Самоучитель

Зольников Д. С.

PHP 5

Как самостоятельно создать сайт любой сложности

2-е издание, стереотипное

NT Press
Москва

УДК 004.438
ББК 32.973.26-018.1
З-79

Подписано в печать 07.02.2007. Формат 70x90 1/16. Гарнитура «Баскервиль».
Печать офсетная.Усл. печ. л. 19,14. Тираж 3000 экз. Заказ № 4375.

Зольников, Д.С.

З-79 PHP 5. Как самостоятельно создать сайт любой сложности / Д.С. Зольников. — 2-е изд., стер. — М.: НТ Пресс, 2007. — 272 с.: ил. — (Самоучитель).

ISBN 978-5-477-00377-8

В книге приведены основные сведения по языку Web-программирования PHP, который позволяет решать задачи любой сложности и формировать динамические разделы сайта: форумы, гостевые книги, каталоги продукции и многое другое. Помимо синтаксиса и возможностей языка рассматривается установка и настройка Web-сервера Apache, на котором, как правило, выполняются PHP-программы.

Издание отличается от традиционных учебников тем, что оно носит не справочный, а обучающий характер. Оно рассчитано на начинающего пользователя, имеющего базовые знания о программировании.

УДК 004.438
ББК 32.973.26-018.1

Содержание

Введение

Настоящее и будущее не представляется без информационных технологий. Каждый день миллионы людей покупают и продают товары через Internet-магазины, пользуются электронной почтой, ищут нужную информацию. Поэтому требования к Web-программистам постоянно растут. Время изготовления, качество и дешевизна продукции были и остаются главными критериями при выборе хорошего производителя. И именно технология серверного программирования позволит грамотно решить многие задачи в кратчайшие сроки.

Предметом этой книги является PHP (Personal Home Page) – универсальный язык скриптов (scripting language), который можно встраивать в HTML. Он используется сегодня более чем девятью миллионами Web-узлами во всем мире. Крупнейшие корпорации, в том числе CBS, Philips, Cisco, Japan Airlines, Air Canada, Lufthansa, GE Marketplace и Lycos/Maxinvest, выбрали PHP для своей работы. Именно простота, гибкость и скорость выполнения команд привели к такому широкому распространению этого языка.

Уровень материала этой книги изначально рассчитан на новичка. Но я уверен, что она будет полезна как начинающим Web-программистам, так и опытным пользователям. Если вы считаете себя специалистом в области информационных технологий, то эта книга послужит хорошим справочником и поможет разобраться в тонкостях серверного программирования. Начинающие пользователи получат все необходимые азы для дальнейшего развития. Самое главное – понять принципы программирования и усвоить возможности языка.

Книга имеет следующую структуру. В главе 1 рассматривается общая информация о PHP. Здесь рассказывается об истории создания и развития PHP, его возможностях, плюсах и минусах и т.п. Опираясь на содержание главы 2, вы сможете установить необходимое программное обеспечение для написания PHP-сценариев и запуска их на своем компьютере. Непосредственное изучение языка начинается с главы 3, в которой речь пойдет об основах синтаксиса PHP: вы узнаете не только о принципах написания PHP-программ, но и о правилах хорошего кода. В главе 4 объясняется, что такое переменная, константа и тип данных. Без этих понятий дальнейшее изучение материала будет очень затруднительным. В главах 5 и 6 говорится об операторах. Они очень часто используются на практике, поэтому уделите им по возможности больше внимания. В главе 7 речь пойдет о функциях: вы узнаете, какие они бывают, как использовать уже существующие и как создавать собственные. В главе 8 рассказывается о массивах. Инициализация, вывод, преобразование, сортировка массивов – все это и многое другое содержится в данной главе. Научиться работать со строками можно, прочитав главу 9.

Следующие главы помогут вам при работе с внешними структурами, такими как файлы, базы данных, cookies и др. В главе 10 речь пойдет о HTML-формах, в главе 11 – о файлах и каталогах, в главе 12 – о базах данных. Работа с графическими изображениями рассматривается в главе 13. Из 14 главы вы узнаете, как работать с датами и временем с помощью PHP. В последних двух главах этой книги мы рассмотрим соответственно регулярные выражения и Cookies.

Обратите внимание на основные определения и понятия. Именно на них опирается вся теория. Рассматривая практические примеры, постарайтесь вникать в суть каждой строчки. Пробуйте изменять параметры и смотрите, на результат. В конце каждой главы есть резюме, где подводятся итоги.

