возможно, вначале вам будет проще обходиться без псевдонимов, указывая вместо них полные названия таблиц.

Часто на сайтах туристических фирм нас встречает короткое сообщение о стоимости самого дешевого тура в ту или иную страну. Сформируем список минимальных предложений по всем странам, с которыми работает наша фирма:

```
mysql> SELECT c.country, MIN(t.price) as min_offer
    -> FROM cities as c, hotels as h, tours as t
    -> WHERE t.hotel = h.id
    -> AND c.id = h.city
    -> GROUP BY c.country;
```

В результате такого запроса вы должны получить по одному предложению по каждой стране.

Обратимся к запросу, возвращающему данные обо всех предложениях по отелям из выбранного города. В каждой строке с предложением должна идти ссылка, предлагающая заказать выбранный тур. Ссылка должна вызывать сценарий с формой заказа, причем в сценарий должен передаваться идентификатор выбранного тура. Идентификатор содержится в результирующей выборке в столбце `tours.id`, и надо создать переменную `$id`, которой передается значение идентификатора. Ссылка на страницу заказа тура может выглядеть так:

```
<a href = "order_tour.php?id = <? echo $id ?>">Заказать тур</a>
```

Тогда в сценарий order_tour.php методом `GET` будет передана переменная `id` с нужным значением.

Теперь клиент или сотрудник фирмы заполняет заявку на тур. Данные из формы заказа должны попасть в таблицы `customers`, `orders` и `order_items`. Заказ тура происходит после того, как пользователь выбрал страну, город, отель и дату начала поездки, т. е. переменные `$country`, `$city`, `$hotel` и `$startdate` уже получили значения.

В результате заполнения заявки на тур мы получаем сведения об имени клиента, его адресе и паспортных данных. Сначала надо проверить, нет ли уже в базе записи об этом клиенте:

```
mysql> SELECT id, count(*) AS nalichie
    -> FROM customers
    -> WHERE family_name = 'Petrov'
    -> AND passport = '20 02 #222333'
    -> GROUP BY passport;
```

Если такой заказчик уже есть, то теперь мы знаем его номер (`id`). Если нет, то надо создавать новую запись в базе:

```
mysql> INSERT INTO customers
    -> (family_name, name, second_name, address, passport)
```

```
    -> VALUES ('Petrov', 'Sergey', 'Nikolaevitch',
    ->         'SPb, Nevsky pr. 28, 112', '20 02 #222333');
```

Так или иначе, получив `id` клиента, запишем его заказ в таблицу `orders`. Для внесения даты используем PHP-функцию `date("Y-m-d")`, которая вернет нам дату в нужном формате

```
mysql> INSERT INTO orders (order_date, customer, amount)
    -> VALUES ('2011-09-07', 28, 'cash', 25500.00);
```

И тут же запросим, какой номер заказа присвоен этому только что сделанному заказу:

```
mysql> SELECT id FROM orders
    -> WHERE customer = 28 AND order_date = '2011-09-07';
```

Занесем теперь все сведения в таблицу `order_items`:

```
mysql> INSERT INTO order_items
    -> (order_id, tour_id, quantity)
    -> VALUES (155, 12, 2);
```

Кроме описанных страниц, наш сайт должен предоставлять описание страны, города или гостиницы, для чего нужно составить соответствующие запросы:

```
mysql> SELECT description FROM country WHERE name = 'London';
```

Аналогичные запросы надо составить и для страниц, на которых будет выводиться описание города и отеля.

Для администратора фирмы потребуются Web-страницы, отражающие состояние дел — сведения о покупателях и заказах. Такие отчеты можно сформировать, сделав запросы к таблицам `customers`, `orders` и `order_items`.

В *главе 22* описываются классы, создаваемые для реализации рассматриваемого проекта. Написанные SQL-запросы составляют основу методов этих классов.

Глава 22

Объявление классов

Объектный подход при создании приложения помогает структурировать данные и упорядочить методы их получения из базы. Объекты, с которыми нам придется работать, представляют собой экземпляры различных классов. В классах определены методы, позволяющие оперировать заказами, сведениями о заказчиках, списках и описаниях стран, городов, гостиниц и туров.

Методы классов нужно написать так, чтобы они упростили запросы к базе и позволили по максимуму использовать код повторно. Основная работа на сайте связана с базой данных, поэтому любой класс должен предоставлять возможность подсоединения к серверу MySQL и отсоединения от него.

Далее в этой главе описываются классы, применяемые на страницах сайта туристической фирмы.

Класс *hat_foot*

Класс предназначен для организации вывода статической части всех страниц сайта (листинг 22.1). На рисунках, выводимых приведенными методами, изображено название фирмы и контактные данные.

Листинг 22.1. Класс `hat_foot`

```
<?php
class hat_foot
{
 public $title = "Магазин путешествий";
 function hat()
  {
    echo "<html>\n<head>\n";
    echo "<title> $this->title </title>";
    ?>
    </head>
    <body>
    <?php
```

```
    $size = getimagesize('title.gif');
    echo '<div align=right><img src="title.gif" '.$size[3].'></div>';
  }
 function footer()
  {
    $size = getimagesize('footer.gif');
    echo '<p><img src="footer.gif"'. $size[3].'></p>';
    ?>
    </body>
    </html>
    <?php
  }
}
?>
```

В начале каждой страницы сайта необходимо вывести рисунок с логотипом и названием фирмы. Это можно сделать, создав объект `$page` класса `hat_foot` и вызвав метод `hat()`, выводящий в браузер заголовочный рисунок:

```
$page = new hat_foot;
$page->hat();
```

Аналогично можно вызвать метод `footer()`, выводящий внизу страницы адрес и контактные телефоны фирмы.

Класс *baza*

Класс позволяет сделать базовые определения и является прародителем нескольких классов (листинг 22.2). В этом же классе определен метод `show_country_list()`, выводящий список всех стран, представленных в базе туристической фирмы. Все методы определены так, что при возникновении ошибок в браузер выводится только специально созданное предупредительное сообщение. Если запросы к базе завершились неудачей и не содержат данных, то вывод списков, описаний и других затребованных сведений не производится.

Листинг 22.2. Класс `baza`

```
<?php
class baza
{
  // Задание констант класса
  const USERNAME = 'root';
  const PASSWORD = 'secret';
  const DBNAME = 'firma';
  const SERVER = 'localhost';
```

```
  /*
  Конструктор класса устанавливает соединение с базой данных
  */
  function __construct($name = NULL)
  {
    if ($mysqli = new mysqli(self::SERVER, self::USERNAME, self::PASSWORD,
self::DBNAME))
    {
      $this->connection = $mysqli;
    }
    else
    {
      echo "Не удается соединиться с сервером MySQL";
      exit;
    }
    if ($name)
    {
      $this->name = $name;
    }
  }

  function show_country_list()
  {
    $quest = "SELECT name FROM countries";
    if ($result = $this->connection->query($quest))
    {
      while ($row = $result->fetch_assoc())
      {
        $spisok[] = $row['name'];
      }
      $result->close();
      return $spisok;
    }
  }

  function _desctruct()
  {
    $this->connection->close();
  }
}
?>
```

Класс *country*

Класс `country` содержит операции, которые следует выполнить для получения описания страны, списка городов в выбранной стране, а также списка самых дешевых туров в каждую из стран (листинг 22.3). Данный класс является дочерним по отношению к классу `baza` и родительским по отношению к классу `city`.

Листинг 22.3. Класс `country`

```
<?php
<?php
class country extends baza
{
  /*
  Метод show_description() выводит описание объекта.
  Выборка данных производится из таблицы, имя которой определяется
  в зависимости от класса, из которого вызывается метод.
  */
  function show_description($country = NULL)
  {
    if ($this instanceof country)
$quest = "SELECT description FROM countries WHERE name=\"$this->name\"";
    if ($this instanceof city)
$quest="SELECT description FROM cities WHERE name=\"$this->name\" AND coun-
try=\"$country\"";
    // Запрос описания объекта - страны или города
    if ($result = $this->connection->query($quest))
    {
      $row = $result->fetch_assoc();
      $opisanie = $row['description'];
    }
    $result->close();
    return $opisanie;
  }
  /*
  Вывод списка городов в выбранной стране
  */
  function show_city_list()
  {
    $request="SELECT name FROM cities WHERE country=\"$this->name\"";
    if ($result = $this->connection->query($request))
    {
      while ($row = $result->fetch_assoc())
```

```
      {
        $spisok[] = $row['name'];
      }
      $result->close();
      return $spisok;
    }
  }
  /*
  Самое дешевое предложение по выбранной стране
  */
  function min_off()
  {
    $request = "SELECT MIN( t.price ) AS min_offer ".
               "FROM cities AS c, hotels AS h, tours AS t ".
               "WHERE t.hotel = h.id ".
               "AND c.id = h.city ".
               "AND c.country = \"$this->name\"";
    if ($result = $this->connection->query($request))
    {
      $row = $result->fetch_assoc();
      return $row['min_offer'];
    }
  }

  function insert_country($name, $description)
  {
    $name = $this->connection->real_escape_string($name);
    $description = $this->connection->real_escape_string($description);
    $request = "INSERT INTO countries (name, description) VALUES (\"$name\",
\"$description\")";
    if ($result = $this->connection->query($request))
    {
      echo "Запись о стране выполнена.";
    }
    else
    {
      echo "Запись не выполнена. Пожалуйста, проверьте вводимые данные.";
    }
  }
}
?>
```

Запрос самого дешевого предложения по стране в методе `min_off()` производится с помощью внутреннего объединения таблиц `cities`, `hotels` и `tours`. Свойство `name` объекта этого класса устанавливается при запуске конструктора: название страны передается как параметр конструктора.

Все текстовые данные, полученные от клиента, обрабатываются методом `real_escape_string()` перед внесением их в базу. Этот метод экранирует все потенциально опасные символы с помощью символа обратного слэша (\).

Класс *city*

Класс `city` предназначен для работы с таблицей `cities`. Этот класс описывается предельно просто, поскольку он наследует конструктор родительского класса `country`, а также все методы родителя (листинг 22.4). Объекты этого класса потребуются, когда надо будет вывести описание какого-либо города. Тогда будет вызван метод `show_description()`, унаследованный от родительского класса `country`.

Листинг 22.4. Класс `city`

```
<?php
class city extends country
{
  function insert_city($country, $name, $description)
  {
    $country = $this->connection->real_escape_string($country);
    $name = $this->connection->real_escape_string($name);
    $description = $this->connection->real_escape_string($description);
    $request = "INSERT INTO cities (name, description, country)
                VALUES (\"$name\", \"$description\", \"$country\")";
    if ($result = $this->connection->query($request))
    {
      echo "Запись сведений о городе выполнена.";
    }
    else
    {
      echo "Запись не выполнена. Пожалуйста, проверьте вводимые данные.";
    }
  }
}
?>
```

Класс *hotel*

Класс позволяет оперировать данными базы и выводить в браузер описание отеля (листинг 22.5).

Листинг 22.5. Класс `hotel`

```
<?php
class hotel extends country
{
  function show_description($city,$country)
  {
    $query = "SELECT distinct h.name, h.description, h.stars, h.photos ".
             "FROM hotels as h, cities as c ".
             "WHERE c.name= \"$city\" ".
             "AND c.country=\"$country\" ".
             "AND h.name=\"$this->name\" ".
             "AND c.id=h.city";
    if ($result = $this->connection->query($query))
    {
      while ($row = $result->fetch_assoc())
      {
        // Создание строки звездочек для показа звездности отеля
        $st = "";
        for ($i = 0; $i < $row['stars']; $i++)
          $st = $st."*";
        echo "<table border=0> <tr> <td>";
        // Вывод картинки отеля
        $jpg = $row["photos"];
        echo "<img src='richmond.jpg'>";
        // Вывод названия отеля, его звездности и описания
        echo "</td><td valign=top> Отель ".$row['name']. $st."</td></tr>\n";
        echo "<tr><td colspan=2>". $row['description']."</td></tr></table>";
      }
    }
  }

  function insert_hotel($strana, $gorod, $hotel, $description, $stars, $foto)
  {
    $request="SELECT id from cities
              where name = \"$gorod\" AND country =\"$strana\"";
    if ($result = $this->connection->query($request))
    {
      $city = $result->fetch_assoc();
      $city_id = $city['id'];
      $hotel = $this->connection->real_escape_string($hotel);
      $description = $this->connection->real_escape_string($description);
      $foto = $this->connection->real_escape_string($foto);
```

```
      $name = $this->connection->real_escape_string($stars);
      $strana = $this->connection->real_escape_string($strana);
      $request = "INSERT INTO hotels
            (name, description, photos, stars,city)
           VALUES (\"$hotel\", \"$description\", \"$foto\",
\"$stars\",\"$city_id\")";
      if ($result = $this->connection->query($request))
      {
        echo "Запись об отеле выполнена.";
      }
      else
      {
        echo "Запись не выполнена. Проверьте вводимые данные.";
      }
    }
    else
    {
      echo "Ошибка в запросе. Пожалуйста, проверьте вводимые данные.";
    }
  }
}
?>
```

Класс *tour*

Класс позволяет получить список туров в выбранный город и страну, а также содержит вспомогательные методы для поиска даты начала тура и названия отеля по номеру тура и для определения стоимости тура по его номеру (листинг 22.6).

Листинг 22.6. Класс `tour`

```
<?php
class tour extends baza
{
  /*
  Вывод списка туров в выбранный город и страну
  */
  function display($country,$city)
  {
    $query = "SELECT h.id as h_id, h.name,
              t.price, t.duration, t.startdate,t.id ".
             "FROM hotels as h, tours as t, cities as c ".
             "WHERE t.hotel=h.id  ".
             "AND c.name=\"$city\" AND c.country=\"$country\" ".
             "AND c.id=h.city";
```

```
    echo "<table border=0 bgcolor='#CCFF99'>";
    echo "<tr><th>Отель</th> <th>Цена</th> <th>Дата</th>".
"<th>Дни</th><th> </th></tr>";
    if ($result = $this->connection->query($query))
    {
      while ($row = $result->fetch_assoc())
      {
        // Задание свойств объекта по результатам выборки из базы данных
        $this->name = $row['name'];
        $this->price = $row['price'];
        $this->duration = $row['duration'];
        $this->startdate = $row['startdate'];
        $this->id = $row['id'];
        echo "<td><a href='display_hotel.php?name=".
        $this->name."&gorod=".$city."&strana=".$country."'>".
        $this->name."</a></td>";
        echo "<td>".$this->startdate."</td>";
        echo "<td>".$this->duration."</td>";
        echo "<td> <a href='order.php?id=".$this->id.
              "'>Заказать тур </a></td></tr>";
      }
    }
    $result->close();
    echo "</table>";
  }
  /*
  Вывод названия отеля и даты начала тура
  */
  function info($tour_id)
  {
    $query="SELECT t.startdate, h.name
            FROM tours as t, hotels as h
            WHERE t.id=$tour_id
            AND t.hotel=h.id
            AND t.id=\"$tour_id\"";

    if ($result = $this->connection->query($query))
    {
      while ($row = $result->fetch_assoc())
      {
        echo "<p>Отель: ". $row['name'].
        "  Дата начала тура: ".$row['startdate']."</p>";
      }
    }
```

```
}

/*
Поиск стоимости тура по его номеру и типу команты
*/
function find_price($tour_id)
{
  $query = "SELECT price FROM tours WHERE id = $tour_id";
  if ($result = $this->connection->query($query))
  {
    while ($row = $result->fetch_assoc())
    {
      return $row["price"];
    }
  }
}

function insert_tour($strana, $gorod, $hotel, $startdate, $price, $duration)
{
  $strana = $this->connection->real_escape_string($strana);
  $gorod = $this->connection->real_escape_string($gorod);
  $hotel = $this->connection->real_escape_string($hotel);
  $startdate = $this->connection->real_escape_string($startdate);
  $price = $this->connection->real_escape_string($price);
  $duration = $this->connection->real_escape_string($duration);
  $request="SELECT hotels.id from hotels where hotels.name=\"$hotel\"".
        "AND city=(SELECT cities.id FROM cities WHERE name=\"$gorod\"".
        "AND cities.country=\"$strana\")";
  if ($result = $this->connection->query($request))
  {
    $hotel = $result->fetch_assoc();
    $hotel_id = $hotel['id'];
    echo   $hotel_id;
    $request = "INSERT INTO tours
               (startdate, hotel, price,duration)
               VALUES (\"$startdate\", \"$hotel_id\",
               \"$price\",\"$duration\")";
    if ($result = $this->connection->query($request))
    {
      echo "Запись о туре выполнена.";
    }
    else
    {
     echo "Запись не выполнена. Пожалуйста, проверьте вводимые данные.";
```

```
      }
    }
    else
    {
      echo "Ошибка в запросе. Пожалуйста, проверьте вводимые данные.";
    }
  }
}
?>
```

Метод `info()` нужен для вывода информации о туре по заданному номеру тура. Запрос адресуется двум таблицам для того, чтобы результирующая выборка содержала удобочитаемые столбцы: название отеля, а не только его номер.

Класс *customer*

Класс `customer` описывает операции поиска клиента в базе по фамилии и номеру паспорта. В случае появления нового заказчика его данные заносятся в базу. Кроме того, объявляется метод для постраничного вывода данных о заказчиках (листинг 22.7).