Главная цель этой книги – добиться того, чтобы вы могли создавать собственные приложения. Ставьте перед собой задачи и решайте их. Искренне надеюсь, что изложенный материал поможет вам в этом.

Принятые соглашения

В книге, которую вы держите в руках, содержатся специальные комментарии, выделенные из основного текста особым образом и предназначенные для удобства восприятия материала.

Названия клавиш выделены полужирным: **Insert** или **Shift+F4** (знак плюс в данном случае отражает факт нажатия клавиш «одновременно»: то есть здесь необходимо нажать клавишу **Shift**, и, удерживая ее, нажать на **F4**).

Кроме того, вы можете встретить слова, выделенные полужирным шрифтом – так обозначены надписи и сообщения, которые вы можете видеть непосредственно на экране компьютера, например **Сервис**, **Печать** и т.д.

Определения, новые понятия, встречающиеся в книге в первый раз, выделены *курсивом*.

Символ ➳ указывает на то, что пользователю необходимо произвести выбор указанной после стрелки позиции меню. Например, **Сервис ➳ Настройка параметров учета** означает следующее: в главном меню выберите **Сервис**, затем найдите и выберите пункт **Настройка параметров учета**. Самый последний заголовок в такой последовательности может означать не пункт меню, а имя вкладки диалогового окна, на которую вам следует перейти, или же пункт контекстного меню, который вам необходимо выбрать.


Таким образом в тексте помечены советы.


Так отмечены ключевые термины.

Глава 1 Приступая к работе

Прежде чем приступать к непосредственному изучению азов программирования на PHP, нужно понять, что собственно представляет собой этот язык. Итак, приступим.

PHP – это...

PHP – это язык программирования, который служит разработчикам Web-сайтов верой и правдой уже более девяти лет. Когда-то он мог решать лишь узкий круг задач, теперь функциональность языка настолько велика, что не хватит и тысячи страниц для описания его возможностей. И в тоже время язык PHP прост и доступен широкому кругу пользователей. Изучив его, вы сможете создавать как простые приложения, например счетчик посещений или гостевую книгу, так и большие Web-сайты, способные самостоятельно реагировать на действия пользователя. Сегодня PHP используют сотни тысяч разработчиков, миллионы Web-сайтов работают на этом простом и эффективном языке программирования.

История PHP

История PHP началась в далеком 1994 году, когда обычный программист Расмус Ледфорд решил написать небольшой набор скриптов на языке Perl для ведения статистики посещений домашней страницы, где было расположено его резюме. Набор скриптов он назвал Personal Home Page Tools (Инструменты для персональной домашней страницы). Вскоре Расмус реализовал на языке Си более функциональную версию в связке с базами данных. Она уже позволяла пользователям создавать несложные Web-приложения.

В 1997 году появляется PHP/FI 2.0 (Personal Home Page / Forms Interpreter – Персональная домашняя страница / Интерпретатор форм), способный работать с HTML-формами. В его создании принимали участие уже несколько человек. Новый язык заинтересовал многих Web-разработчиков: так, в 1997 году PHP/FI использовался примерно на 50 тыс. доменах в сети Internet, что составляло около 1% от их общего числа. В ноябре этого же года вышел официальный релиз PHP/FI 2.0, а за ним и совершенно новый PHP 3.0, очень напоминающий современный язык PHP-сценариев.

С самого начала PHP разрабатывался как продукт с открытым кодом. Это означает, что в его разработке могут принимать участие все, кто пожелают. Такими людьми стали Зив Сураски (Zeev Suraski) и Энди Гутманс (Andi Gutmans), которые переписали код PHP заново, так как PHP/FI 2.0 был мало пригоден для решения задач электронной коммерции. Расмус, Зив и Энди решили объединить свои усилия для совместной разработки PHP 3.0, объявив его преемником PHP/FI 2.0. Программисты добились того, что с помощью языка можно было работать с базами данных, различными протоколами и функциями API. Очень важным достижением стала реализация идеологии объектно-ориентированного программирования. После продолжительного тестирования в 1998 году был выпущен официальный релиз PHP 3.0.

Качественно новый язык получил новое название, сохранив старую аббревиатуру, теперь содержащую рекурсивный акроним PHP: Hypertext Preprocessor (PHP: препроцессор гипертекста). Разработчики

не остановились на достигнутом и решили увеличить производительность при работе со сложными сценариями. После продолжительной работы программисты успешно справились с поставленными задачами. Продуктом их деятельности явился движок Zend Engine (от Zeev и Andi), который стал основой для работы PHP. Помимо улучшения производительности, новый движок включал поддержку сессий, буферизацию вывода, более безопасные способы обработки поступающей от пользователя информации и многое другое. PHP 4.0 вышел почти через два года после своего предшественника PHP 3.0 – в мае 2000 года.

Сейчас можно говорить о том, что PHP не уступает другим языкам Web-программирования ни в функциональности, ни в скорости. Выход в свет PHP 5.0, который основан на новом движке Zend Engine 2, значительно переработанном и улучшенном создателями, подтверждает это. Подробнее о Zend Engine читайте на официальном сайте www.zend.com. В PHP 5 более глубоко реализован объектно-ориентированный подход. Это обстоятельство наверняка привлечет многих передовых программистов мира, что позволит языку стать еще более мощным.

Почему именно PHP

Зададимся вопросом, чем привлекает PHP Web-программистов, и какие он имеет преимущества перед другими конкурирующими языками, такими как Perl или ASP. Для этого нужно разобраться с его основными характеристиками.

Открытый код

Мы уже говорили о том, что язык PHP изначально создавался как продукт с открытым кодом (open source). Это означает, что любой человек может получить его исходный текст, причем совершенно бесплатно. Это не могло не отразиться на массовости применения языка. Преимущество открытого кода заключается еще в том, что выявление ошибок может производиться не только разработчиками, но и другими программистами.

Производительность

Другим важным критерием оценки применимости языка является его производительность. До некоторого времени, а именно до версии 3.0, PHP работал значительно медленнее со сложными сценариями. Но благодаря постоянной работе программистов в этом направлении, версии 4.0 и 5.0 не уступают в скорости сценариям на Perl и ASP. К тому же, разработчику предоставляется множество стандартных функций, написанных на Си, что позволяет еще более увеличить производительность и расширить возможности языка.

Переносимость

Переносимость языка означает, что программный код может использоваться на разных системах. Таким свойством обладает PHP. Однажды написанная на нем программа может работать на многих операционных системах (Windows, Linux, Unix и другие) и почти на всех Web-серверах (это понятие будет разобрано позже). В этом плане ASP значительно уступает PHP, так как приложения на нем в основном предназначены для работы на сервере IIS (Internet Information Server) от компании Microsoft.

Среда разработки

Язык PHP не требует специализированной среды разработки, так как программа на нем является простым текстом. Для пользователей операционной системы Windows это может быть Блокнот, а для системы Unix – Emacs.

Будьте внимательны при использовании таких программных продуктов, как Microsoft Office Word. Убедитесь в том, что сохраняемый вами файл является обычным текстом, иначе ваша программа не сможет корректно выполняться.

Простота синтаксиса

В плане синтаксиса PHP выглядит намного предпочтительнее языка Perl. Синтаксис PHP прост и естественен. Читатели, знакомые с языком

Си, найдут много общего с PHP. В свою очередь Perl является очень надежным и мощным языком программирования, который может решать многие задачи, встречающиеся в сети Internet. Но сложность структуры синтаксиса и большое количество затрачиваемых ресурсов делают применение этого языка менее эффективным по сравнению с PHP.

Благодаря всем этим и другим свойствам, PHP стал очень популярен среди Web-программистов. На сегодняшний день PHP используется на более чем 15 миллионах доменах в сети Internet. Это обстоятельство говорит об эффективности и в тоже время простоте рассматриваемого языка.

Как все это работает

Обычно когда мы отправляем письма через электронную почту, пользуемся услугами поисковиков, общаемся в чате и т.п., то не задумываемся о том, как все это работает. Это естественно, так как в данном случае мы являемся пользователями, а не разработчиками. Цель программиста – скрыть реальный механизм работы приложения, предоставляя пользователю простой и удобный интерфейс. Наша задача – научиться программировать на PHP, поэтому необходимо знать принципы его работы.

Итак, первое, что надо уяснить, – это различие между серверным языком программирования и языком, исполняемым на стороне клиента. Название «серверный язык» PHP получил оттого, что выполнение программы на нем целиком и полностью происходит на *сервере*. Существует множество определений этого понятия. На данном этапе для лучшего визуального представления работы PHP будем ассоциировать его с удаленным компьютером.

Сервер – это компьютер, предоставляющий свои ресурсы другим компьютерам.

Сервер работает в связке с *клиентом*. Он может представлять собой любой компьютер пользователя с браузером.