Листинг 22.7. Класс `customer`

```
<?php
class customer extends baza
{
  function find_client($family_name, $passport)
  {
    $family_name = $this->connection->real_escape_string($family_name);
    $passport = $this->connection->real_escape_string($passport);
    $query = "SELECT id from customers ".
             "WHERE family_name='$family_name' AND passport='$passport' ".
             "GROUP BY passport";
    if ($result = $this->connection->query($query))
    {
      while ($row = $result->fetch_assoc())
      {
        echo "<p>Заказчик найден, его номер ".$row['id'];
        return $row['id'];
      }
    }
  }
  function insert_client($name, $family_name, $second_name, $address, $passport)
  {
```

```
  // Запись нового заказчика в базу данных
  $name = $this->connection->real_escape_string($name);
  $family_name = $this->connection->real_escape_string($family_name);
  $second_name = $this->connection->real_escape_string($second_name);
  $address = $this->connection->real_escape_string($address);
  $passport = $this->connection->real_escape_string($passport);
  $query="INSERT INTO customers
       (name,family_name,second_name, address,passport)VALUES ".
      "('$name','$family_name','$second_name','$address','$passport')";
  if ($result = $this->connection->query($query))
  {
    echo "<br />Запись нового заказчика выполнена";
  }
  // Чтение номера нового заказчика
  $query = "SELECT id FROM customers ".
        "WHERE family_name='$family_name' AND passport='$passport' ".
        "GROUP BY passport";
  if ($result = $this->connection->query($query))
  {
    while ($row = mysqli_fetch_assoc($result))
    {
      $c_id = $row['id'];
      return  $c_id;
    }
  }
}

function customer_list()
{
  @$start = $_GET['start'];
  if(!$start)  $start = 0;   // Начальная строка выборки из базы
  $number = 10;              // Количество записей на странице
  $query = "SELECT count(id) as row_cnt FROM customers";
  if ($result = $this->connection->query($query))
  {
    $row = mysqli_fetch_assoc($result);
    $row_cnt = $row['row_cnt'];
    $chislo_stranits = (int)($row_cnt/$number+1);
  }

  $stop = $start+$number;
  $query = "SELECT name, family_name, passport, id
```

```
                FROM customers ORDER BY id
                LIMIT $start, $stop ";

    $HrefPage = '';
    if ($result = $this->connection->query($query))
    {
      echo "<p><b>Список клиентов</b></p>";
      echo "<table border=1 cellspacing=0 cellpadding=3>";
      echo "<tr><th>№</th> <th>Имя</th> <th>Фамилия</th>".
      "<th>Паспорт</th> </tr>";
      $i = 0;
      while($i<$number)
      {
        if($row = mysqli_fetch_assoc($result))
        {
          echo "<tr><td>".$row['id'] . "</td><td>" . $row['name'].
"</td><td>" . $row['family_name'] . "</td><td>" . $row['passport'].
"</td></tr>";
        }
        $i++;
      }
      echo "</table>";
      $tekush_stranitsa = $start/$number +1;
      echo "Номер страницы :".$tekush_stranitsa."<br>Страницы :";
      for ($link = 1; $link <= $chislo_stranits; $link++)
      {
        $PageStart = ($link - 1) * $number;
        $HrefPage = "<a href = ".$_SERVER['SCRIPT_NAME'].
                    "?start = ".$PageStart." target = _parent> ".$link.
                    "</a>";

        echo "  ".$HrefPage;
      }
    }
  }
}
?>
```

Данные для постраничного вывода списка заказчиков получаются из базы в результате следующего запроса:

```
SELECT name, family_name, passport, id
FROM customers ORDER BY id
LIMIT $start, $stop;
```

Значение параметра `$start` при первом вызове метода устанавливается в ноль, а значение параметра `$stop` отличается от значения `$start` на число записей о заказчике на странице (`$stop = $start + $number`).

Общий список покупателей занимает несколько страниц, число которых определяется в результате запроса:

```
SELECT count(id) as row_cnt FROM customers
```

Затем подсчитывается количество страниц списка:

```
$chislo_stranits = (int)($row_cnt/$number+1);
```

и указывается номер текущей страницы, который вычисляется по формуле:

```
$tekush_stranitsa = $start/$number + 1;
```

Теперь надо вывести номера страниц списка. В ссылках на эти страницы содержится и передаваемое по ссылке значение параметра `$start`:

```
$HrefPage = '';
for ($link = 1; $link <= $chislo_stranits; $link++)
{
  $PageStart = ($link - 1) * $number;
  $HrefPage = "<a href=".$_SERVER['SCRIPT_NAME']."?start=".
              $PageStart." target=_parent> ".$link."</a>";
  echo "  ".$HrefPage;
}
```

На рис. 22.1 показана страница сайта, содержащая постраничный вывод списка клиентов.

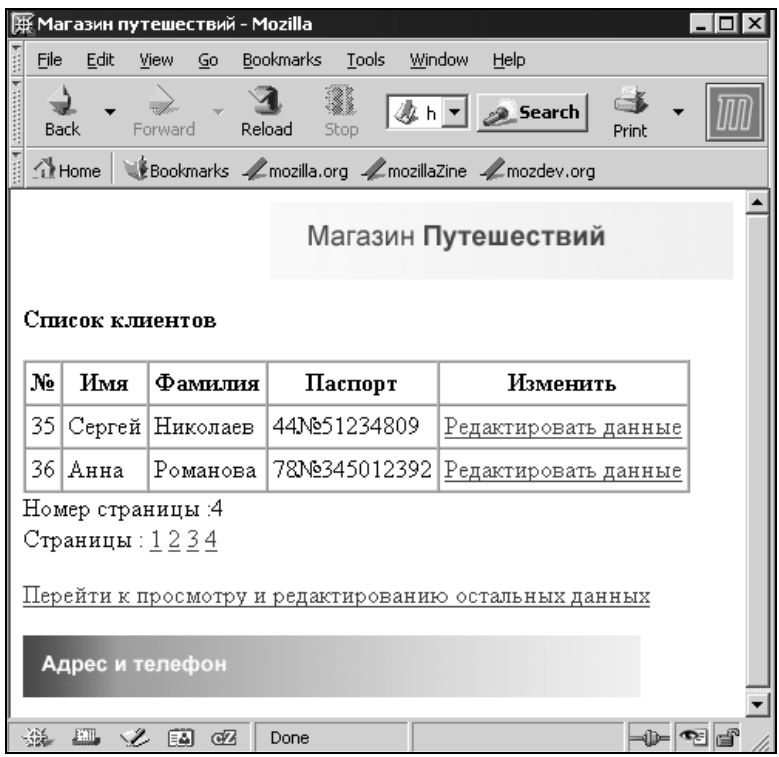


Рис. 22.1. Постраничный вывод списка клиентов

Класс *order*

Класс предназначен для объявления операций по записи заказа и постраничного вывода данных обо всех принятых заказах (листинг 22.8).

Метод `write_order()` позволяет записать новый заказ в базу. При этом необходимо добавить запись в таблицу `orders`. Создав новый заказ, надо получить его номер для того, чтобы использовать этот номер при записи данных в таблицу `order_items`.

Метод `show_orders()` позволяет осуществлять постраничный вывод записей о заказах. Для того чтобы полученные при этом данные содержали удобочитаемые фамилии клиентов, а не их номера, выборка данных производится не только из таблицы `orders`, но и из таблицы `customers`. Кроме того, сумма заказа для этого запроса берется из таблицы `order_items`.

Листинг 22.8. Класс `order`

```
<?php
class order extends baza
{
  function write_order($customer_id, $tour_id, $quantity)
  {
    $segodnya = date("Y-m-d");
    $query = "SELECT price from tours where id=$tour_id";
    if ($result = $this->connection->query($query))
    {
      while ($row = $result->fetch_assoc())
      {
        $price = $row['price'];
        $summa = $quantity*$price;
      }
    }
    $query = "INSERT INTO orders (order_date, customer, amount)
              VALUES('$segodnya',$customer_id, $summa)";
    if ($result = $this->connection->query($query))
    {
      echo "<br />Спасибо, ваш заказ принят.";
    }
    $query = "select LAST_INSERT_ID() as order_id";
    if ($result = $this->connection->query($query))
    {
      while ($row = $result->fetch_assoc())
      {
        $order_id = $row['order_id'];
```

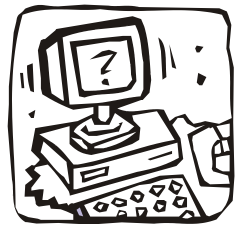
```
    }
  }
  $query = "INSERT INTO order_items (order_id, tour_id, quantity) ".
           "VALUES ($order_id, $tour_id, $quantity)";
  if ($result = $this->connection->query($query))
  {
    echo "<p>Запись в order_items сделана.";
  }
}

function show_orders()
{
  @$start = $_GET['start'];
  if(!$start)  $start = 0;   // Начальная строка выборки из базы
  $number = 10;              // Количество записей на странице
  $query = "SELECT count(id) as row_cnt FROM order_items";
  if ($result = $this->connection->query($query))
  {
    $row = mysqli_fetch_assoc($result);
    $row_cnt = $row['row_cnt'];
    $chislo_stranits = (int)($row_cnt/$number+1);
  }

  $stop = $start+$number;
  $query = "SELECT o.order_date, c.family_name, o.quantity ".
           "FROM customers as c, orders as o, order_items as oi ".
           "WHERE c.id=o.customer ".
           "AND o.id=oi.order_id ".
           "LIMIT $start,$stop ";
  $HrefPage = '';
  if ($result = $this->connection->query($query))
  {
    echo "<p>Список клиентов</p>";
    echo "<table border=1 cellspacing=0 cellpadding=1>";
    echo "<tr><th>Дата заказа</th> <th>Фамилия</th>".
         "<th>Сумма заказа</th> </tr>";
    $i = 0;
    while($i<$number)
    {
      if($row = mysqli_fetch_assoc($result))
      {
        echo "<tr> <td>".$row['order_date']."</td> <td>".
```

```
                $row['family_name']."</td><td>".$row['quantity'].
                "</td></tr>";
          }
          $i++;
        }
        echo "</table>";
        $tekush_stranitsa = $start/$number +1;
        echo "Номер страницы :".$tekush_stranitsa."<br>Страницы :";
        for ($link = 1; $link <= $chislo_stranits; $link++)
        {
          $PageStart = ($link - 1) * $number;
          $HrefPage = "<a href = ".$_SERVER['SCRIPT_NAME'].
                      "?format=orders&start=".$PageStart.
                      " target=_parent> ".$link."</a>";
          echo "  ".$HrefPage;
        }
      }
    }
}
?>
```

Классы, описанные в этой главе, используются при создании объектов в сценариях, представленных в *главе 23*.



Глава 23

Сценарии сайта

Все страницы сайта содержат код создания объектов, являющихся экземплярами различных классов. Для того чтобы не подключать каждый класс по отдельности, в начале сценария вызывается функция автозагрузки класса:

```
function __autoload($class)
{
  // Подключение файла с именем "class_имя_класса.php"
  include("class_" . $class . ".php");
}
```

Кроме того, выводятся рисунки сверху и снизу страницы с помощью методов класса `hat_foot` (см. листинг 22.1).

Домашняя страница сайта

На этой странице надо показать список самых дешевых предложений по странам, а также предложить выбрать страну и город, для которых будут отображены описания туров (листинг 23.1).

На странице выводится форма, в которой надо выбрать страну, затем появится список городов этой страны. На рис. 20.2 был дан вид домашней страницы после выбора страны. Выбрав город, посетитель переходит на страницу с перечнем туров в выбранный город.

Листинг 23.1. index.php — домашняя страница сайта

```
<?php
// Открытие сессии
session_start();
@$strana = $_GET['format'];
// Автозагрузка класса
function __autoload($class)
{
// Подключение файла с именем "class_имя_класса.php"
  include("class_" . $class . ".php");
```

```
}
$db = new baza();
$spisok = $db->show_country_list();
$page = new hat_foot;
$page->hat();
echo "Наши предложения :<br />";
echo "<table border=0>";
for ($i = 0; $i < count($spisok); $i++)
{
  echo '<tr><td bgcolor="#CCFF99">';
  echo $spisok[$i]."<br>";
  echo'</td><td bgcolor="#CCFF99">';
  $starna = new country($spisok[$i]);
  echo "c ".$starna->min_off()."<br>";
  echo "</td></tr>";
}

echo "<table border=0><tr><td colspan=2>".
     "Для просмотра предложений выберите страну и город</td></tr>".
     "<tr> <td>";

// Создание формы для выбора страны с передачей данных в этот же скрипт

echo '<form><select name="format" size="1">';
for ($i = 0; $i < count($spisok); $i++)
{
  // Создание элемента формы для вывода списка стран
  if ($spisok[$i] == $strana )
  {
    echo '<option selected "value="'.$spisok[$i].'">'. $spisok[$i];
  }
  else
  {
     echo '<option "value="'.$spisok[$i].'">'. $spisok[$i];
  }
}
echo '</select><input type="submit" value="OK"></form>';
echo "</td><td>";

// Получение имени страны из вышеописанной формы
@$country = $_GET['format'];
// Регистрация переменной сессии - country
```

```
$_SESSION['strana'] = $country;

// Если страна выбрана, печатаем список городов в данной стране
if ($country != '')
{
  echo '<form action="select_tour.php"><select name="format2" size="1">';
  // Создание элемента формы для вывода списка городов
  $starna = new country($country);
  $goroda = $starna->show_city_list();
  for ($i = 0; $i < count($goroda); $i++)
  {
    echo '<option value="'.$goroda[$i].'">'. $goroda[$i];
  }

  echo '</select><input type="submit" value="OK"></form>';
}
echo "</td></tr></table>";
$page->footer();
?>
```

Приведенный сценарий содержит две формы, причем данные из обеих форм передаются обратно в вызывающий сценарий. Поэтому часть данных, например название выбранной страны, удобнее хранить в сессии.

Выбор и заказ тура

После выбора страны посетитель попадает на страницу, которая содержит список предложений туров в выбранный город (листинг 23.2). На этой странице имеются ссылки на описания стран, городов и отелей. Ссылка **Заказать тур** приводит на страницу order.php, где можно заказать тур.

Листинг 23.2. select_tour.php — перечень туров и ссылки на описания стран, городов и отелей

```
<?php
session_start();
$city = $_GET['format2'];
$country = $_SESSION['strana'];
$_SESSION['strana'] = $country;
$_SESSION['gorod'] = $city;
function __autoload($class)
{
  // Подключение файла с именем "class_имя_класса.php"
  include("class_" . $class . ".php");
```

```
}
$page = new hat_foot;
$page->hat();
echo "<a href = 'display_country.php?name=" . $country . "'>" .
     $country . "</a> ";
echo "<a href = 'display_city.php?name=" . $city .
     "&strana=".$country."'>" . $city."</a>";
$tur = new tour();
$tur->display($country, $city);
$page->footer();
?>
```

Часть параметров в приведенном сценарии передается в адресной строке ссылки, а часть — через переменные сессии. При передаче параметров в адресной строке из соображений безопасности используются имена переменных, больше нигде в скрипте не встречающиеся.

В листинге 23.3 дан код формы, заполняемой при заказе тура. Метод `info()` класса `tour` выводит информацию о выбранном туре.

Листинг 23.3. order.php — форма заказа тура

```
<?php
// Автозагрузка класса
function __autoload($class)
{
  // Подключение файла с именем "class_имя_класса.php"
  include("class_" . $class . ".php");
}
$page = new hat_foot;
$page->hat();
$tour_id = $_GET['id'];

$tur = new tour();
$tur->info($tour_id);
?>

<form method="POST" action="process_order.php">
<table>
<tr>
<td>Фамилия: </td>
<td><input type="text" name="fname"></td>
</tr><tr>
<td>Имя: </td>
```

```
<td><input type="text" name="name"></td>
</tr><tr>
<td>Отчество: </td>
<td><input type="text" name="sname"></td>
</tr><tr>
<td>Адрес: </td>
<td><input type="text" name="ad"></td>
</tr><tr>
<td>Паспортные данные: </td>
<td><input type="text" name="pas"></td>
</tr><tr>
<td>Количество заказываемых туров: </td>
<td><input type="text" name="many"></td>
</tr><tr>
</td>
</tr>
</table>
<input type="hidden" name="tour_id" value="<?php echo $tour_id; ?>">
<input type="submit" value="Заказать">

</form>
<?php
$page->footer();
?>
</body>
</html>
```

Сценарий process_order.php, представленный в листинге 23.4, получает данные из приведенной формы. Данные передаются методом `POST`, затем из них удаляются HTML-теги с помощью функции `htmlspecialchars()`.

Метод `find_price()` возвращает стоимость заказанного тура.

Листинг 23.4. process_order.php — оформление заказа

```
<?php
// Автозагрузка класса
function __autoload($class)
{
  // Подключение файла с именем "class_имя_класса.php"
  include("class_" . $class . ".php");
}
$page = new hat_foot;
$page->hat();
```

```
$family_name = htmlspecialchars($_POST['fname']);
$name = htmlspecialchars($_POST['name']);
$second_name = htmlspecialchars($_POST['sname']);
$address = htmlspecialchars($_POST['ad']);
$passport = htmlspecialchars($_POST['pas']);
$quantity = intval(htmlspecialchars($_POST['many']));
$tour_id = htmlspecialchars($_POST['tour_id']);

$tur = new tour();
$price = doubleval($tur->find_price($tour_id));

$client = new customer;
@$nomer_clienta = $client->find_client($family_name, $passport);
$zakaz = new order();
if (isset($nomer_clienta))
{
  $zakaz->write_order($nomer_clienta, $tour_id, $quantity);
}
else
{
  @$nomer_clienta = $client->insert_client($name, $family_name,
                    $second_name, $address, $passport);
  $zakaz->write_order($nomer_clienta, $tour_id, $quantity);
}
$page->footer();
?>
```

Порядок записи заказа зависит от того, есть ли уже заказчик в базе. Сначала производится его поиск по базе методом `find_client()`, если заказчик найден, то заказ записывается в таблицы `orders` и `order_items` методом `write_order()`. Поскольку поиск клиента может завершиться неудачей, то переменная `$nomer`, которой передается значение идентификатора найденного клиента, предваряется оператором подавления ошибки (`@`). Если же клиент делает свой первый заказ, то к записи заказа добавляется запись данных о новом клиенте методом `insert_client()`, и только потом выполняется регистрация заказа.

Страницы описаний стран, городов и отелей

Сценарии, выводящие описания страны, города и отеля, однотипны. Сценарий вывода описания страны (листинг 23.5) получает название страны по ссылке, т. е. методом `GET`, затем происходит вызов метода `show_description()` для получения информации о стране. Метод `show_city_list()` возвращает список городов выбранной страны.