Клиент – компьютер, потребляющий ресурсы сервера.

Обращение клиента к серверу происходит посредством запросов, которые обрабатываются специальной программой, называемой *Web-сервер.*

Web-сервер – программа, обрабатывающая запросы от пользователей World Wide Web.

Рассмотрим схему взаимодействия клиента и сервера при вызове обычной HTML-страницы (см. рис. 1.1). Например, пользователь набрал в адресной строке браузера http://www.htmlsite.ru/index.html. При этом выполняются следующие действия:

- браузер клиента формирует соответствующий запрос, который затем посылается на сервер;
- Web-сервер читает его, находит соответствующую страницу (index.html) на сервере и отправляет страницу обратно клиенту;
- браузер клиента обрабатывает ее и выводит на экран.

Обратите внимание, что пересылаемая страница может содержать не только чистый HTML-код, но и java-скрипты, каскадные таблицы стилей (CSS) и другие технологии, которые работают на стороне клиента. Такие страницы обычно называют статическими, потому что они не могут изменяться в пределах сервера под влиянием пользователя.

Теперь рассмотрим случай вызова пользователем PHP-страницы (см. рис. 1.2). Например, http://www.phpsite.ru/index.php:

- браузер клиента формирует соответствующий запрос, который затем посылается на сервер;
- Web-сервер читает его и находит соответствующую страницу (in dex.php) на сервере;
- выполняется PHP-код;
- Web-сервер отправляет результирующую страницу обратно клиенту;
- браузер клиента обрабатывает ее и выводит на экран.

Итак, перед вами два примера. В первом случае какая страница хранится на сервере, такая и отправляется пользователю. Во втором

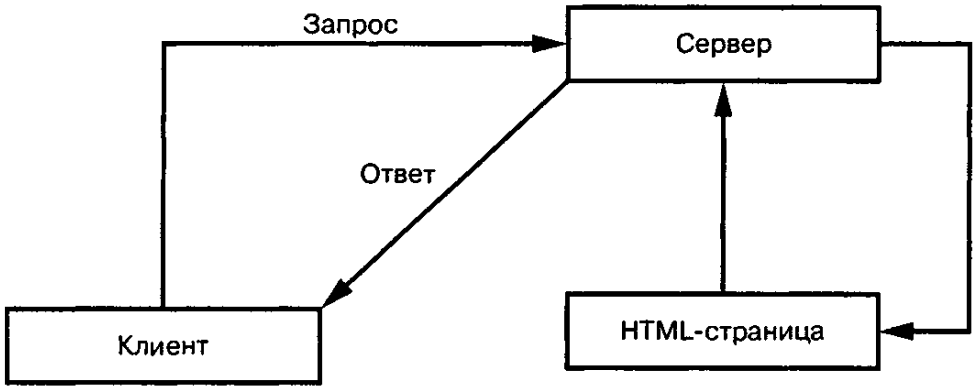

Рис. 1.1 ▾ Вызов HTML-страницы

случае страница формируется в результате выполнения PHP-кода, который находится в файле index.php. На самом деле они могут иметь расширения .php, .php4, .phtml и другие в зависимости от настроек Web-сервера (более подробно об этом читайте в следующей главе). Именно тот факт, что страница генерируется, а не просто пересылается пользователю, позволяет добиться большей гибкости и функциональности Web-сайта. До появления серверных языков программирования большие Web-сайты состояли из сотен статических страниц, которые имели ссылки друг на друга. Такое положение вещей приводило к тому, что изменение какой-либо информации могло занимать у программиста продолжительное время. На смену статическим Web-сайтам пришли