Листинг 23.5. display_country.php — описание страны

```
<?php
// Автозагрузка класса
function __autoload($class)
{
  // Подключение файла с именем "class_имя_класса.php"
  include("class_" . $class . ".php");
}
$country = $_GET['name'];
$strana = new country($country);
echo $strana->show_description();
echo "<br/>Города: <br/>";
$spisok = $strana->show_city_list();
foreach ($spisok as $gorod)
{
  echo $gorod."<br/>";
}
?>
```

Сценарий, представленный в листинге 23.6, получает по ссылке название страны и города (учтите, что города в разных странах могут называться одинаково). Метод `show_description()` класса `city` унаследован от класса `country`, а потому в отдельных комментариях не нуждается.

Листинг 23.6. display_city.php — описание города

```
<?php
// Автозагрузка класса
function __autoload($class)
{
  // Подключение файла с именем "class_имя_класса.php"
  include("class_" . $class . ".php");
}
$page = new hat_foot;
$page->hat();
$city = $_GET['name'];
$country = $_GET['strana'];
echo "Страна :".$country."  Город: ".$city."<br/>";
$gorod = new city($city);
echo $gorod->show_description($country);
$page->footer();
$gorod->connect_close();
?>
```

Вывод описания отеля представлен в листинге 23.7. Данные о стране, городе и названии отеля сценарий получает методом `GET` по ссылке, как и предыдущие сценарии, а вот метод `show_description()` в классе `hotel` перегружен и выводит сведения об отеле, включающие фотографии и данные о категории отеля.

Листинг 23.7. display_hotel.php — описание отеля

```
<?php
// Автозагрузка класса
function __autoload($class)
{
  // Подключение файла с именем "class_имя_класса.php"
  include("class_" . $class . ".php");
}
$page = new hat_foot;
$page->hat();
$country = $_GET['strana'];
$city = $_GET['gorod'];
$hotel = $_GET['name'];

echo "Страна: ".$country."  Город: ".$city."<br/>";
$gostinica = new  hotel($hotel);
$gostinica->show_description($city,$country);
$page->footer();
?>
```

Администрирование сайта

Первый сценарий интерфейса администрирования очень короткий (листинг 23.8). Он постранично выводит из базы список клиентов методом `customer_list()` и содержит ссылку на меню администратора, позволяющее редактировать данные в базе.

Листинг 23.8. admin.php — вывод списка клиентов и ссылка на меню администратора

```
<?php
function __autoload($class)
{
  // Подключение файла с именем "class_имя_класса.php"
  include("class_" . $class . ".php");
}
$page = new hat_foot;
$page->hat();
$client = new customer;
```

```
$client->customer_list();
echo "<p><a href=admin_insert.php>".
     "Перейти к просмотру и редактированию остальных данных</a>";
$page->footer();
?>
```

Следующий сценарий, используемый администратором сайта (листинг 23.9), предназначен для вывода меню и выполнения административных функций по добавлению и изменению данных базы.

Листинг 23.9. admin_insert.php — сценарий с меню администратора и основными формами для редактирования данных базы

```
<?php
function __autoload($class)
{
// Подключение файла с именем "class_имя_класса.php"
  include("class_" . $class . ".php");
}
@$tema=$_GET['format'];
$page= new hat_foot;
$page->hat();
if(!$tema)
{
echo "<table border=0><tr><td colspan=2>Выберите, с какими данными вы хотите
работать: </td></tr><tr><td>";
echo '<form><select name="format" size="1">';
echo '<option value="countries">Добавить сведения о стране';
echo '<option value="cities">Добавить сведения о городе';
echo '<option value="hotels">Добавить сведения об отеле';
echo '<option value="tours">Добавить тур';
echo '</select><input type="submit" value="OK"></form>';
}
else
{
       switch ($tema)
       {
       case 'orders':
       $order = new order();
       $order->show_orders();
       break;
       case 'countries':
       echo "Введите данные: \n";
       echo "<form method=POST action=insert_country.php>";
```

```
    echo "<input type=text name=strana value='Название страны'>\n";
    echo "<input type=text name=descr value='Описание страны'>\n";
    echo "<input type=hidden name=tbl value=".$tema.">\n";
    echo "<input type=submit value=OK>\n";
    echo "</form>";
    break;
    case 'cities':
    echo "Введите данные: \n";
    echo "<table border=0><form method=POST action=insert_city.php>";
    echo "<tr><td><input type=text name=strana value='Название
    страны'></td></tr>\n";
    echo "<tr><td><input type=text name=gorod value='Название
    города'></td></tr>\n";
    echo "<tr><td><textarea name=descr cols=40 rows=3>Описание
    города</textarea></td></tr>\n";
    echo "<input type=hidden name=tbl value=".$tema.">\n";
    echo "<tr><td><input type=submit value=OK></td></tr>\n";
    echo "</table></form>";
    break;
    case 'hotels':
    echo "Введите данные: \n";
    echo "<table border=0><form method=POST action=insert_hotel.php>";
    echo "<input type=hidden name=tbl value=".$tema.">\n";
    echo "<tr><td><input type=text name=strana value='Название
    страны'></td></tr>\n";
    echo "<tr><td><input type=text name=gorod value='Название
    города'></td></tr>\n";
    echo "<tr><td><input type=text name=hotel value='Название
    отеля'></td></tr>\n";
    echo "<tr><td><textarea name=descr cols=40 rows=3>Описание
    отеля</textarea></td></tr>\n";
    echo "<tr><td><input type=text name=stars value='Класс отеля
    в  *'></td></tr>\n";
    echo "<tr><td><input type=text name=foto value='Имя файла
    с фото'></td></tr>\n";
echo "<tr><td><input type=submit value=OK></td></tr>\n";
    echo "</table></form>";
    break;
    case 'tours':
    echo "Введите данные: \n";
    echo "<table border=0><form method=POST action=insert_tour.php>";
    echo "<tr><td><input type=text name=strana value='Название
    страны'></td></tr>\n";
    echo "<tr><td><input type=text name=gorod value='Название
    города'></td></tr>\n";
```

```
        echo "<tr><td><input type=text name=hotel value='Название
        отеля'></td></tr>\n";
        echo "<tr><td><input type=text name=startdate value='Дата
        YYYY-MM-DD'></td></tr>\n";
        echo "<tr><td><input type=text name=price value='Стоимость
        тура'></td></tr>\n";
        echo "<tr><td><input type=text name=duration value='Сколько
        дней'></td></tr>\n";
        echo "<input type=hidden name=tbl value=".$tema.">\n";
        echo "<tr valign=top><td><input type=submit value=OK></td></tr>\n";
        echo "</table></form>";
        break;
        }
}
$page->footer();
?>
```

Сценарий записи данных о стране весьма прост (листинг 23.10). Кроме многократно описанных классов автозагрузки и вывода на экран общих для всех страниц рисунков, этот сценарий только получает данные, переданные методом `POST`, из соответствующего массива и заносит их в базу, вызывая метод `insert_country()`, определенный для объектов класса `country`.

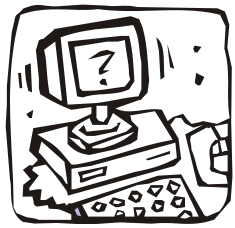
Листинг 23.10. insert_country.php — запись данных о стране в базу

```
<?php
// Автозагрузка класса
function __autoload($class)
{
  // Подключение файла с именем "class_имя_класса.php"
  include("class_" . $class . ".php");
}
$page = new hat_foot;
$page->hat();
$description = $_POST['descr'];
$strana = $_POST['strana'];
$tbl_name = $_POST['tbl'];
$country = new country();
$country->insert_country($strana, $description);
$page->footer();
?>
```

Остальные сценарии внесения изменения в базу однообразны и не содержат принципиально нового кода. Все сценарии рассматриваемого приложения можно найти на компакт-диске.

ЧАСТЬ IV

XML и PHP



Глава 24

Язык XML

Компьютерные системы и базы данных хранят информацию в несовместимых форматах. Одной из самых сложных задач для разработчиков программного обеспечения была реализация обмена данными между такими системами через Интернет. Изобилие разнообразных платформ, приложений, языков программирования и форматов данных потребовало создать простые стандарты для хранения и передачи информации. Только один формат понятен для всех приложений — простой текст.

В 1994 г. для руководства разработкой общих WWW-протоколов, таких как HTML, CSS и XML, был создан Консорциум W3C (World Wide Web Consortium). Наиболее важная деятельность консорциума — это разработка Web-спецификаций (они называются *рекомендациями*), которые описывают протоколы (такие как HTML и XML) и другие составные части WWW.

Материал этой и следующих глав, посвященных языку XML, не является полным описанием этого языка. За всеобъемлющей информацией следует обращаться к документам W3C. Рассмотрение XML в этой книге ограничено теми аспектами, которые нашли применение в сценариях PHP. Модуль PHP 5 включает в себя расширения для обработки XML-документов.

XML (eXtensible Markup Language) — это расширяемый язык разметки, созданный не для форматирования и отображения данных, а для описания данных. XML — это просто текст с тегами. Так же как в HTML, теги представляют собой короткую строку символов, заключенных в угловые скобки. В отличие от языка HTML, в XML теги не определены, при создании XML-документов автор создает свои собственные теги. XML-документы используют простой синтаксис.

Синтаксис XML. Правильно оформленный XML

Программы, читающие и анализирующие XML-документы, часто называют *парсерами* (от англ. *parse* — обрабатывать). Современные браузеры, такие как Internet Explorer или Mozilla, включают в себя парсеры XML-документов, отвечающие стандартам W3C. XML-документ, имеющий синтаксис, соответствующий стандартам, называется *правильным* (well formed) XML-документом. Если при чтении XML-файла браузер обнаружил ошибку, в окне появляется сообщение о найденной ошибке, и содержимое документа не отображается. XML-спецификация консорциума W3C указывает, что программа не должна продол-

жать обработку XML-документа, если она обнаружит ошибку. Это сделано для того, чтобы было легко создавать программное обеспечение для XML и чтобы все XML-документы были совместимы.

Критерии правильности XML-документа можно сформулировать весьма кратко.

- Каждый открывающий тег должен иметь соответствующий закрывающий тег. В отличие от HTML, в XML все теги должны быть парными:

```
<p>Текст параграфа.</p>
```

Пустой тег может записываться следующим образом: `<br />`.

- Теги не могут перекрывать друг друга. В XML все элементы должны быть правильно вложены друг в друга:

```
<parent>
  <child>text</child>
</parent>
```

Все XML-документы должны иметь единственную пару тегов, задающую корневой элемент. Все остальные элементы должны быть вложены в корневой элемент.

XML-элементом является все, что заключено между начальным тегом элемента и конечным тегом элемента, включая сами теги.

Элемент может иметь элементное содержимое, смешанное содержимое, простое текстовое содержимое, или он может быть пустым. Все элементы могут иметь дочерние элементы. В приведенном ранее примере содержимое элемента `parent` является элементным, а элемента `child` — текстовым. Элементы также могут иметь атрибуты.

Пример XML-документа представлен в листинге 24.1.

Листинг 24.1. XML-документ

```
<person>
  <name>
    <first_name>Ivan</first_name>
    <second_name>Vasilievitch</second_name>
    <family_name>Bunsha</family_name>
  </name>
  <address>Moscow</address>
  <telephone>111-22-33</telephone>
</person>
```

Наберите этот пример в любом текстовом редакторе, сохраните в файле с расширением xml и откройте его в браузере. Вы увидите, как браузер обрабатывает XML-документ (рис. 24.1).

- Имена элементов должны подчиняться соглашениям XML о названиях:
  - названия могут содержать буквы, цифры и другие символы;
  - названия не могут начинаться с цифры или знака препинания;
  - названия не могут начинаться с букв xml (или XML, или Xml и т. д.);
  - в названии не должно быть пробелов.

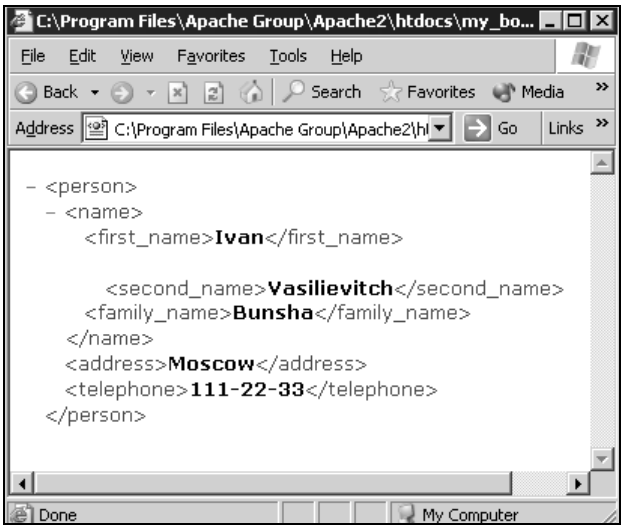


Рис. 24.1. XML-документ в браузере

- Регистр символов (верхний/нижний) для XML существенен. Тег `<Letter>` отличается от тега `<letter>`. Таким образом, начальные и конечные теги должны писаться в одном регистре.
- XML сохраняет пробелы внутри текста.

  В XML перевод строки всегда делается с помощью символов `LF`.

  В приложениях под Windows новая строка текста обычно отмечается двумя парами символов `CR LF` (*carriage return, line feed* — возврат каретки, перевод строки). В приложениях под UNIX новая строка обычно отмечается символами `LF`.

XML-декларация

В начале документа бывает полезно указать конкретный тип этого документа. XML предоставляет в наше распоряжение специальную декларацию для того, чтобы пометить документы как документы XML.

XML-декларация всегда начинается с символов `<?xml` и заканчивается символами `?>`. Декларация должна располагаться в самом начале файла, т. е. первым символом файла должна быть угловая скобка и никаких концов строки и пробелов. Декларация сама по себе не является частью XML-документа. Она не является XML-элементом и может не иметь закрывающего тега.

Типичная декларация выглядит так:

```
<?xml version='1.0' ?>
```

Кодировка в XML

XML-документ может содержать символы различных языков мира. Чтобы программа обработки (парсер) корректно обрабатывала эти символы, нужно сохранять документ с указанием используемой кодировки.

Тогда декларация может выглядеть так:

```
<?xml version="1.0" encoding="windows-1251"?>
```

Без указания кодировки в XML-документ нельзя вставлять буквы русского языка, в противном случае ошибка неминуема. Но, указав кодировку, вы можете использовать русские буквы даже в именах тегов. Следует помнить, что внутреннее представление данных в PHP осуществляется в кодировке UTF-8.

Кроме того, в настоящее время UTF-8 получает все более широкое распространение, поэтому при возможности целесообразно использовать именно эту кодировку, обозначая ее в декларации:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Атрибуты

Элементы XML могут содержать атрибуты в начальном теге. Атрибуты — это пары "имя = значение", поставленные в соответствие одному из элементов. Атрибуты применяются для предоставления дополнительной информации об элементах. Они должны находиться при открывающем теге, но не при закрывающем.

Атрибуты всегда должны иметь значение, даже если оно — всего лишь пустая строка, и эти значения должны заключаться в кавычки. Допускаются как двойные, так и одинарные кавычки. Например:

```
<file type="gif">computer.gif</file>
```

Комментарии

Комментарии позволяют вставлять в XML-документ текст, который на самом деле не является частью документа, а предназначен для тех, кто будет читать исходный XML-документ.

Комментарии не могут располагаться внутри тега.

Стандарт XML устанавливает, что XML-анализатор не должен передавать эти комментарии приложению, т. е. что никто не ждет от программного приложения использования информации, содержащейся в комментарии.

Комментарии всегда начинаются символами `<!--` и заканчиваются символами `-->` (листинг 24.2).

Листинг 24.2. Применение комментариев в XML

```
<?xml version="1.0" encoding="utf-8"?>
<name>
  <first>Фродо</first>
  <!-- Отчество Фродо до нас не дошло -->
  <last>Сумкинс</last>
</name>
```

Процессуальная инструкция

Процессуальная инструкция (т. е. инструкция по обработке вкладываемого в нее текста) имеет следующую общую форму записи:

```
<?Имя_приложения инструкция ?>
```

Инструкция передает информацию указанному в ее начале приложению. С одной процессуальной инструкцией мы уже имели дело:

```
<?php Код_на_PHP ?>
```

Эта инструкция, как вы знаете, предназначена модулю PHP и указывает, что текст внутри инструкции подлежит обработке. В дальнейшем мы будем часто использовать еще одну процессуальную инструкцию:

```
<?xml-stylesheet type="text/xsl" href="patterns.xsl"?>
```

Эта инструкция предназначается парсеру XML и указывает на необходимость подключения файла стилевых таблиц patterns.xsl и обработки XML-документа с помощью этого подключенного файла.

Пространства имен XML

Поскольку имена элементов в XML не определены, возможны конфликты, когда два различных документа используют одно и то же имя тега для описания различных данных.

В следующем XML-документе информация заключена в таблицу (table):

```
<table>
  <tr>
    <td>sun</td>
    <td>rain</td>
  </tr>
</table>
```

А в этом XML-документе содержится информация о столе как части мебели (table):

```
<table>
  <name>Kitchen table</name>
  <width>80</width>
  <length>120</length>
</table>
```

Если эти два XML-документа соединить вместе, возникнет конфликт имен элементов, потому что оба документа содержат элементы `<table>` с разным содержанием и смыслом.

Пространства имен (namespaces) XML позволяют избегать конфликтов имен элементов. Пространство имен XML — это коллекция имен, используемых в XML-документах для обозначения элементов и атрибутов. Эта коллекция идентифицируется именем ресурса в Интернете (URI).

Пользуясь терминами интернет-технологий, можно сказать, что русский язык — это коллекция слов. О принадлежности слова к тому или иному языку говорят, например, так: это слово японского языка. Название языка идентифицирует коллекцию слов, так же как URI идентифицирует пространство имен.

Для указания пространства имен в начальный тег элемента помещается атрибут с использованием следующего синтаксиса:

```
xmlns:prefix="namespace"
```

Здесь `xmlns` — ключевое слово, означающее, что начинается определение пространства имен; *`prefix`* — префикс, который будет использоваться во всем документе для обозначения принадлежности тега, перед которым он ставится, к этому пространству имен; "*`namespace`*" — идентификатор пространства имен.

Когда пространство имен задается в начальном теге элемента, все дочерние его элементы, обладающие тем же префиксом, относятся к тому же пространству имен.

Адрес ресурса, участвующий в идентификации пространства имен, не используется парсером для поиска информации. Единственная задача этого адреса — дать пространству имен уникальное имя. Тем не менее, очень часто пространство имен используют как указатель на реальную Web-страницу, содержащую информацию об этом пространстве имен.

В следующем XML-документе (листинг 24.3) пространство имен определяется с помощью URI **http://www.w3.org/TR/html4/**.

Листинг 24.3. Пространство имен

```
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

Кроме использования префиксов `h` к тегу `<table>` был добавлен атрибут `xmlns`, чтобы дать префиксу имя, связанное с пространством имен.

Особые символы

Некоторые особые XML-символы не могут быть корректно обработаны и должны заменяться *ссылками на сущности*.

Если вы поместите внутрь XML-элемента символ левой угловой скобки <, это вызовет ошибку, потому что парсер интерпретирует его как начало нового элемента. Поэтому писать вот так нельзя:

```
<message>if salary < 1000 then</message>
```

Чтобы избежать ошибки парсера, нужно заменить символ < ссылкой на сущность:

```
<message>if salary &lt; 1000 then</message>
```

В XML есть пять изначально заданных сущностей:

- `<` — меньше чем (<);
- `>` — больше чем (>);
- `&` — амперсанд (&);
- `&apos;` — апостроф (');
- `&quot;` — двойная кавычка (").

Ссылки на сущности всегда начинаются с амперсанда (`&`) и заканчиваются точкой с запятой.

CDATA

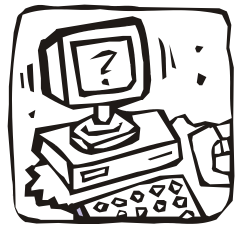
Если содержимое XML-элемента содержит множество символов левой угловой скобки или знак амперсанда (например, надо вывести программный код), то можно задать секцию `CDATA`, очень похожую по смыслу ее применения на тег `<pre>` в HTML.

`CDATA` (Character DATA) переводится как "символьные данные". Все, что находится внутри раздела `CDATA`, игнорируется парсером. Секция `CDATA` начинается символами `<![CDATA[` и заканчивается символами `]]>` (листинг 24.4).

Листинг 24.4. Раздел `CDATA`

```
<script>
<! [ CDATA [
function matchwo(a,b)
{
  if (a < b && a < 0) then
  {
     return 1
  } else
 {
     return 0
}
]]>
</script>
```

В этом примере все, что расположено внутри секции `CDATA`, не будет анализироваться парсером.



Глава 25

Преобразование XML-документов с помощью стилевых таблиц XSL

Как было показано в *главе 24*, в самом XML-документе нет никаких указаний на то, как его следует отображать. А вид документа в браузере без этих указаний явно не годится для созерцания пользователем. Значит, надо разработать правила преобразования XML-документа к удобочитаемому виду. Для HTML-документов существуют стилевые таблицы CSS, а для XML-документов — язык XSL.

XSL (eXtensible Stylesheet Language) — расширяемый язык стилей, применяется для преобразования и форматирования XML-документов.

XSL состоит из трех языков:

- XSLT (eXtensible Stylesheet Language Transformations) — язык преобразований XML-документов;
- XPath — язык определения частей и путей к элементам XML;
- XSL Formatting Objects — язык определения формата показа XML.

XSLT — это язык преобразования XML-документа в документы другого вида. Такое преобразование позволяет выделить нужную часть информации из документа и представить ее в удобном для чтения виде.

XPath — язык поиска частей XML-документа. XPath — это та часть языка XSLT, которая позволяет найти требуемый элемент XML-документа.

XSLT и XPath были предложены в виде двух отдельных официальных рекомендаций W3C 16 ноября 1999 г.

XSLT является наиболее важной частью стандарта XSL. Именно он применяется для того, чтобы преобразовывать XML-документ в другой XML-документ или в документы другого типа.

С точки зрения XSLT существует входной документ — XML-документ, подлежащий преобразованию, и выходной документ — результат преобразования. XSLT можно применять для преобразования XML-документа в формат, знакомый браузерам.

Кроме того, XSLT может добавлять совершенно новые элементы в выходной документ или удалять элементы. Этот язык может изменить порядок следования элементов, произвести проверку и на ее основе решить, какие элементы показывать, и еще многое другое.

Процесс преобразования XML-документа с помощью таблицы стилей XSL напоминает прохождение света через призму: под действием стекла призмы исходный свет преобразуется и выглядит на выходе совсем иначе (рис. 25.1).

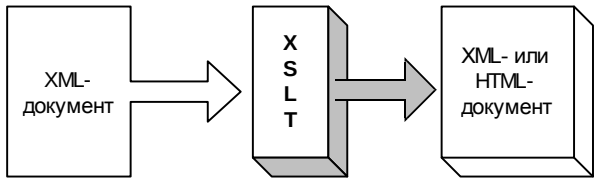


Рис. 25.1. Преобразование XML-документа с помощью XSLT-преобразований

В процессе преобразования XSLT использует XPath для определения тех частей во входном документе, которые соответствуют одному или нескольким заранее заданным шаблонам. Когда такое соответствие обнаруживается, XSLT преобразует соответствующую часть входного документа в часть выходного документа. Те части входного документа, которые не соответствуют шаблону, будут попадать в выходной документ без изменений.

Таблицы стилей XSL

Преобразование XML-документа в соответствии с XSL-шаблонами может выполнять сам браузер. Для этого в него должен быть встроен XML-парсер, соответствующий стандартам W3C. В этой главе мы будем создавать XML- и XSL-документы и обрабатывать их парсером браузера — будь то Internet Explorer или Mozilla, т. к. в обоих браузерах парсеры соответствуют стандартам W3C.

Чтобы сделать XML-данные доступными для всех видов браузеров, вне зависимости от того, какие парсеры в них встроены, нам следует преобразовывать XML-документ на сервере и отправлять его браузеру уже в виде HTML-документа. В *главе 29* будет показано, что сценарий PHP тоже может служить парсером XML-документов. Для этого необходимо использовать некоторые расширения PHP, позволяющие производить XSL-преобразования и анализ XML-документов.

Для преобразования документов с помощью браузера ему необходимо передать таблицу стилей XSL, хранящуюся в одном файле, и сам XML-документ, сохраненный в другом файле. Браузер будет читать XML-документ, искать в нем соответствие шаблонам, указанным в таблице стилей. В случае если браузер обнаружит соответствие шаблону в документе, в выходной документ будут переданы данные, указанные в шаблоне, и браузер будет читать XML-документ дальше, и так пока не проверит все шаблоны. Для того чтобы браузер использовал файл с таблицей стилей XSL, в XML-документ (листинг 25.1) следует добавить ссылку на стилевой файл сразу после XML-декларации.

Листинг 25.1. XML-документ с ссылкой на таблицу стилей

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="movie.xsl"?>
<movies>
<movie id="1">
  <title>Бриллиантовая рука</title>
```

```
  <characters>
   <character>
    <name>Семен Семеныч</name>
    <actor>Юрий Никулин</actor>
   </character>
   <character>
    <name>Лелик</name>
    <actor>Анатолий Папанов</actor>
   </character>
  </characters>
  <plot>
   Приключения
  </plot>
  <rating type="stars">5</rating>
</movie>

<movie id="2">
  <title>Бриллианты навсегда</title>
  <characters>
   <character>
    <name>Джеймс Бонд</name>
    <actor>Шон Коннери</actor>
   </character>
   <character>
    <name>Подруга Бонда</name>
    <actor>Джилл Сент-Джон</actor>
   </character>
  </characters>
  <plot>
   Шпионские приключения
  </plot>
 <rating type="stars">4</rating>
</movie>
</movies>
```

Шаблоны в таблицах стилей XSL

Поскольку таблица стилей сама по себе является XML-документом, она начинается с декларации XML:

```
<?xml version="1.0" encoding="utf-8"?>
```

Тег `<xsl:stylesheet>`, который находится на второй строке таблицы стилей, задает начало таблицы, в этой же строке указывается и пространство имен:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Для определения вида, в котором будут выводиться XML-элементы, в XSL используется один или несколько шаблонов. Для связывания шаблона с XML-элементом предназначен специальный атрибут соответствия.

Создадим таблицу стилей XSL с шаблоном преобразования. Тег `<xsl:template>` задает начало шаблона. Атрибут шаблона `match="/"` связывает шаблон и корень (`/`) оригинального XML-документа. Браузер просматривает XML-документ и ищет в нем корень документа. Как только корень найден, из таблицы стилей берется та часть, которая следует за началом шаблона `<xsl:template match="/">`, и передается в выходной документ.

Остальная часть документа содержит сам шаблон, кроме двух последних строк, которые задают конец шаблона и конец таблицы стилей (листинг 25.2).

Листинг 25.2. Таблица стилей XSL

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html> <body>
  <table border="1">
    <tr>
      <th>Название</th> <th>Жанр</th>
    </tr>
    <tr>
      <td> Текст </td>
      <td> Текст </td>
    </tr>
  </table>
</body> </html>
</xsl:template>
</xsl:stylesheet>
```

Впишите в XML-документ, представленный в листинге 25.1, ссылку на стилевой файл из листинга 25.2, сохраните XML- и XSL-документы и откройте XML-файл в браузере. Вы должны увидеть то, что изображено на рис. 25.2.

Название	Жанр
Текст	Текст

Рис. 25.2. Результат форматирования XML-документа с помощью XSL-таблицы, представленной в листинге 25.2

Да, выходной документ содержит текст шаблона и больше ничего. Но ведь мы и попросили найти корень XML-документа и передать на выход текст шаблона. Посмотрите исходный код полученного документа — это все тот же XML, но обработка с помощью таблицы стилей изменила его вид в браузере до неузнаваемости.

Передача содержимого элемента в выходной документ

Теперь научимся передавать данные из XML-документа в выходной документ. Для передачи XML-элемента в выходной поток XSL-преобразования применяется элемент `<xsl:value-of>` (листинг 25.3).

Листинг 25.3. Применение элемента `<xsl:value-of>`

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html> <body>
  <table border="1">
    <tr>
      <th>Название</th> <th>Жанр</th>
    </tr>
    <tr>
      <td><xsl:value-of  select="movies/movie/title"/></td>
      <td><xsl:value-of  select="movies/movie/plot"/></td>
    </tr>
  </table>
</body> </html>
</xsl:template>
</xsl:stylesheet>
```

Значение атрибута `select` называется *XSL-паттерном* (XSL Pattern). Прямой слэш (`/`) в значении атрибута используется для указания на вложенные элементы.

Вот теперь мы получили более интересный результат (рис. 25.3).

Но, найдя по одному элементу `title` и `plot`, таблица стилей закончила свою работу и не стала искать ничего больше.

Название	Жанр
Бриллиантовая рука	Приключения

Рис. 25.3. Результат форматирования XML-документа с помощью XSL-таблицы, представленной в листинге 25.3

Создание цикла с помощью <xsl:for-each>

Для просмотра всех названий фильмов и их жанров нам потребуется цикл, который можно организовать с помощью XSL-элемента `<xsl:for-each>` (листинг 25.4).

Листинг 25.4. Организация цикла с помощью <xsl:for-each>

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html> <body>
  <table border="1">
    <tr>
      <th>Заголовок</th> <th>Название</th>
    </tr>
    <xsl:for-each select="movies/movie">
      <tr>
        <td><xsl:value-of  select="title"/></td>
        <td><xsl:value-of  select="plot"/></td>
      </tr>
    </xsl:for-each>
</table>
</body> </html>
</xsl:template>
</xsl:stylesheet>
```

Элемент `<xsl:for-each>` определяет местоположение элементов в XML-документе и повторяет часть шаблона для каждого. Результат работы этой таблицы стилей показан на рис. 25.4.

Название	Жанр
Бриллиантовая рука	Приключения
Бриллианты навсегда	Шпионские приключения

Рис. 25.4. Результат форматирования XML-документа с помощью XSL-таблицы, представленной в листинге 25.4

Сортировка данных в выходном документе

XSL-шаблоны можно применять и для сортировки элементов в XML-документе. Добавим элемент `<xsl:sort>` внутрь элемента `<xsl:for-each>` в XSL-файле (листинг 25.5).

Листинг 25.5. Сортировка выходных данных

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html> <body>
  <table border="1">
    <tr>
      <th>Герой</th> <th>Исполнитель</th>
    </tr>
    <xsl:for-each select="movies/movie/characters/character">
    <xsl:sort select="name"/>
      <tr>
        <td><xsl:value-of  select="name"/></td>
        <td><xsl:value-of  select="actor"/></td>
      </tr>
    </xsl:for-each>
</table>
</body> </html>
</xsl:template>
</xsl:stylesheet>
```

Выполнив этот пример, вы получите таблицу, как на рис. 25.5.

Герой	Исполнитель
Джеймс Бонд	Шон Коннери
Лелик	Анатолий Папанов
Подруга Бонда	Джилл Сент-Джон
Семен Семеныч	Юрий Никулин

Рис. 25.5. Результат форматирования XML-документа с помощью XSL-таблицы, представленной в листинге 25.5

Выберем теперь только один фильм, задав фильтр в элементе `<xsl:for-each>` XSL-файла (не забывайте о чувствительности значений атрибута к регистру):

```
<xsl:for-each select="movies/movie[title='Бриллиантовая рука']">
```

Допустимыми операциями фильтрования являются:

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

Таблица стилей примет такой вид (листинг 25.6).

Листинг 25.6. Фильтрация данных

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html> <body>
  <table border="1">
    <tr>
      <th>Жанр</th>
    </tr>
    <xsl:for-each select="movies/movie[title='Бриллиантовая рука']">
      <tr>
        <td> <xsl:value-of  select="plot"/> </td>
     </tr>
     </xsl:for-each>
</table>
</body> </html>
</xsl:template>
</xsl:stylesheet>
```

А в браузере вы увидите аналогичное тому, что представлено на рис. 25.6.

Жанр
Приключения

Рис. 25.6. Результат форматирования XML-документа с помощью XSL-таблицы, представленной в листинге 25.6

Язык преобразования XSLT

Во время XSL-преобразования XSLT-парсер считывает XML-документ и таблицу стилей XSLT. На основе инструкций, которые парсер находит в таблице стилей XSLT, он вырабатывает новый XML-документ или его фрагмент.

Древовидная структура XML-документа

Каждый правильный XML-документ рассматривается парсером как дерево. Дерево — это структура данных, образованная соединенными между собой узлами, ветвящимися от верхнего узла, он называется *корнем*. Корень соединен с вложенными в него узлами, каждый из них может, в свою очередь, содержать вложенные узлы и т. д. Узлы, у которых нет собственных вложенных узлов, называются *листьями*.

Каждый узел дерева вместе со своими дочерними узлами тоже образует дерево. Элемент, непосредственно соединенный с корнем, называется корневым.

В XSLT элементы, атрибуты, пространства имен, процессуальные инструкции и комментарии считаются узлами. Кроме того, корень документа отличается от корневого элемента. XSLT-процессоры моделируют XML-документ как дерево, образованное семью видами узлов:

- корень;
- элементы;
- текст;
- атрибуты;
- пространства имен;
- процессуальные инструкции;
- комментарии.

Значения узлов XML-документа зависят от их типов:

- значение элементного узла — это весь текст между начальным и конечным тегом данного элемента, исключая вложенные в данный элемент теги;
- значением корня является значение корневого элемента;
- значение атрибута указывается в кавычках;
- значением текста является сам узел;
- значение комментария — текст комментария без знаков `<!--` и `-->`;
- значение пространства имен — URI этого пространства;
- значение процессуальной инструкции — это представленные в ней данные. Разграничители процессуальной инструкции `<?` и `?>` в значения не включаются.

На входе XSL-преобразования должен быть XML-документ. Нельзя применить XSLT для трансформации не XML-форматов, например, таких как PDF, TeX, Microsoft Word и т. д. Однако XSLT может работать с теми HTML-документами, которые удовлетворяют условиям правильности XML.

Шаблоны в таблицах стилей XSLT

XSLT-документ содержит правила-шаблоны. Правило-шаблон включает в себя образец, который определяет соответствующие ему узлы дерева. Когда XSLT-парсер преобразует XML-документ, он двигается по дереву XML-документа и по очереди проверяет каждый узел. Если парсер обнаруживает, что некоторый узел соответствует образцу одного из правил-шаблонов таблицы стилей, он выводит этот шаблон в выходной документ. Обычно шаблоны содержат разметку, какие-то новые данные, другие данные копируются из исходного XML-документа.

Для описания правил, шаблонов и образцов в XSLT используется язык XML. Корневым элементом XSLT-документа является либо элемент `<xsl:stylesheet>`, либо `<xsl:transform>`, которые находятся в пространстве имен `http://www.w3.org/1999/XSL/Transform`. По договоренности это пространство имен отображается в префиксе `xsl`.

Правила по умолчанию для обработки XML-документа стилевыми таблицами

В XSLT имеется несколько правил-шаблонов, которые по умолчанию неявно включаются во все таблицы стилей. В совокупности эти правила означают, что если применить к XML-документу пустую таблицу стилей, в которой нет ничего, кроме пустого элемента `<xsl:stylesheet>` или `<xsl:transform>`, из исходного документа в выходной копируются все значения элементных узлов, содержащиеся в исходном документе. Но в выходном документе будет отсутствовать разметка.

Это правила чрезвычайно низкого приоритета, поэтому правила с любыми другими соответствиями будут иметь приоритет над правилами по умолчанию. Для того чтобы парсер использовал таблицу стилей при обработке XML-документа, следует указать во второй строке XML-документа (файл movie.xml — см. листинг 25.1) ссылку на файл таблицы стилей:

```
<?xml-stylesheet type="text/xsl" href="patterns.xsl"?>
```

Таблица же patterns.xsl может содержать только правила по умолчанию:

```
<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Сохраните все эти файлы и посмотрите, как будет выглядеть movie.xml после обработки таблицей patterns.xsl в браузере.

Использование шаблонов в таблице стилей

Каждое правило-шаблон задается элементом `<xsl:template>`. Искомый образец размещается в атрибуте `match` элемента `<xsl:template>`.