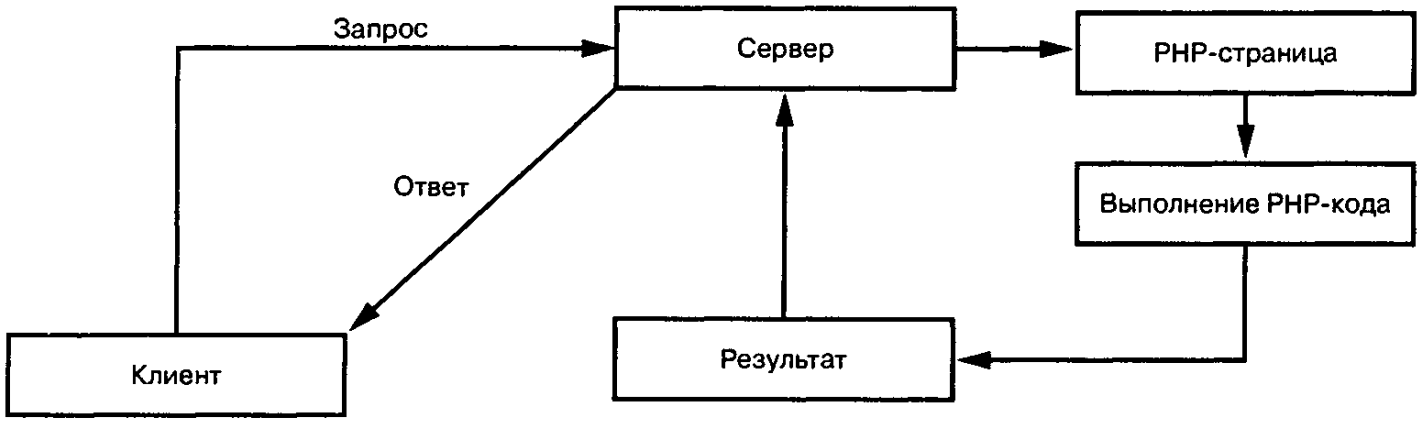

Рис. 1.2 ▾ Вызов PHP-страницы

динамические, которые позволяли резко сократить количество страниц на сервере и повысить производительность. В настоящее время применение серверных технологий, в частности PHP, стало неотъемлемым, так как сложность задач, которые ставятся перед Web-сайтом, постоянно растет. А использование статических страниц подобно подравниванию газона вручную, когда рядом стоит газонокосилка и при том совершенно бесплатная.

От интерпретатора к компилятору

Мы уже говорили, что версия PHP 3.0 значительно уступала другим языкам в скорости работы программы. Особенно это было ощутимо в больших сценариях с использованием циклов, когда одни и те же действия повторялись многократно. Так что же изменилось с появлением PHP 4.0, который уже был наравне по производительности с Perl и ASP? Чтобы разобраться в этом вопросе, подробно рассмотрим обработку PHP-кода.

До третьей версии включительно PHP был *интерпретатором*. Это сложно произносимое слово применимо к тем обработчикам кода, которые последовательно выполняют конструкции языка шаг за шагом.

Интерпретатор – это утилита, которая получает исходный код файла и последовательно выполняет все указанные в нем конструкции языка.

На первых порах интерпретатора было вполне достаточно, так как программы были простые. Но со временем сценарии становились все сложнее и сложнее, а когда встречались циклы с более чем миллионом итераций, программа могла выполняться в течение нескольких минут, что непозволительно при работе в Internet. Основная задержка происходила из-за того, что перед каждой итерацией интерпретатор производил синтаксическую проверку выполняемой конструкции.

Такое положение вещей не устраивало ни пользователей, ни разработчиков. Для повышения производительности была добавлена фаза *трансляции*, в результате которой исходный код программы преобразуется в более удобный для обработки интерпретатором код.

Трансляция – это процесс преобразования кода из одного языка программирования в другой, более удобный для дальнейшего выполнения.

За счет введения фазы трансляции производительность PHP повысилась в несколько раз, так как при этом синтаксическая проверка выполняется только в самом начале. Такая структура обработки кода программы схожа с работой компилятора. Отличие состоит в том, что последний работает с машинным кодом, который выполняется значительно быстрее. К счастью, решение Internet-задач не требует такой производительности, которую имеет компилятор (например, языка Си), поэтому сегодня дальнейшее развитие PHP в этом плане нецелесообразно.

Более подробно рассматривать обработку PHP-кода мы не будем, так как это не является основной темой книги. Далее мы познакомимся с возможностями языка.

Возможности PHP

В основном PHP используется в связке с базами данных. На данный момент они являются самыми удобными хранилищами информации. Сейчас даже самые обычные гостевые книги делают с применением баз данных, а большие информационные порталы вообще сложно представить без них. Язык PHP позволяет работать с различными типами баз данных, такими как MySQL, ODBC, Oracle (OCI7 и OCI8), InterBase и др. Интерфейс доступа к ним очень простой и доступный даже для начинающих.

Также широко используется возможность работы с электронной почтой. Поддерживаются такие протоколы, как IMAP, POP3, SMTP и другие для отправки и приема сообщений. Поэтому очень просто создать PHP-приложение для рассылки писем, например открыток своим друзьям.