Существуют различные способы преобразования XML-документов с помощью XSLT в другие форматы, например в HTML:

- способ первый: XML-документ и связанная с ним таблица стилей отправляются клиенту (Web-браузеру), который преобразует документ как указано в таблице стилей, и после этого представляет результат пользователю. В этой главе мы будем использовать этот способ;
- способ второй: сервер применяет таблицу стилей XSLT к XML-документу и преобразует XML-документ в другой формат (обычно в HTML). После этого результат отправляется клиенту (Web-браузеру). Так работает PHP-сценарий, мы займемся этим в *главах 30* и *31*, посвященных PHP-расширениям SimpleXML и функциям DOM.

Когда XSLT-процессор начинает считывать исходный документ, первый узел, с которым он сталкивается, — это корневой узел.

Вот пример шаблона, который будет применяться к корневому узлу исходного дерева:

```
<xsl:template match="/">
  <html>
    <head> </head>
    <body> </body>
```

```
  </html>
</xsl:template>
```

Приведенное правило относится к корневому узлу, и в нем XSLT-процессору дается указание вывести следующий текст:

```
<html>
  <head> </head>
  <body> </body>
</html>
```

Если вы оставите в таблице стилей XSLT только приведенный шаблон, то на выходе получите только эти шесть тегов. Так произойдет потому, что в этом шаблоне отсутствуют инструкции, заставляющие преобразователь двигаться далее по дереву и искать другие соответствия с шаблонами таблицы стилей.

Чтобы задействовать содержимое дочерних узлов, нужно рекурсивно обрабатывать узлы всего XML-документа. Элемент, предназначенный для этого, — `<xsl:apply-templates>`. Включая его в шаблон, вы даете указание парсеру сравнить каждый дочерний элемент исходного элемента с другими шаблонами в таблице стилей и, если соответствие обнаружено, вывести шаблон для этого узла в выходной документ. Шаблон узла, соответствие с которым обнаружено, тоже может содержать элементы `<xsl:apply-templates>` для поиска соответствий с дочерними узлами уже этого узла. Когда процессор обрабатывает узел, он рассматривает его как целое дерево.

Будем использовать следующую таблицу стилей XSLT (листинг 25.7).

Листинг 25.7. Таблица стилей XSLT

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
       <xsl:apply-templates/>
    </html>
  </xsl:template>

  <xsl:template match="movies">
    <body>
       <xsl:apply-templates/>
    </body>
  </xsl:template>

  <xsl:template match="movie">
     Описание фильма
  </xsl:template>
</xsl:stylesheet>
```

Все, что мы увидим в браузере в результате применения нашей таблицы стилей, — это дважды повторенная фраза "Описание фильма".

Вот что происходит, когда эта таблица стилей применяется к документу:

1. Корневой узел сравнивается со всеми правилами-шаблонами таблицы стилей. Он соответствует первому шаблону. На выход передается тег `<html>`.
2. Элемент `<xsl:apply-templates>` дает указание парсеру обрабатывать дочерние узлы корневого узла исходного документа.
3. Первый дочерний узел, процессуальная инструкция `<xml-stylesheet>`, сравнивается с правилами-шаблонами. Она не соответствует ни одному из них, так что на выход ничего не передается.
4. Второй дочерний узел корневого узла, корневой элемент `<movies>`, сравнивается с правилами-шаблонами. Он соответствует второму правилу-шаблону. В выходной документ записывается тег `<body>`.
5. Элемент `<xsl:apply-templates>` внутри элемента `<body>` заставляет парсер обрабатывать дочерние узлы элемента `<movies>`.
6. Его первый дочерний элемент, `<movie>`, сравнивается с правилами-шаблонами. Он соответствует третьему правилу-шаблону. На выход передается текст "Описание фильма".
7. Второй дочерний элемент, `<movie>`, сравнивается с правилами-шаблонами. Он тоже соответствует третьему правилу-шаблону. На выход передается текст "Описание фильма".
8. На выход передается закрывающий тег `<body>`.
9. На выход передается закрывающий тег `<html>`.

Обработка завершена.

Атрибут *select*

Вместо текста "Описание фильма" в выходном документе из предыдущего примера интереснее было бы написать название фильма, которое задается дочерним элементом `<title>`. Для этого нужно задать шаблоны, которые должны применяться к дочерним элементам `<title>` элемента `<movie>`. Чтобы отобрать не все дочерние узлы данного узла, а лишь некоторый их набор, нужно снабдить `<xsl:apply-templates>` атрибутом `select`, выделяющим нужные дочерние узлы, например:

```
<xsl:template match="movie">
   <xsl:apply-templates select="title"/>
</xsl:template>
```

Если теперь заменить в предыдущей таблице стилей слова "Описание фильма" на шаблонное правило `<xsl:apply-templates select="title"/>` и применить таблицу к документу, то мы получим следующий результат:

```
Бриллиантовая рука Бриллианты навсегда
```

Вычисление значения узла с помощью элемента <xsl:value-of>

Элемент <xsl:value-of> вычисляет некоторое значение и копирует полученный результат в выходной документ. Атрибут select элемента <xsl:value-of> определяет, какое именно значение вычисляется.

Допустим, вы хотите заменить текст "Описание фильма" именем фильма, заданным элементом <title>. Вы можете заменить вот так:

```
<xsl:template match="movie">
  <xsl:value-of select="title"/>
</xsl:template>
```

Результат преобразований будет точно таким же, как в предыдущем пункте.

Соответствие именам элементов

Как указывалось ранее, большая часть шаблонов содержит имя элемента, указывающее на все элементы с таким именем. Например, следующий шаблон соответствует элементам <movie> и выводит содержимое их дочерних элементов <plot> полужирным шрифтом:

```
<xsl:template match="movie">
  <b><xsl:value-of select="plot"/></b>
</xsl:template>
```

Появление этого шаблона в таблице приведет к выводу в браузер такого результата:

Приключения Шпионские приключения

Поставьте вместо имени узла символ точки, и вы получите на выходе все значение элементного узла <movie>. Поскольку в нашем примере шаблон применяется к обоим элементам <movie>, то из документа извлекаются значения обоих элементов <movie>. То есть символ точки означает требование вывести все значение узла.

Соответствие дочерним узлам

При использовании атрибута match вы не ограничены только непосредственными дочерними узлами текущего узла. Для выбора других элементов можно использовать символ /. Один этот символ соответствует корневому узлу, но его можно размещать между двумя именами элементов, это означает, что второй элемент является дочерним элементом первого.

При использовании этой конструкции в элементах <xsl:template> можно выбирать только некоторые элементы данного вида. Таблица в следующем примере заставит вывести жирным шрифтом только значения элементов <plot>, являющихся дочерними по отношению к элементам <movie> (листинг 25.8).

Листинг 25.8. Отбор дочерних узлов

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
     <head><title>Дочерние узлы</title></head>
     <body><xsl:apply-templates select="movies"/></body>
  </html>
</xsl:template>

<xsl:template match="movie/plot">
  <tr>
    <td><strong><xsl:value-of  select="."/></strong></td>
  </tr>
</xsl:template>

</xsl:stylesheet>
```

Поиск потомков

Иногда бывает легче пропустить промежуточные узлы и просто отбирать все элементы определенного типа вне зависимости от того, являются ли они непосредственными дочерними элементами, внуками или правнуками или еще более отдаленными потомками. Двойной слэш (`//`) указывает на всех потомков элемента, независимо от их глубины. Например, следующее правило-шаблон будет применяться ко всем элементам-потомкам `<actor>` элемента `<movies>`, независимо от глубины, на которой они находятся (листинг 25.9).

Листинг 25.9. Поиск потомков любого поколения

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="movies//actor">
     <i><xsl:value-of select="."/></i>
  </xsl:template>
</xsl:stylesheet>
```

То есть имена актеров будут писаться курсивом.

Следующая таблица отыщет только элементы `<name>`, где бы в документе они ни находились, и выведет их значения курсивом (листинг 25.10).

Листинг 25.10. Поиск элемента с помощью символов //

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="//name">
    <i><xsl:value-of  select="."/></i>
  </xsl:template>
</xsl:stylesheet>
```

Соответствие атрибутам

Перед именем атрибута, который вы хотите отобрать, надо поставить символ `@`. Например, следующее правило-шаблон соответствует атрибутам `stars` и выводит их значение курсивом:

```
<xsl:template match="@stars">
  <i><xsl:value-of  select="."/></i>
</xsl:template>
```

Однако если просто добавить это правило к таблице стилей, значения этих атрибутов в выходном документе автоматически не появятся, поскольку атрибуты не являются дочерними узлами содержащих их элементов. Таким образом, по умолчанию, когда XSLT-процессор двигается по дереву, он не видит атрибутных узлов. Их нужно обрабатывать, ясно прописывая их в подходящем значении атрибута `select` элемента `<xsl:apply-templates>` (листинг 25.11).

Листинг 25.11. Поиск атрибутов

```
<?xml  version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/movies">
    <html> <body>
       <xsl:apply-templates/>
    </body> </html>
  </xsl:template>

  <xsl:template match="movie">
    <xsl:value-of  select="."/>
    <xsl:apply-templates  select="@id"/>
  </xsl:template>

  <xsl:template match="@id">
     <b><xsl:value-of  select="."/></b>
  </xsl:template>
</xsl:stylesheet>
```

Применение этой таблицы приведет к появлению в выходном документе не только значений элементных узлов, но и значений атрибутов `id`.

Добавление атрибутов в выходной поток

Теперь у нас задача, похожая на предыдущую: создать в выходном документе атрибут `href`, значение которого сформируем из данных входного документа.

Атрибут нам нужен для тега ссылки `<a>`, а указывать ссылка должна на файл, имя которого берем из значения элемента `<rating>` с добавлением расширения html (листинг 25.12).

Листинг 25.12. Добавление атрибутов в выходной поток

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="movie/characters/character/name">
     <a>
     <xsl:attribute name="href">
         <xsl:value-of  select="../actor"/>.html
     </xsl:attribute>
     <xsl:value-of  select="."/>
     </a>
  </xsl:template>
</xsl:stylesheet>
```

В результате действия этой стилевой таблицы на наш XML-документ в выходном документе имена героев фильмов будут являться текстом ссылок на файлы, а имена этих файлов совпадают с именами актеров-исполнителей.

Форматирование с помощью атрибута *mode*

Иногда бывает нужно включить одно и то же содержимое исходного документа в выходной документ несколько раз, отформатировав его по-разному. Решение проблемы заключается в том, чтобы придать каждому правилу свой атрибут `mode`. Тогда вы сможете выбрать применяемый шаблон, устанавливая атрибут `mode` элемента `<xsl:apply-templates>`.

Например, мы хотим создать страницу, на которой вначале названия фильмов перечислены в виде ссылок (т. е. внутри тега `<a> </a>`), а потом идут названия фильмов, выделенные жирным шрифтом, с их описанием. В следующем примере таблица стилей содержит два шаблона для поиска элемента `<movie>` с двумя разными значениями атрибута `mode`. Обработка элементов, соответствующих шаблону, должна вестись по двум правилам, описанным внутри двух тегов `<xsl:template match="movie">` с соответствующими значениями атрибута `mode` (листинг 25.13).

Листинг 25.13. Использование атрибута `mode`

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/movies">
      <html>
      <head><title>Modes</title></head>
      <body>
      <h2>Ссылки на фильмы</h2>
      <!-- Задаем атрибут mode = "short" в теге начала шаблона -- >
      <ul> <xsl:apply-templates select="movie" mode="short"/> </ul>
      <h2>Описание фильмов</h2>
      <!-- Задаем атрибут mode = "full" в теге начала шаблона -- >
      <xsl:apply-templates select="movie" mode="full"/>
      </body> </html>
  </xsl:template>

  <!-- Правило обработки входных элементов для mode = "short" -- >
  <xsl:template match="movie" mode="short">
     <li><a>
     <!-- Создаем атрибут для ссылки -- >
      <xsl:attribute name="href">#
          <xsl:value-of select="title"/>
      </xsl:attribute>
      <xsl:value-of  select="title"/>
      </a></li>
  </xsl:template>
  <!-- Правило обработки входных элементов для mode = "full" -- >
  <xsl:template  match="movie"  mode="full">
     <h3>
     <xsl:attribute name="plot">
          <xsl:value-of  select="title"/>
     </xsl:attribute>
     <xsl:value-of  select="title"/>
     </h3>
     <p>
     <xsl:value-of  select="plot"/>
     </p>
  </xsl:template>
</xsl:stylesheet>
```

Глава 26

Применение XPath при обработке XML-документов

Язык XPath является частью XSL и состоит из набора правил для выделения частей XML-документа. XPath используется для поиска XML-элементов в документе с помощью выражений в виде путей (path).

Все работает примерно так: на языке XPath пишется путь до нужного XML-элемента. Спросив, как найти дорогу к нужному дому в городе, вы можете услышать: "Отсюда идите прямо до третьего перекрестка". Аналог такого объяснения на XPath может быть записан как путь от текущего XML-элемента к элементу-потомку в третьем поколении. Этот путь используется XSL-таблицей для поиска, а потом и для обработки XML-элемента. Выражения на языке XPath — это пути до искомого XML-элемента.

Рассмотрим пример XML-документа (листинг 26.1).

Листинг 26.1. Исходный XML-документ

```
<?xml version="1.0" encoding="utf-8"?>
<note>
  <date>
    <day>01</day>
    <month>01</month>
    <year>2011</year>
  </date>
  <to id="25">Константин</to>
  <to id="26">Ольга</to>
  <from>Владимир</from>
  <heading>Напоминалка</heading>
  <body>У меня день рождения!</body>
</note>
```

На языке XPath можно построить выражения для обозначения элементов этого XML-документа:

- `/note` — это выражение XPath указывает на корневой элемент;
- `/note/to` — выражение XPath выделяет все элементы `to`, являющиеся дочерними элементами по отношению к элементу `note`.

Если выражение начинается со слэша (/), оно представляет запись абсолютного пути к элементу, т. е. пути от корня документа. Если слэша в начале выражения нет, то речь идет об относительном пути — пути от текущего элемента.

Если выражение начинается с двойного слэша (//), тогда в документе выделяются все элементы, которые удовлетворяют критерию остальной части выражения, даже если они размещены на различных уровнях XML-документа.

Следующее выражение XPath выделяет все элементы <title> в документе: `//title`.

Звездочка (*) применяется для выделения любых XML-элементов. Следующее выражение XPath выделяет все элементы <month>, которые являются внуками элемента <movies>, чьими бы детьми они не были: `/note/*/month`.

Следующее выражение XPath выделяет все элементы документа: `//*`.

Выделение ветвей

Используя в XPath-выражении квадратные скобки, можно еще точнее указать элемент.

Следующее выражение XPath выделяет первый дочерний элемент элемента <to>, который является дочерним по отношению к <note>: `/note/to[1]`.

Следующее выражение XPath выделяет все элементы <date>, которые имеют дочерний элемент <year> со значением 2011: `date[year=2011]`.

Пример XSL-таблицы, выводящий в браузер только "01 01 2011" — жирным шрифтом и только дату — представлен в листинге 26.2.

Листинг 26.2. Таблица преобразования с выделением дочернего элемента

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <head>
      <title>Пример XPath</title>
    </head>
    <body>
      <xsl:apply-templates select="note"/>
    </body>
  </html>
</xsl:template>
<xsl:template match="note">
  <b><xsl:value-of  select="date[year=2011]"/></b>
</xsl:template>
</xsl:stylesheet>
```

Выделение нескольких путей

Используя в XPath-выражении оператор |, можно одновременно выделить несколько путей. Это похоже на ветвление в регулярных выражениях — мы ищем любой элемент из нескольких предложенных.

Следующее выражение XPath выделяет все элементы <day> и <month>, которые являются дочерними элементами элементов <date> (которые, в свою очередь, являются дочерними элементами элемента <note>):

```
/note/date/day | /note/date/month
```

Следующее выражение XPath выделяет в документе все элементы day и month:

```
//day | //month
```

Следующее выражение XPath выделяет в документе все элементы day, month и year:

```
//day | //month | //year
```

Выделение атрибутов

Для обозначения атрибутов в XPath также применяется префикс @.

Следующее выражение XPath выделяет все атрибуты с именем id: //@id.

Выражение XPath может выделить все элементы <to>, у которых есть атрибут id: //to[@id].

Пример XSLT-таблицы с использованием такого выражения приведен в листинге 26.3.

Листинг 26.3. Таблица XSLT с выделением атрибутов

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
<xsl:template match="//to[@id]">
  <b><xsl:value-of  select="." /></b>
</xsl:template>
</xsl:stylesheet>
```

В браузере элементы <to>, имеющие атрибут id, будут выделены жирным шрифтом.

Следующее выражение XPath выделяет все элементы <to>, у которых есть атрибут id, имеющий значение '25': //to[@id='25'].

Оси и проверки узлов

Путь в выражении XPath состоит из шагов:

```
шаг1 / шаг2 / шаг3
```

Парсер читает выражение для пути и сравнивает его с разбираемым фрагментом XML-документа. Так же как курсор указывает на текущий символ в окне текстового редактора, так существует и текущий XML-узел в разбираемом документе, разбором которого в данный момент занимается парсер. Шаги анализируются по очереди слева направо. Каждый шаг сравнивается с узлами в текущем наборе узлов.

Ось задает набор узлов относительно текущего узла. Описания осей даны в табл. 26.1. Для определения, найден ли интересующий узел, в пределах данной оси используется *проверка* узла — проверка имени или типа узла.

Таблица 26.1. *Оси выражений XPath*

Имя оси	Описание
`ancestor`	Содержит всех предков текущего узла. Эта ось всегда включает в себя корневой узел, если только текущий узел сам не является корневым
`ancestor-or-self`	Содержит текущий узел и всех его предков
`attribute`	Содержит все атрибуты текущего узла
`child`	Содержит все дочерние узлы данного узла
`descendant`	Содержит всех потомков текущего узла. Эта ось не содержит атрибутные узлы или узлы пространств имен
`descendant-or-self`	Содержит текущий узел и всех его потомков
`following`	Содержит все узлы документа, расположенные после закрывающего тега данного узла
`following-sibling`	Содержит все узлы (все потомки одного и того же родительского узла), идущие после данного узла. Если текущий узел является атрибутным или узлом пространства имен, эта ось окажется пустой
`namespace`	Содержит все узлы пространств имен текущего узла
`parent`	Содержит родительский узел текущего узла
`preceding`	Содержит все узлы документа, расположенные перед открывающим тегом данного узла
`preceding-sibling`	Содержит все узлы-братья (все потомки одного и того же родительского узла), идущие перед данным узлом. Если текущий узел является атрибутным или узлом пространства имен, эта ось окажется пустой
`self`	Содержит текущий узел

Синтаксис шага с использованием оси такой:

```
axisname::nodetest[predicate]
```

Здесь *axisname* — имя оси, *nodetest* — проверка узла, а *predicate* — условие.

Пример из листинга 26.4 демонстрирует, как найти следующий элемент того же поколения, что и текущий элемент.

Листинг 26.4. Использование осей в выражениях XPath

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <body>
      <title>XPath</title>
      <xsl:apply-templates select="note"/>
    </body>
  </html>
</xsl:template>

<xsl:template match="note">
  <h1>Элемент-собрат</h1>
  <xsl:apply-templates  select="//heading" />
</xsl:template>

<xsl:template  match="heading">
  <p> <xsl:value-of select="following-sibling::body"/> </p>
</xsl:template>
</xsl:stylesheet>
```

Сокращенная запись путей

Пути могут записываться в сокращенной форме. Допустимые сокращения приведены в табл. 26.2. Наиболее важное сокращение — из шага можно убирать запись оси `child`.

Таблица 26.2. *Сокращенная запись выражений XPath*

Сокращение	Значение	Пример
`name`	`child::name`	`"date"` является сокращенной записью `"child::date"`
`@`	`attribute::`	`"to[@ide="22"]"` является сокращенной записью `"child::to[attribute::id="22"]"`
`.`	`self::node()`	`".//year"` является сокращенной записью `"self::node()/descendant-or-self::node()/child::year"`

Таблица 26.2 *(окончание)*

Сокращение	Значение	Пример
`..`	`parent::node()`	"`../date`" является сокращенной записью "`parent::node()/child::date`"
`//`	`/descendant-or-self::node()/`	"`//year`" является сокращенной записью "`/descendant-or-self::node()/child::year`"

В следующем примере сначала ищем элемент `<year>`, а потом от этого элемента выходим на два уровня выше и там выбираем элемент `<from>` (листинг 26.5).

Листинг 26.5. Сокращенная запись осей

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/note">
<html>
  <body>
    <table>
      <th>Element</th>
      <xsl:apply-templates select="date"/>
    </table>
  </body>
</html>
</xsl:template>

<xsl:template match="date">
  <xsl:apply-templates select="year"/>
</xsl:template>

<xsl:template match="year">
  <tr>
    <td><xsl:value-of select="../../from"/> </td>
  </tr>
</xsl:template>
</xsl:stylesheet>
```

Функции языка XPath

XPath может работать с числами и строками, а также с арифметическими операциями. Кроме того, библиотека функций XPath содержит набор функций для преобразования данных. Функции различных типов описаны в табл. 26.3—26.6.

Таблица 26.3. *Функции для определения набора узлов*

Имя	Описание
`count()`	Возвращает число выделенных элементов
`id()`	Выделяет элементы по их уникальному идентификатору `ID`
`last()`	Возвращает число, равное количеству узлов в выделенном наборе узлов
`local-name()`	Возвращает локальную часть полного имени узла из набора узлов, который идет в документе первым
`name()`	Возвращает имя элемента
`namespace-uri()`	Возвращает URI пространства имен полного имени узла из набора узлов, который идет в документе первым
`position()`	Возвращает число, равное номеру позиции выделенного узла

Таблица 26.4. *Строковые функции*

Имя	Описание	Пример и результат
`concat()`	Возвращает конкатенацию (объединение) своих аргументов	`concat('The',' ','XML')` Результат: `'The XML'`
`contains()`	Возвращает `true`, если первая строка содержит вторую строку, иначе возвращает `false`	`contains('XML','X')` Результат: `true`
`normalize-space()`	Удаляет из строки ведущие и замыкающие пробелы	`normalize-space(' The XML')` Результат: `'The XML'`
`starts-with()`	Возвращает `true`, если первая строка начинается со второй строки, иначе возвращает `false`	`starts-with('XML','X')` Результат: `true`
`string()`	Преобразует объект в строку	`string(3.14)` Результат: `'3,14'`
`string-length()`	Возвращает число символов в строке	`string-length('Beatles')` Результат: `7`
`substring()`	Возвращает подстроку	`substring('Beatles',1,4)` Результат: `'Beat'`
`substring-after()`	Возвращает подстроку после заданной подстроки	`substring-after('12/10','/')` Результат: `'10'`

Таблица 26.4 *(окончание)*

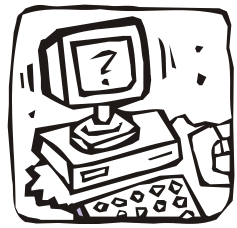
Имя	Описание	Пример и результат
`substring-before()`	Возвращает подстроку перед заданной подстрокой	`substring-before('12/10','/')` Результат: `'12'`
`translate()`	Заменяет буквы в строке	`translate('12:30',':','!')` Результат: `'12!30'`

Таблица 26.5. *Числовые функции*

Имя	Описание	Пример и результат
`ceiling()`	Возвращает округленное вверх значение аргумента	`ceiling(3.14)` Результат: `4`
`floor()`	Возвращает округленное вниз значение аргумента	`floor(3.14)` Результат: `3`
`number()`	Преобразует свой аргумент к числовому типу	`number(price)`
`round()`	Возвращает целое число, ближайшее к значению аргумента. Если таких чисел два, то возвращается большее из них	`round(3.14)` Результат: `3`
`sum()`	Строчное значение каждого узла в наборе узлов преобразуется в число, возвращается сумма этих чисел	`sum(/cd/price)`

Таблица 26.6. *Логические функции*

Имя	Описание	Пример и результат
`boolean()`	Преобразует свой аргумент в булево значение	`boolean(0)` Результат: `false`
`false()`	Возвращает булево значение `false`	`number(false())` Результат: `0`
`not()`	Возвращает `true`, если аргумент имеет значение, и `false` — наоборот	`not(false())` Результат: `true`
`true()`	Возвращает булево значение `true`	`number(true())` Результат: `1`



Глава 27

Объектная модель документа

Если вы знакомы с JavaScript, значит, у вас уже есть представление о том, что такое объектная модель документа.

Объектная модель документа (Document Object Model, DOM) создается анализатором документа. В нашем случае анализатором является XML-парсер браузера или сценарий PHP.

Обычно DOM добавляется как слой между XML-анализатором и приложением, которому требуется информация из документа, т. е. имеется в виду, что анализатор берет данные из XML-документа и передает их в DOM. Затем DOM используется приложениями более высокого уровня. DOM — это программный интерфейс, он дает возможность создавать документы, двигаться по документу, перемещать, копировать и удалять его части, добавлять или изменять атрибуты.

Следовательно, чтобы написать приложение, которое имело бы доступ к XML-документу через DOM, необходимо установить на компьютере XML-анализатор и реализацию DOM.

В *главе 31* мы будем использовать встроенную в PHP 5 библиотеку DOM-функций, которая представляет собой реализацию интерфейса DOM.

Но сначала рассмотрим, как строится объектная модель XML-документа. Интерфейс DOM представляет приложению документ в упорядоченном структурированном виде. Процесс напоминает уборку квартиры, когда перемешанные вещи разбираются по ящикам, раскладываются по полкам, а в модели DOM эти полки и ящики к тому же имеют названия.

Дерево документа

Модель DOM не накладывает ограничений на структуру документа. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел содержит элемент, атрибут, текстовый, графический или любой другой объект. Узлы связаны между собой отношениями "родитель—потомок".

Рассмотрим объектную модель, создаваемую парсером в результате обработки следующего XML-документа (листинг 27.1).

Листинг 27.1. XML-документ

```
<parent>
  <child id="123"> Text node </child>
</parent>
```

DOM представляет собой дерево, отображающее структуру XML-документа.

Отдельные XML-элементы можно выделять из XML-документа, двигаясь по дереву узлов, получать к ним доступ через их номер или имя. Узлы могут относиться к различным типам. В табл. 27.1 перечислены основные типы узлов.

Таблица 27.1. *Типы узлов*

NodeType — тип узла	**NodeTypeString — строковое значение типа узла**	**NodeName — имя узла**	**NodeValue — значение узла**
1	element	Имя тега	null
2	attribute	Имя атрибута	Значение атрибута
3	text	#text	Содержимое узла
4	cdatasection	#cdatasection	Содержимое узла
8	comment	#comment	Текст комментария
9	document	#document	null

В документе, приведенном в листинге 27.1, есть два элементных узла — `parent` и `child`, один атрибутный узел — `id` и текстовый узел — текстовое содержимое узла.

Далее в этой главе приводятся сведения об объектной модели документа, соответствующей стандарту W3C, в том объеме, который потребуется для работы с библиотекой DOM-функций PHP 5.

Объект *Node*

Объект `Node` представляет любой узел дерева. Узел может быть элементным, текстовым или узлом любого другого типа. Все эти типы узлов, как и объекты вообще, имеют свойства и методы. Свойства и методы объекта `Node` описаны в табл. 27.2. и 27.3.

Таблица 27.2. *Свойства объекта* `Node`

Имя	**Описание**
attributes	Возвращает массив `NamedNodeMap` (карта именованных узлов), содержащий все атрибуты данного узла
childNodes	Возвращает `NodeList` — массив всех дочерних узлов данного узла
firstChild	Возвращает первый дочерний узел данного узла

***Таблица 27.2** (окончание)*

Имя	Описание
`lastChild`	Возвращает последний дочерний узел данного узла
`nextSibling`	Возвращает следующий узел, дочерний элемент того же родительского элемента
`nodeName`	Возвращает `nodeName` (имя узла) в зависимости от его типа
`nodeType`	Возвращает `nodeType` (тип узла) в виде числа
`nodeValue`	Возвращает или устанавливает значение данного узла в зависимости от его типа
`parentNode`	Возвращает родительский узел данного узла
`previousSibling`	Возвращает предыдущий дочерний элемент, обладающий тем же родительским элементом, что и данный узел

***Таблица 27.3.** Методы объекта `Node`*

Имя	Описание
`appendChild(`*`newChild`*`)`	Привязывает новый узел *`newChild`* как последний в списке дочерних узлов данного узла
`hasChildNodes()`	Возвращает значение `true`, если узел имеет хотя бы один дочерний узел
`insertBefore(`*`newNode`*`, `*`refNode`*`)`	Вставляет новый узел *`newNode`* перед существующим узлом *`refNode`*
`removeChild(`*`nodeName`*`)`	Удаляет узел *`nodeName`* (имя узла)
`replaceChild(`*`newNode`*`, `*`oldNode`*`)`	Заменяет старый узел *`oldNode`* новым узлом *`newNode`*

Пример. В документе, приведенном в листинге 27.1, узел `child` имеет следующие свойства:

```
nodeName = "child"
nodeType = 1
parentNode = "parent"
```

Например, метод `removeChild(child)`, вызванный для объекта `parent`, удалит элемент `child` из документа.

Объект *NodeList*

Объект `NodeList` представляет собой упорядоченный список узлов. Этот список может быть сформирован по-разному, например в него могут быть отобраны все вложенные в некоторый элемент узлы с помощью свойства `childNodes`, описанного в табл. 27.2. В объектной модели определен также ряд методов, возвращающих

наборы узлов, т. е. формирующих объект NodeList. Узлы в списке пронумерованы, нумерация начинается с нуля. В дальнейшем мы будем использовать свойство length объекта NodeList, возвращающее количество узлов в этом объекте, и метод item(*index*), возвращающий узел из списка NodeList, по номеру, переданному в качестве параметра *index*.

В примере из листинга 27.1 объект parent имеет список дочерних узлов, состоящий из одного узла child. Следовательно, длина списка length = 1.

Объект *Document*

Объект Document — это объект, описывающий корень дерева документа. Все остальные узлы являются childNodes (дочерними узлами) объекта Document.

Свойство documentElement возвращает корневой элемент документа. Методы объекта Document описаны в табл. 27.4.

Таблица 27.4. *Методы объекта* Document

Имя	Описание
createAttribute(*attributeName*)	Создает узел attribute с заданным именем атрибута
createElement(*tagName*)	Создает элемент с заданным именем тега
createProcessingInstruction(*target*, *text*)	Создает узел процессуальных инструкций processingInstruction, содержащий заданную цель и текст
getElementById(*id*)	Возвращает элемент, значение *id* которого указано в аргументе
getElementByTagName(*tagName*)	Возвращает заданный узел и все его дочерние узлы в виде списка узлов NodeList

Метод createElement("kinder") создаст пару тегов <kinder> в рассматриваемом документе. Но для того чтобы включить этот новый элемент в документ, потребуется еще применить метод appendChild("kinder") объекта Node.

Если надо найти элементный узел, соответствующий тегу <child>, надо вызвать метод getElementsByTagName("child").

Объект *Element*

Объект Element представляет элементные узлы документа. Если элементный узел содержит текст, этот текст будет представлен в текстовом узле.

Свойство tagname этого объекта содержит имя узла. В табл. 27.5 перечислены методы объекта Element.

Таблица 27.5. *Методы объекта* `Element`

Имя	Описание
`getAttribute(`*`attributeName`*`)`	Возвращает значение заданного атрибута
`getAttributeNode(`*`attributeName`*`)`	Возвращает заданный узел-атрибут как объект
`getElementsByTagName(`*`tagName`*`)`	Возвращает заданный узел и все его дочерние узлы в виде списка узлов `NodeList`
`removeAttribute(`*`attributeName`*`)`	Удаляет значение заданного атрибута. Если атрибут имеет значение по умолчанию, оно устанавливается
`removeAttributeNode(`*`attributeNode`*`)`	Удаляет заданный атрибутный узел. Если атрибутный узел имеет значение по умолчанию, этот атрибут вставляется
`setAttribute(`*`attributeName`*`, `*`attributeValue`*`)`	Вставляет новый атрибут
`setAttribute-Node(`*`attributeNodeName`*`)`	Вставляет новый атрибутный узел

Указанные методы позволяют получать значение атрибутов, создавать и удалять атрибуты тегов.

Объект *Attr*

Объект `Attr` возвращает атрибут элементного объекта как атрибутный узел. Объект `Attr` имеет те же самые свойства и методы, что и остальные узлы в целом. Свойства, специфичные для объекта `Attr`, перечислены в табл. 27.6.

Таблица 27.6. *Свойства объекта* `Attr`

Имя	Описание
`name`	Возвращает или устанавливает имя атрибута
`value`	Возвращает или устанавливает значение атрибута

Дерево документа начинается с корня — это основание дерева, объект `Document` (рис. 27.1). От корня формируется список узлов `NodeList`, в котором находится один элемент — корневой элемент `html`, являющийся элементным узлом (`Element`). К этому узлу подсоединяется список дочерних узлов, в котором находятся элементные узлы `head` и `body`. У каждого из этих узлов имеется список дочерних узлов, в котором есть текстовые узлы и карта (`NamedNodeMap`), содержащая список атрибутов.

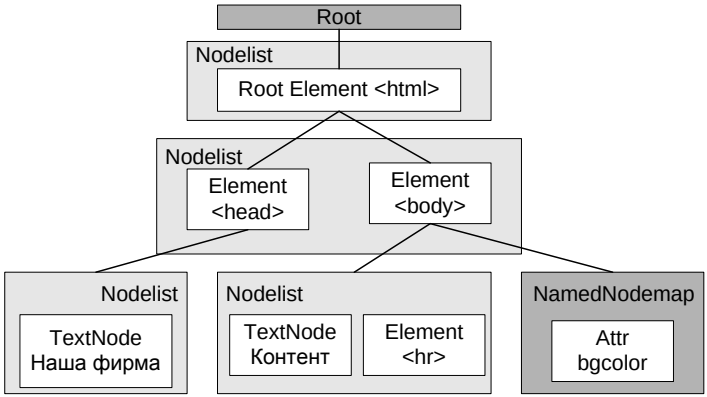
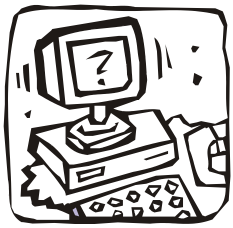


Рис. 27.1. Дерево документа в объектной модели

Модель DOM представляет собой универсальное средство анализа документа, отличаясь при этом изрядной сложностью. Как всегда, сложность и универсальность идут вместе. В *главе 31* мы увидим, как DOM реализована в языке PHP в расширении, содержащем классы для работы с XML-документами.



Глава 28

Новостная лента RSS

Возможно, что из всех форматов XML-документов Web-мастеру чаще всего приходится иметь дело с новостными лентами RSS. RSS — это семейство XML-форматов, предназначенных для описания лент новостей, анонсов статей и т. п. С помощью RSS можно публиковать любой материал, состоящий из нескольких частей. После того как информация преобразована в формат RSS, программа, обрабатывающая RSS-файлы, может извлекать из них информацию, а также добавлять новый материал.

В зависимости от версии стандарта RSS может расшифровываться по-разному:

- Rich Site Summary (RSS 0.9*x*) — обогащенная сводка сайта;
- RDF Site Summary (RSS 0.9 и 1.0) — сводка сайта с применением инфраструктуры описания ресурсов;
- Really Simple Syndication (RSS 2.*x*) — очень простое приобретение информации.

Версия стандарта 2.0 является продолжением ветки версии 0.9.

Многие современные браузеры умеют работать с RSS-лентами, среди них Mozilla Firefox, Mozilla Thunderbird, Opera, Microsoft Internet Explorer, начиная с 7-й версии. Кроме того, существуют специализированные приложения, собирающие и обрабатывающие информацию RSS-каналов.

Обычно с помощью RSS дается краткое описание новой информации, появившейся на сайте, и ссылка на ее полную версию. Интернет-ресурс в формате RSS называется *RSS-каналом* или *RSS-лентой*.