Ранее говорилось о том, что PHP генерирует страницы, но его возможности на этом не исчерпываются. С помощью PHP можно формировать изображения, файлы типа PDF, а также Flash-анимацию.

Работа с файлами тоже является частью рассматриваемого языка. PHP представляет полный спектр функций для этого: вы сможете удалять,

копировать, создавать файлы, записывать и изымать из них нужную информацию.

PHP имеет возможность работать с технологиями в области электронной коммерции, такими как CCVS, Cybercash, VeriSign Payflow Pro и др.

Конечно, нельзя обойти стороной введение в PHP объектно-ориентированного подхода. Это позволило языку подняться в своем развитии на планку выше и привлечь к себе внимание ведущих программистов планеты.

Это всего лишь малая часть того, что может PHP. Его функциональность может поразить даже самых требовательных программистов. Вам же остается только научиться пользоваться предоставляемыми ресурсами, и тогда вы сможете самостоятельно создавать Web-приложения высокого уровня.

Что потребуется для работы

Рассказ о работе и возможностях языка PHP мог бы занять десятки страниц. Но думаю, что стоит перейти к знакомству с программным обеспечением, которое позволит вам работать с ним на своем компьютере. Для этого вам понадобится Web-сервер и модули PHP.

До этого момента мы говорили о работе с удаленным компьютером. Но при таких условиях создавать и тестировать PHP-приложения неудобно, так как для этого надо постоянно поддерживать с ним связь. При отсутствии соединения работа вообще не представляется возможной. Поэтому Web-сервер и модули PHP лучше установить на свой компьютер. При этом он будет являться одновременно клиентом и сервером.

Apache

Многие ведущие программисты используют один из лучших Web-серверов – Apache, разработчики которого предлагают как версии под

Windows, так и для UNIX подобных систем. Мы не будем исключением. Apache завоевал огромную популярность в сети Internet: сейчас под ним работает до 67% Web-сайтов, в то время как IIS применяется лишь на 21% серверов. Скачать версию Apache для операционной системы Windows можно с официального сайта www.apache.org.

PHP

Что касается PHP, то создатели языка предлагают нам как последнюю стабильную версию PHP 4.3.7, так и релиз PHP 5.0. Принципиальной разницы в выборе версии PHP нет: вы можете изучить основы языка на PHP 4, а затем установить одну из последних версий и оценить новые возможности движка и стандартных функций. В этой книге описана работа с PHP и Apache на платформе Windows, но благодаря переносимости языка, вы сможете запускать созданные вами PHP-программы и в среде UNIX. Скачать дистрибутив PHP можно с сайта www.php.net.

MySQL

Мы уже говорили, что язык PHP позволяет программисту работать со множеством баз данных. Одной из наиболее популярных СУБД (Система управления базами данных) является MySQL. В этой книге читатель познакомится с основами языка SQL (Structed Query Language – Структурированный язык запросов) и научится работать с базами данных посредством языка PHP. Существует бесплатная версия MySQL для Windows, которую всегда можно скачать с официального сайта www.mysql.com.

В следующей главе описано, как установить и настроить каждый из необходимых компонентов. Комментарии и пояснения, приведенные там, помогут связать их воедино. Умение устанавливать и настраивать Apache, PHP и MySQL является неотъемлемой частью работы программиста, создающего приложения для Web.

В поисках ответа

На сегодняшний день издано множество книг, справочников и пособий по PHP. К сожалению, эта информация объективно не может охватить все проблемы, возникающие при написании PHP-программ. Поэтому для поиска ответов последние 10 лет все чаще и чаще используется Internet.

Полезные сайты

На официальных сайтах разработчиков вы всегда сможете найти информацию о новых стандартных функциях языка, интересные статьи, найденные ошибки, описание алгоритмов программирования на PHP. Исчерпывающую документацию по Apache или MySQL приведена на сайтах:

- официальный сайт разработчиков PHP: http://www.php.net;
- сайт разработчиков движка Zend Engine: http://www.zend.com;
- клуб разработчиков PHP: http://www.phpclub.net;
- документация по MySQL на русском языке: http://dev.mysql.com/doc/;
- документация по Web-серверу Apache: http://httpd.apache.org/docs/.

Как искать

Итак, вы столкнулись с неразрешимой для вас проблемой. Это может быть возникающая ошибка в процессе выполнения программы, незнание способа реализации и т.п. Сразу надо сказать, что наверняка в подобную ситуацию попадал кто-нибудь еще, и выход из нее подробным образом разобран в таких разделах, как FAQ (Frequently Asked Questions / Перечень типичных вопросов), или подобных им.

Ваша проблема еще не решена? Тогда можно задать вопрос на форумах, где профессиональные программисты наверняка разберутся, что к чему.

И помните, что последней инстанцией являются непосредственно разработчики программного продукта. Так как в большинстве своем они заинтересованы в распространении языка PHP, то всегда будут рады прийти вам на помощь.

Заключение к главе

Итак, в этой главе вы познакомились с историей PHP, его возможностями, отличительными особенностями и принципом работы. Мы разобрали, что такое Web-сервер, интерпретатор, транслятор и другие понятия. И, наконец, теперь вы знаете, где, что и как искать.

Глава 2 Установка и конфигурирование

В предыдущей главе мы познакомились с программным обеспечением, с помощью которого можно создавать и тестировать приложения на PHP. В этой главе мы разберем один из способов установки и конфигурирования этих программ. В качестве операционной системы будем использовать Microsoft Windows 2000 Service Pack 4. Для других версий Windows порядок установки и конфигурирования программного обеспечения почти такой же. Исключение составляет операционная система Windows 95, так как PHP 5 не работает с ней.

Установка программ

Прежде чем приступать к установке программного обеспечения нужно скачать его дистрибутив. В нашем случае мы использовали ресурсы официальных сайтов производителей. Итак, начнем.

Apache

Программный продукт: Apache v2.0.50
Официальный сайт: http://httpd.apache.org/

Где скачать: http://httpd.apache.org/download.cgi
Файл: apache_2.0.50-win32-x86-no_ssl.msi

1. Запустите файл (apache_2.0.50-win32-x86-no_ssl.msi). На экране появится окно установки (рис. 2.1). Нажмите кнопку **Next**.

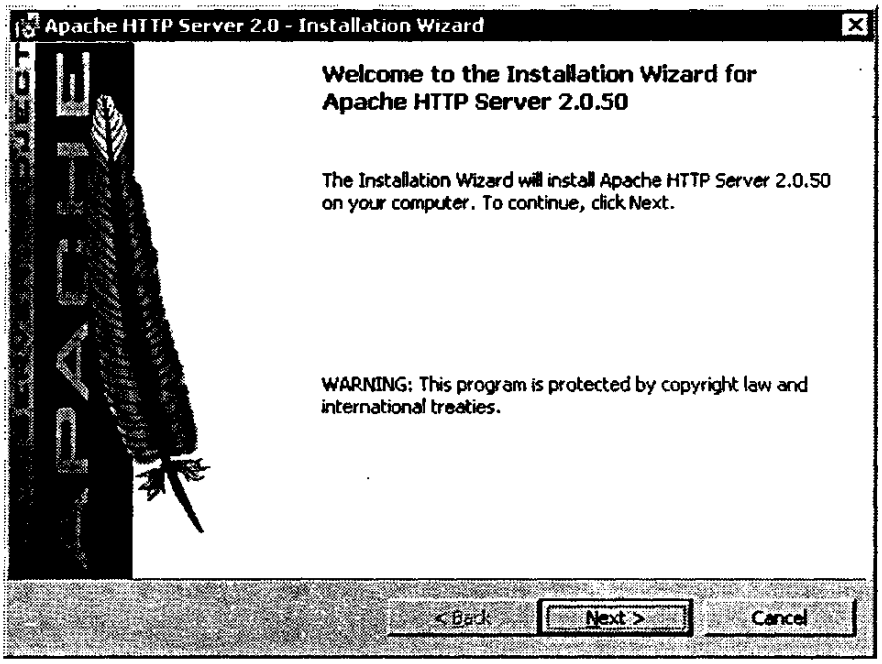

Рис. 2.1▾ Установка Apache

2. В следующем окне поставьте флажок, который означает принятие лицензионного соглашения (I accept the terms in the license agreement) и нажмите кнопку **Next**.
3. Далее вам предлагается ознакомиться с информацией о Web-сервере Apache. После прочтения нажмите кнопку **Next**.
4. Затем от вас требуется ввести домен сети (Network Domain), название сервера (Server Name) и адрес электронной почты администратора (Administrator's Email Address), а также выбрать порт для работы Apache. Заполните эти данные по вашему усмотрению (например, как на рис. 2.2). Не бойтесь ошибиться, так как позже у вас будет возможность редактировать эти данные. После заполнения нажмите кнопку **Next**.