Пример новостной ленты представлен в листинге 28.1.

Листинг 28.1. Новостная лента в формате RSS 2.0

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
<channel>
  <title>HardwarePortal.ru: Новости HardwarePortal.ru</title>
  <link>http://www.hwp.ru</link>
  <description>Портал компьютерного &quot;железа&quot;.</description>
  <language>ru</language>
  <lastBuildDate>Mon, 28 Aug 2010 18:14:06 +0400</lastBuildDate>
```

```
  <webMaster>likeoff@mail.ru</webMaster>
  <image>
    <title>HardwarePortal.ru</title>
    <url>http://hwp.ru/img/logo.gif</url>
    <link>http://www.hwp.ru.ru</link>
    <width>100</width>
    <height>100</height>
    <description>HardwarePortal.ru</description>
  </image>
  <item>
    <title>Ноутбуки для поклонников английского футбола</title>
    <link>http://hwp.ru/scripts/news_show.php?5717</link>
    <description>Fujitsu Siemens представила ноутбуки</description>
    <pubDate>Mon, 28 Aug 2010 16:00:54 +0400</pubDate>
  </item>
  <item>
    <title>Canon обновляет семейство зеркалок EOS</title>
    <link>http://hwp.ru/scripts/news_show.php?5692</link>
    <description>Canon представила зеркальную камеру</description>
    <pubDate>Thu, 24 Aug 2010 15:57:32 +0400</pubDate>
  </item>
  <item>
    <title>Проектор со встроенным DVD-плеером</title>
    <link>http://hwp.ru/scripts/news_show.php?5685</link>
    <description>Toshiba представляет цифровой проектор </description>
    <pubDate>Thu, 24 Aug 2010 10:34:53 +0400</pubDate>
  </item>
</channel>
</rss>
```

На верхнем уровне RSS-документа находится элемент `rss`, который содержит обязательный атрибут `version`, указывающий версию документа. Обязательные элементы новостной ленты перечислены в табл. 28.1, а необязательные — в табл. 28.2. На уровень ниже от элемента `rss` находится единожды встречающийся элемент `channel` (канал), который содержит информацию о канале (метаданные) и о его содержании.

Таблица 28.1. *Обязательные элементы канала*

Элемент	Описание
`title`	Имя канала. Это имя будет указываться при формировании ссылки
`link`	URL HTML-сайта, соответствующего каналу
`description`	Предложение с описанием канала

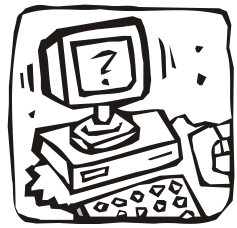
Таблица 28.2. *Необязательные элементы канала (неполный список)*

Элемент	Описание
language	Язык написания канала
copyright	Авторские права канала
managingEditor	Электронный адрес (e-mail) человека, отвечающего за содержание
webMaster	Электронный адрес (e-mail) человека, ответственного за техническое обеспечение канала
pubDate	Дата публикации сведений канала
lastBuildDate	Время последнего изменения сведений канала
category	Одна или несколько категорий, к которым принадлежит канал
generator	Строка, указывающая программу, с помощью которой создан канал

Элемент channel может содержать несколько элементов item. Статья (item) может содержать текст — совсем как статья в газете или журнале. Тогда ее описание (description) — это краткий анонс, а ссылка (link) указывает на полный текст истории. Статья может быть полным текстом и сама, тогда ее описание содержит текст (HTML-код с закодированными символами "больше", "меньше" и амперсанд), а ссылка и заголовок (title) могут отсутствовать. Все элементы внутри item не обязательны, но хотя бы один элемент title или description должен присутствовать. Дочерние элементы тега item представлены в табл. 28.3.

Таблица 28.3. *Дочерние элементы тега* `item`

Элемент	Описание
title	Заголовок статьи
link	URL статьи
description	Описание статьи
author	E-mail автора статьи
category	Включает статью в одну или несколько категорий
comments	URL страницы комментариев к статье
enclosure	Описывает тип вложения в статью
guid	Строка, уникально идентифицирующая статью
pubDate	Дата публикации статьи
source	RSS-канал, откуда пришла статья



Глава 29

Создание и анализ XML-документов средствами PHP. SAX-парсер

XML-документы представляют собой простой текст. Поэтому для их анализа могут быть использованы функции для работы с регулярными выражениями. Кроме того, в PHP 5 включен ряд расширений, предназначенных именно для работы с XML.

- Первым надо упомянуть семейство функций XML Parser, потому что оно появилось еще в версии PHP 4 и в настоящее время широко используется на сайтах для обработки новостных лент и других XML-документов. В этих функциях применяется подход к обработке документов, называемый SAX, который использует модель обработки событий.
- В PHP 5 появилось принципиально новое расширение для работы с XML — SimpleXML, про которое говорят, что оно должно кардинально изменить характер работы с XML.
- Расширение DOM-функций — это тоже принципиально новая разработка, не являющаяся прямым наследником расширения DOM XML в PHP 4, которое не соответствовало стандарту W3C.
- Еще одно полезное расширение поставляется в версии PHP 5 для Windows — семейство XSL-функций.

Информация, содержащаяся в RSS-ленте или любом другом XML-файле, может быть представлена на Web-странице только после обработки этого файла и извлечения из него фрагментов текста, содержащих требуемую информацию.

В этой главе мы рассмотрим применение расширений PHP для обработки XML-документов. Особое внимание будет уделено работе с различными кодировками символов. В настоящее время рекомендуем создавать Web-приложения в кодировке UTF-8, в частности потому, что внутреннее представление данных в PHP идет как раз в этой кодировке. Однако до сих пор русифицированные приложения часто создают документы в кодировке Windows-1251. Часть функций осуществляет автоматическое преобразование данных в кодировку UTF-8, и разработчику не приходится самому заботиться об этом, но преобразованные и обработанные в сценарии данные будут переданы в выходной документ в кодировке UTF-8, какого бы типа ни был этот документ. Нам самим придется позаботиться о перекодировке, если она необходима.

SAX

SAX (Simple Application Programming Interface for XML) — это подход к обработке XML-документов, основанный на анализе дерева документа и обработке событий. SAX не является стандартом W3C. Этот метод предназначен для чтения данных из XML-документа, но не для создания новых документов. Он относится к семейству потоковых парсеров, т. е. данные из XML-документа потоком передаются в парсер и по кусочкам обрабатываются в нем. Такая обработка не позволяет вернуться назад по документу от точки парсирования или изменить что-либо в нем.

Создание парсера

SAX основан на событийном программировании. Программа-парсер разбирает XML-документ, вызывая функции-обработчики определенных событий. События происходят, когда в документе встречается открывающий или закрывающий тег. Событием также является нахождение парсером текстового содержания узла, для обработки которого тоже определена функция-обработчик.

Расширение expat, включенное в ядро PHP, позволяет создавать обработчики событий и в отличие от некоторых других расширений, предназначенных для работы с XML, не является исключительно объектно-ориентированным.

Функции обработки событий предназначены для работы с тегами, имена которых написаны заглавными буквами. Это преобразование парсер выполняет сам автоматически, в документе теги могут быть записаны в любом регистре.

Что касается кодировок, то приходится помнить, что есть две кодировки: кодировка входного документа и кодировка выходного документа. Внутреннее представление документа в PHP — всегда UTF-8. Начиная с версии PHP 5.3, следует создавать документы только в кодировке UTF-8. Общая тенденция развития как Интернета в целом, так и языка PHP, диктует нам, что о других кодировках пора уже забыть.

Напишем вначале простой сценарий, использующий обработку событий. Логика действий по созданию и использованию парсера отчасти напоминает создание сессий: сначала создается сессия с уникальным идентификатором, а потом в ней хранятся переменные. Так и в случае с парсером: вначале необходимо создать парсер, которому затем надо будет указать функции обработки событий:

```
$xml_parser = xml_parser_create();
```

Эта функция возвращает указатель на парсер, который используется другими функциями, работающими с XML.

Определение функций-обработчиков событий

Определяя функции-обработчики, возьмем простые примеры, не слишком полезные из-за своей простоты, но подходящие нам для демонстрации идеи.

Функция обработки открывающего тега должна принимать три аргумента. Первый — идентификатор парсера, второй — имя найденного стартового тега, третий — массив атрибутов для начального тега.

```
function startElement($parser, $name, $attrs)
{
  echo " Start tag $name <br />";
}
```

Функция обработки закрывающего тега принимает два аргумента: идентификатор парсера и имя найденного закрывающего тега.

```
function endElement($parser, $name)
{
  echo " End tag $name <br />";
}
```

Для применения функций обработки событий в парсере надо вызвать функции PHP для создания обработчика событий.

Функция `xml_set_element_handler()` задает обработчики событий, которые происходят, когда XML-парсер встречает открывающий или закрывающий тег. Первый ее аргумент — идентификатор парсера, второй — имя функции обработки открывающего тега, а третий — имя функции обработки закрывающего тега. Если вместо имен функций-обработчиков поставить `false`, то соответствующее событие будет проигнорировано.

Укажем имена функций-обработчиков нашему парсеру:

```
xml_set_element_handler($xml_parser, "startElement", "endElement");
```

Чтение и обработка XML-документа

Обработчики событий определены. Откроем файл с XML-документом:

```
$fp = fopen("note.xml","r");
```

Теперь надо читать файл по частям, каждая часть — 4 Кбайт:

```
$data = fread($fp, 4096)
```

Читаем фрагменты файла и передаем прочитанное парсеру до тех пор, пока не дойдем до конца файла. Для обработки фрагмента файла надо вызвать функцию:

```
xml_parse($xml_parser, $data, feof($fp))
```

Эта функция принимает указатель на парсер, фрагмент текста в переменной `$data` и условия завершения парсирования — достижение конца файла. Функция сканирует данные, вызывает подключенные программы обработки событий и возвращает `true` или `false` в зависимости от успешности выполнения.

По окончании работы освободим парсер, т. е. удалим указатель и освободим используемые ресурсы:

```
xml_parser_free($xml_parser);
```

Все вместе это выглядит так, как представлено в листинге 29.1.

Листинг 29.1. Простейший SAX-парсер XML-документа

```
<?php
function startElement($parser, $name, $attrs)
{
  echo "Start tag $name <br />";
}

function endElement($parser, $name)
{
  echo "End tag $name <br />";
}
$xml_parser = xml_parser_create();
xml_set_element_handler($xml_parser, "startElement", "endElement");
$fp = fopen("note.xml","r");
while ($data = fread($fp, 4096))
  xml_parse($xml_parser, $data, feof($fp));
xml_parser_free($xml_parser);
fclose($fp);
?>
```

Если применить этот парсер к следующему XML-документу:

```
<note>
  <to>Text</to>
</note>
```

то мы увидим в браузере:

```
Start tag
Start tag
End tag
End tag
```

Функции обработки текстового содержимого узла и обработки ошибок

Для обработки содержимого XML-элементов надо сделать следующий шаг — указать опции работы парсера (по стандарту буквы в тегах в XML-документе должны преобразовываться к верхнему регистру — это называется *case folding*):

```
xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, true);
```

Нужно также определить функцию-обработчик текстового содержимого XML-узла:

```
function characterData($parser, $data)
{
  echo $data;
}
```

и указать ее парсеру:

```
xml_set_character_data_handler($xml_parser, "characterData");
```

Функция `xml_set_character_data_handler()` задает обработчик для события, состоящего в том, что парсер встречает символьные данные в XML-документах. Символьные данные включают в себя текстовое содержимое узла и пробелы между тегами. Функция, указанная во втором аргументе, должна принимать два аргумента. Первый — идентификатор парсера, а второй — символьные данные.

Если же появится ошибка, например из-за того, что документ оказался синтаксически некорректным, то надо вызвать функции, оповещающие о неприятностях:

- `xml_get_error_code($xml_parser)` — возвращает число, являющееся кодом ошибки;
- `xml_error_string($code)` — возвращает текстовое сообщение, соответствующее коду ошибки;
- `xml_get_current_line_number($xml_parser)` — возвращает номер строки, в которой произошла ошибка.

SAX-парсер новостной ленты

Рассмотрим пример обработки RSS-ленты (листинг 29.2).

Листинг 29.2. SAX-парсер новостной ленты RSS

```
<?php
/* Инициализация переменных.
   Следующая переменная будет служить флагом, который указывает,
   находится ли парсер внутри тега item или нет. Только внутри
   этого тега находится информация, которую мы хотим получить
   из новостной ленты.
*/
$insideitem = false;
/*
   Следующие переменные будут использованы для хранения содержания
   соответствующих XML-элементов. Эти переменные внутри функций
   обработки будут объявлены глобальными.
*/
$tag = "";
$title = "";
$description = "";
$link = "";
/*
   Функция обработки открывающего тега принимает в качестве
   второго аргумента имя тега и проверяет, не является ли этот тег
   тегом item. В случае успеха устанавливает флаг в переменной
   $insideitem. Парсер преобразует имена тегов к верхнему регистру,
   поэтому их и приходится сравнивать в таком виде.
```

```
*/
function startElement($parser, $name, $attrs) {
  global $insideitem, $tag, $title, $description, $link;
  if ($insideitem) {
     $tag = $name;
  } elseif ($name == "ITEM") {
     $insideitem = true;
  }
}
/*
   Функция обработки закрывающего тега принимает в качестве второго
   аргумента имя тега, проверяет, не является ли этот тег тегом item.
   В случае успеха распечатывает данные, накопленные в переменных
   $title, $description и $link, а затем присваивает этим переменным
   значение "пустая строка". Перед распечаткой значения переменных
   обрабатываются функцией trim(), отсекающей пробелы в начале и
   конце строки, функцией htmlspecialchars(), обезвреживающей
   содержимое элементов, и функцией iconv(), преобразующей данные
   в кодировку Windows-1251.
*/
function endElement($parser, $name) {
   global $insideitem, $tag, $title, $description, $link;
  if ($name == "ITEM") {
    $description = htmlspecialchars(trim($description));
    $title = htmlspecialchars(trim($title));
    printf("<dt><b><a href='%s'>%s</a></b></dt>", trim($link), $title);
    printf("<dd>%s</dd>", $description);
    $title = "";
    $description = "";
    $link = "";
    $insideitem = false;
  }
}
/*
   Функция обработки символьного содержимого XML-элемента выполняет
   действия только в случае, когда событие происходит внутри тега item,
   это определяется по состоянию флага. В случае успеха анализируется,
   на каком теге произошло событие, и текстовое содержимое элемента,
   переданное функции во втором аргументе, добавляется к значению
   соответствующей переменной.
*/
function characterData($parser, $data) {
```

```
  global $insideitem, $tag, $title, $description, $link;
  if ($insideitem) {
     switch ($tag) {
        case "TITLE":
          $title .= $data;
          break;
        case "DESCRIPTION":
          $description .= $data;
          break;
        case "LINK":
          $link .= $data;
          break;
     }
  }
}
// Создание парсера
$xml_parser = xml_parser_create();
// Задание функций-обработчиков
xml_set_element_handler($xml_parser, "startElement", "endElement");
xml_set_character_data_handler($xml_parser, "characterData");
/*
   Открытие файла RSS-ленты. В случае ошибки сработает оператор die,
   вызвающий фукнцию sprintf(), которая распечатает сообщение об ошибке.
*/
$fp = fopen("rss2.xml","r") or die("Error reading RSS data.");
// Чтение четырех килобайтов данных из файла
while ($data = fread($fp, 4096))
   // Вызов парсера для обработки данных из файла.
   xml_parse($xml_parser, $data, feof($fp))
     or die(sprintf("XML error: %s at line %d",
     xml_error_string(xml_get_error_code($xml_parser)),
     xml_get_current_line_number($xml_parser)));
// Закрытие файла с данными после того, как он прочитан полностью.
fclose($fp);
// Освобождение ресурсов парсера.
xml_parser_free($xml_parser);
?>
```

Глава 30

Расширение SimpleXML в PHP 5

PHP 5 содержит принципиально новое расширение для обработки XML-документов — SimpleXML. Поддержка этого расширения осуществляется по умолчанию, т. е. вам не придется подключать какие-либо библиотеки к PHP.

Документация производителя на это расширение на сегодняшний день весьма лаконична. В PHP 5 определен класс `SimpleXMLElement`, экземпляром которого и становится загружаемый XML-документ. Некоторые из определенных в этом классе методов возвращают массивы, а некоторые — объекты того же класса `SimpleXMLElement`.

SimpleXML позволяет загрузить XML-документ, преобразовать его в объектную форму и получить доступ к значениям узлов XML. Объект класса `SimpleXMLElement` — это вовсе не DOM-представление XML-документа, но его можно преобразовать в объект DOM с помощью методов, объявленных в расширении SimpleXML.

Можно поменять содержание XML-элементов документа и вывести результат в виде XML в окно браузера или сохранить в файле.

С помощью только SimpleXML можно создавать XML-документы с нуля, но именно чтение и модификация уже созданных документов поражают своей простотой по сравнению с иными способами парсирования XML-данных.

Рассмотрим файл book_utf.xml (листинг 30.1).

Листинг 30.1. XML-документ в файле book_utf.xml

```
<?xml version="1.0" encoding="utf-8"?>
<kniga>
   <nazvanie lang="Russian">Волны гасят ветер</nazvanie>
   <avtor>Аркадий Стругацкий</avtor>
   <avtor>Борис Стругацкий</avtor>
</kniga>
```

Для того чтобы получить доступ к элементам этого документа, его необходимо загрузить и представить в виде дерева элементов. Если этот документ хранится в файле, то его загружают функцией `simplexml_load_file()`:

```
xml = simplexml_load_file('book_utf.xml');
```

Для записи длинного текста в PHP применяют так называемый heredoc-синтаксис, позволяющий с помощью специального идентификатора (символов <<<) объединить несколько строк воедино и сделать их значением переменной. Если документ хранится в виде heredoc-строки, то подойдет функция `simplexml_load_string()` (листинг 30.2).

Листинг 30.2. Чтение данных из строки

```
<?php
$string = <<<XML
<?xml version="1.0" encoding="utf-8"?>
<kniga>
<nazvanie lang="Russian">Волны гасят ветер</nazvanie>
<avtor>Аркадий Стругацкий</avtor>
<avtor>Борис Стругацкий</avtor>
</kniga>
XML;
$xml = simplexml_load_string($string);
?>
```

Обе эти функции выстраивают дерево документа и возвращают объект класса `SimpleXMLElement`, представляющий собой весь документ целиком. Собственно объектом является корневой элемент, который и помещается в переменную `$xml`. Но этот элемент содержит дочерние элементы `title` и `author`, которые представляются в виде массива. Причем получить к ним доступ можно, просто указав имя элемента:

```
echo $xml->nazvanie;
```

Несмотря на то, что и сам элемент `nazvanie` является объектом класса `SimpleXMLElement`, на экран будет выведено текстовое содержание узла `nazvanie`, т. е. будет произведено автоматическое преобразование объекта в строку.

Список дочерних элементов можно также представить в виде численно-индексированных массивов, где каждый неповторяющийся в документе элемент представлен в виде массива из одного элемента, а одноименные элементы — в виде массива с общей нумерацией. Прочитать текстовое значение элементов можно следующим образом:

```
echo $xml->nazvanie[0], $xml->avtor[0], $xml->avtor[1];
```

Следовательно, можно перебрать значения одноименных элементов и в цикле, указав псевдоним `$item` для перебираемых элементов:

```
foreach ($xml->avtor as $item)
  {
    echo $item;
  }
```

Доступ к значению атрибута осуществляется по имени этого атрибута:

```
echo $xml->nazvanie['lang'];
```

Опять же атрибут оказывается объектом, но преобразование типа для печати производится автоматически.

Для сравнения значений элементов также можно применить знакомые приемы, не заботясь о типах сравниваемых данных, поскольку и здесь осуществляется автоматическое преобразование типов.

```
$xml = simplexml_load_file('book_utf.xml');
if ($xml->nazvanie == "Волны гасят ветер") echo "Эту книгу мы искали!";
```

Изменение значений узлов также не представляет трудности:

```
$xml->nazvanie = "Жук в муравейнике";
```

После внесения изменений встает вопрос о сохранении XML-документов или их фрагментов в файле.

Для этого существует функция `asXML()`, которая возвращает строку, если вызывается без параметров. Указав же в качестве параметра имя файла, мы сохраним XML-документ на диск. В этом случае функция возвращает `true` или `false` в зависимости от успеха (листинг 30.3).

Листинг 30.3. Сохранение XML-данных в файл

```
<?php
$xml = simplexml_load_file('book_utf.xml');
$xml->nazvanie = "Жук в муравейнике";
$xml->asXML("book_new.xml");
?>
```

Аналогично можно сохранить и фрагмент документа, указав, например:

```
$xml->nazvanie->asXML("book_part.xml");
```

Открыв указанный файл, вы обнаружите там следующее:

```
<nazvanie lang="Russian">Жук в муравейнике</nazvanie>
```

Посмотрим теперь методы, предоставляющие более гибкие способы доступа к элементам. Существует метод для получения списка атрибутов элемента — `attributes()` (листинг 30.4).

Листинг 30.4. Применение метода `attributes()`

```
<?php
$xml = simplexml_load_file('book_utf.xml');
foreach($xml->nazvanie->attributes() as $a => $b) {
    echo $a,'="',$b,'"<br>';
}
?>
```

Вернемся к рассматривавшейся в *главе 28* новостной ленте, сохраненной в файле rss.xml. В этом документе есть несколько элементов `title`, находящихся на разных уровнях вложенности относительно корневого элемента. Применив SimpleXML-метод `xpath()`, можно распечатать их значения, указав путь до элементов в соответствии с синтаксисом языка XPath, как это представлено в листинге 30.5.

Листинг 30.5. Доступ к элементам с помощью метода `xpath()`

```
<?php
$xml = simplexml_load_file('rss.xml');
$zagolovki = $xml->xpath("//title");

while(list($nomer, $node) = each($zagolovki))
{
  echo $nomer." : ".$node,"<br>";
}
?>
```

Как видно из вывода, создаваемого циклом `while`, метод `xpath()` возвращает численно-индексированный массив элементов, соответствующих указанному пути, т. е. будут отобраны все элементы по имени `title`, наследниками какого бы поколения они ни являлись.

Можно и просто перебрать все элементы-наследники с помощью метода `children()`, который вернет их в виде массива элементов (листинг 30.6).

Листинг 30.6. Перебор элементов-наследников

```
<?php
$xml = simplexml_load_file('rss.xml');
foreach ($xml->channel->children() as $second_gen)
{
  echo $second_gen."<br>";
}
?>
```

Методы `addChild()` и `addAttribute()` позволяют добавить элемент и атрибут к существующему документу, как это показано в листинге 30.7.

Листинг 30.7. Добавление элемента и атрибута

```
<?php
$xml = simplexml_load_file('book_utf.xml');
$new_element = $xml->addChild('izdano', 'Мир');
$new_element->addAttribute('kogda', '2008');
$xml->asXML("more_info.xml");
?>
```

В сохраненном файле появится новый элемент:

```
<izdano kogda="2008">Мир</izdano>
```

Если же вы захотите создать новый XML-документ, то можно вызвать конструктор класса `SimpleXMLElement`, но для его работы необходим хотя бы один параметр — строка, представляющая собой правильный XML-документ, как показано в листинге 30.8.

Листинг 30. 8. Использование конструктора класса `SimpleXMLElement`

```
<?php
$xml = new SimpleXMLElement("<root></root>");
$new_element = $xml->addChild('child', 'Мир');
$new_element->addAttribute('data', '2008');
$xml->asXML("small.xml");
?>
```

Применение методов SimpleXML настолько увлекательно и выглядит так элегантно, что его не слишком портят даже извечные проблемы русскоязычного Web-мастера. Но все-таки следует отметить, что это расширение предназначено для работы в первую очередь с небольшими документами, чего никак нельзя сказать про героя следующей нашей главы — расширение DOM-функций.

Глава 31

Расширение DOM в PHP 5

Более широкие возможности обработки и форматирования XML-документов по сравнению с SimpleXML предоставляет библиотека DOM-функций, встроенная в ядро PHP 5. Это расширение позволяет использовать интерфейс прикладного программирования (API) DOM для работы с XML-документом. Модуль DOM-функций следует стандарту DOM Level 2, как сказано в документации, настолько точно, насколько это возможно. Интерфейс DOM является полностью объектно-ориентированным. Объекты, с которыми работает расширение DOM, были описаны в *главе 27*. Иными словами можно сказать так: в расширении DOM-функций определены классы, соответствующие стандарту DOM, свойства и методы этих классов соответствуют тому же стандарту. Тот или иной метод, неожиданно для новичка возникающий в сценарии, определен именно в стандарте DOM, там и следует искать полный перечень методов и свойств объектов, которые могут применяться в расширении PHP. Кроме того, в расширении определены некоторые собственные классы, например `DOMXpath`. Также есть и специфические свойства классов, явно добавленные для удобства работы с XML-документами. Отсюда следует совет внимательно изучить документацию по этому расширению.

Примечание

В PHP 4 имелось расширение DOM XML. Это расширение не соответствует стандарту W3C и в PHP 5 не вошло. По-видимому, решение о полной замене расширения было принято не сразу, поскольку в книгах, посвященных PHP 5, можно найти описание функций именно расширения DOM XML. Это совсем разные расширения, и попытка их одновременного использования успехом не увенчается.

Расширение PHP 5 DOM-функций не надо подключать в виде отдельной библиотеки! Если же вы попытаетесь подключить расширение DOM XML-функций к PHP 5, то расширение DOM перестанет корректно работать.

Применение DOM-функций для создания, модификации и чтения XML-документов

В настоящее время наиболее привлекательной и перспективной технологией создания Web-приложений, несомненно, является Ajax. Ajax — это технология создания сайтов, позволяющая ускорить взаимодействие клиента и сервера за счет

запросов, выполняемых сценарием JavScript незаметно для клиента. При этом ответ сервера зачастую состоит из XML-документа, обрабатываемого сценарием JavaScript, после чего данные ответа включаются в ранее загруженную Web-страницу.

Рассмотрение того, как именно JavaScript отправляет запрос на сервер и как потом обрабатывает ответ, выходит за рамки тематики данной книги.

Книга К. Дари, Б. Бринзаре, Ф. Черчез-Тоза и М. Бусика "AJAX и PHP. Разработка динамических веб-приложений" (СПб.: Символ-Плюс, 2009 г.) является лучшим способом знакомства с технологией Ajax для русскоязычного Web-мастера. Она великолепно вводит в курс дела, но при одном условии: читатель основательно знаком с расширением DOM-функций PHP 5.

Наиболее часто используемым средством генерации отклика сервера является сценарий PHP. Причем если касаться того, функции какого расширения PHP генерируют этот XML-документ, то теоретически это может быть и SimpleXML, но на практике с учетом сложности и объема необходимой работы вариант один — расширение DOM-функций.

Посмотрим, как можно создать XML-документ, используя расширение DOM-функций. В расширении имена классов соответствуют стандарту, но с добавкой в начале слова аббревиатуры DOM. Таким способом из стандартного имени объекта `Document` получился класс `DOMDocument`, а из имени объекта `NodeList` — имя класса `DOMNodeList`. Создадим сначала пустой объект `DOMDocument`:

```
$doc = new DOMDocument;
```

У конструктора этого класса есть два необязательных параметра — номер версии XML-документа и кодировка. По умолчанию в текущей версии PHP применяется кодировка UTF-8. Можно создать этот объект и так:

```
$doc = new DOMDocument("1.0","utf-8");
```

При этом необходимо помнить, что в версии PHP 5.3 применение иных кодировок вообще проблематично.

Создадим теперь корневой элемент `root` и подключим его к списку узлов. Как следует из объектной модели документа, для этого требуется выполнить два действия: вызвать метод создания XML-элемента `createElement()`, в качестве аргумента он принимает имя элемента, и метод `appendChild()`, который присоединяет вновь созданный элемент `$node` к списку элементов, дочерних по отношению к корневому элементу объекта `$doc`.

```
$node = $doc->createElement("root");
$newnode = $doc->appendChild($node);
```

Сохраним результат в виде XML-документа и выведем на экран с помощью метода `saveXML()`:

```
echo $doc->saveXML();
```

В браузере будет выведен созданный XML-документ:

```
<?xml version="1.0" encoding="utf-8" ?>
<root />
```

Можно сохранить документ в файле example.xml методом `save()`:

```
$doc->save('example.xml');
```

Сведем весь код вместе (листинг 31.1).

Листинг 31.1. Создание XML-документа с помощью DOM-функций

```
$doc = new DOMDocument("1.0","utf-8");
$node = $doc->createElement("root");
$newnode = $doc->appendChild($node);
echo $doc->saveXML();
```

Теперь создадим файл, включающий в себя PHP-дескриптор (процессуальную инструкцию на языке XML) `<?php ?>` с кодом PHP. Для этого нам потребуется метод `createProcessingInstruction()`. Кроме того, создадим атрибут тега с помощью метода `setAttribute()` (листинг 31.2).

Листинг 31.2. Создание HTML-документа с помощью DOM-функций

```
<?php
$dom = new DOMDocument("1.0");

// Создаем и подключаем к списку элементы html, head, title и body.
$root = $dom->appendChild($dom->createElement("html"));
$head = $root->appendChild($dom->createElement("head"));
$title = $head->appendChild($dom->createElement("title"));
$body = $root->appendChild($dom->createElement("body"));

// Создаем и подключаем процессуальную инструкцию с кодом PHP внутри.
$body->appendChild($dom->createProcessingInstruction('php',
                   'echo date("Y-m-d");?'));

// Создаем и подключаем элементы br и hr.
$body->appendChild($dom->createElement("br"));
$body->appendChild($dom->createElement("hr"));

// Создаем атрибут id со значением 123
$body->setAttribute('id', '123');

// Создаем и подключаем текстовый узел "Our_document",
// являющийся содержанием узла title.
$title->appendChild($dom->createTextNode("Our_document"));

// Сохраняем результат в файле 1.php
if ($dom->saveHTMLFile('1.php')) echo "Файл успешно создан!";
?>
```

Файл 1.php будет выглядеть так (кодировка выходного документа — UTF-8):

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Our_document</title>
  </head>
  <body id="123">
     <?php echo date("Y-m-d");?>
     <br>
     <hr>
  </body>
</html>
```

Вернемся к обработке RSS-ленты. Создадим сценарий обработки новостного канала с применением DOM-функций. Для этого используем метод `getElementsByTagName()`, возвращающий список элементов с именем, указанным в качестве аргумента (листинг 31.3).

Листинг 31.3. Обработка новостного канала RSS

```
<?php
// $dom — объект класса DOMDocument
$dom = new DOMDocument();

// Загрузим новостную ленту из файла
$dom->load("rss.xml");
/*
   Запросим списки узлов с именами title и description
   методом getElementsByTagName().
   Возвращаемые значения $titles и $description будут объектами
   класса NodeList.
   Метод просматривает документ полностью, проходя по всем
   поколениям дочерних элементов.
*/
$titles = $dom->getElementsByTagName("title");
$description = $dom->getElementsByTagName("description");
/*
   Метод item() объекта NodeList возвращает узел (объект Node).
   В качестве параметра метода указывается номер узла в списке.
   Распечатаем свойство "значения данного узла" (nodeValue)
   объекта Node.
*/
echo $titles->item(0)->nodeValue . "<br>";
echo $description->item(0)->nodeValue . "<br>";
```

```
/*
   Теперь нам потребуются все остальные узлы из списка
   (являющиеся потомками элементов item). В условии завершения
   цикла используется свойство length объекта NodeList.
*/
for ($i = 1; $i < $titles->length; $i++)
{
    echo "<b>".$titles->item($i)->nodeValue . "</b><br>";
    echo $description->item($i)->nodeValue . "<br>";
}
?>
```

Попробуем применить объект `DOMXPath` для анализа уже разбиравшегося XML-документа. Найдем в этом документе элемент `name`, в каком бы месте дерева документа он ни находился (листинг 31.4).

Листинг 31.4. Использование выражений XPath в расширении DOM

```
<?php
/* Создадим новый объект, опуская необязательные параметры,
   но будем помнить, что наш XML-документ содержит русские буквы.
*/
$doc = new DOMDocument;

/*
   Вот пример использования свойства preserveWhiteSpace объекта
   DOMDocument. По умолчанию значение этого свойства установлено
   в true, что предписывает парсеру не удалять избыточные пробелы.
*/
$doc->preserveWhiteSpace = false;
// Загружаем файл с документом.
$doc->Load('movie.xml');
// Создаем объект класса DOMXPath.
$xpath = new DOMXPath($doc);

// Создаем теперь путь для поиска элемента.
$query = '//name';
/*
   Вызываем метод query(), который выполняет поиск по заданному пути
   и возвращает список найденных узлов в виде списка NodeList.
*/
$entries = $xpath->query($query);
/*
```

```
   Вспомните теперь, что у объекта NodeList есть дочерние объекты Node.
   Возьмем объект за объектом из списка NodeList и распечатаем
   свойство nodeValue объекта Node.
*/
foreach ($entries as $entry) {
   echo iconv("UTF-8", "windows-1251", $entry->nodeValue)."<br />";
}
?>
```

Конечно, обработка XML-документов с помощью SimpleXML или DOM непроста, но задумайтесь над альтернативой! XML-документ — это текст, а текст всегда можно обработать с помощью регулярных выражений. Захотите ли вы пойти по этому пути?

Расширение XSL в PHP 5

Расширение XSL реализует стандарт XSL, выполняя XSLT-преобразования с использованием библиотеки libxslt. Оно является частью библиотеки DOM и поставляется в составе PHP 5, но требует подключения. Откроем файл php.ini и добавим в общий список расширений еще одну строку:

```
extension=php_xsl.dll
```

Проверьте, что эта библиотека действительно хранится в каталоге ext вместе с прочими библиотеками. Перезапустите Apache, и мы готовы к применению нового расширения.

Единственный класс, определенный в расширении XSL, — это `XSLTProcessor`. Его конструктор при вызове создает XSLT-парсер. В классе определены методы, позволяющие обработать XML-документ стилевыми таблицами и сохранить результат в виде HTML-, XML-документа или объекта класса `DOMDocument`.

Посмотрим пример использования методов расширения XSL (листинг 31.5).

Листинг 31.5. Преобразование XML-документа с помощью таблицы XSLT и расширения XSL, входящего в PHP 5

```
<?php
// Создание объекта класса DOMDocument
$xml = new DOMDocument;
// Загрузка XML-документа, описывающего фильмы (см. листинг 25.1)
$xml->load('movie.xml');
/*
   Создание объекта для размещения в нем стилевой таблицы
   (XSL-таблица ведь тоже создана на языке XML).
*/
```

```
$xsl = new DOMDocument;
// Загрузка стилевой таблицы (файл movie.xsl из гл. 25)
$xsl->load('movie.xsl');

// Создание парсера-преобразователя
$proc = new XSLTProcessor;
// Присоединение стилевой таблицы
$proc->importStyleSheet($xsl);
// Выполнение преобразования и сохранение результата в файле out.html
$proc->transformToURI($xml, 'out.html');
?>
```

Приложение

Описание компакт-диска

На компакт-диске, прилагаемом к книге, находятся примеры, разобранные в частях I—IV.

Номера каталогов с примерами — это номера частей, внутри подкаталога примеры пронумерованы так же, как в тексте книги.

Кроме того, на диске имеются дистрибутивы описываемого в книге программного обеспечения и документация по PHP 5 (табл. П1).

Таблица П1. *Содержание компакт-диска*

Каталоги	**Описание**
Part 1	Примеры к *части I*
Part 2	Примеры к *части II*
Part 3	Примеры к *части III*
Part 4	Примеры к *части IV*
Apache	Дистрибутив Web-сервера Apache 2.2
Editor	Редактор Notepad++
MySQL_5	Дистрибутив сервера MySQL 5.1
PHP_5	Дистрибутив PHP 5.3
PHP_Documentation	Документация по языку PHP
phpMyAdmin	Web-интерфейс для работы с MySQL

Предметный указатель

А

Б

В, Д

З, И

К