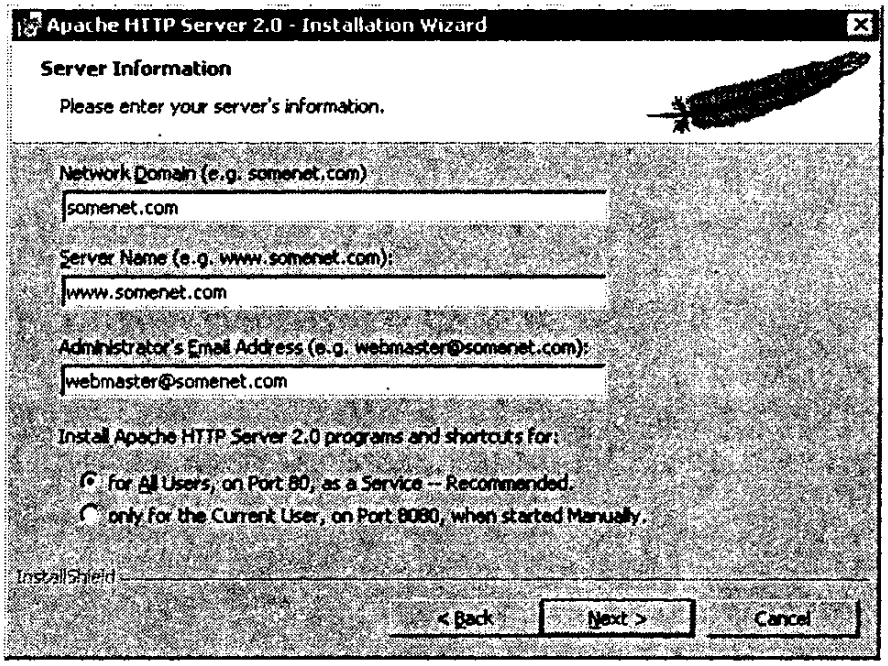

Рис. 2.2 ▼ Параметры сервера

5. Далее нужно выбрать тип установки: **Typical** (Обычная) или **Custom** (Выборочная). Поставьте флажок **Typical** и нажмите кнопку **Next**.

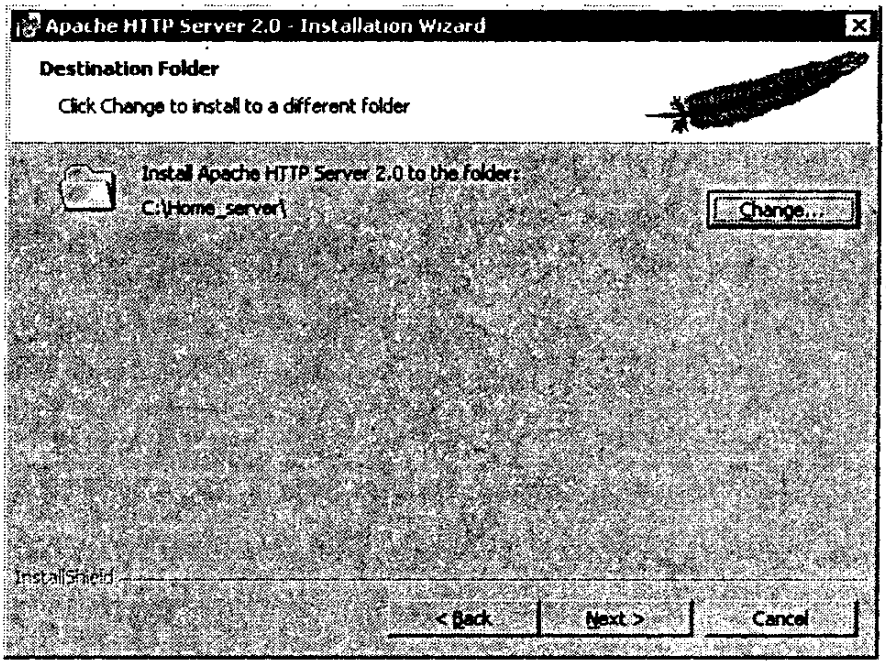

Рис. 2.3 ▼ Окно выбора каталога установки

6. В следующем окне вас ожидает выбор директории, куда будет устанавливаться Apache. По умолчанию программа установки предлагает путь C:\Program Files\Apache Group\. На самом деле принципиальной разницы в выборе места установки нет. В нашем случае мы выделим специальную папку для всех программных продуктов (рис. 2.3).

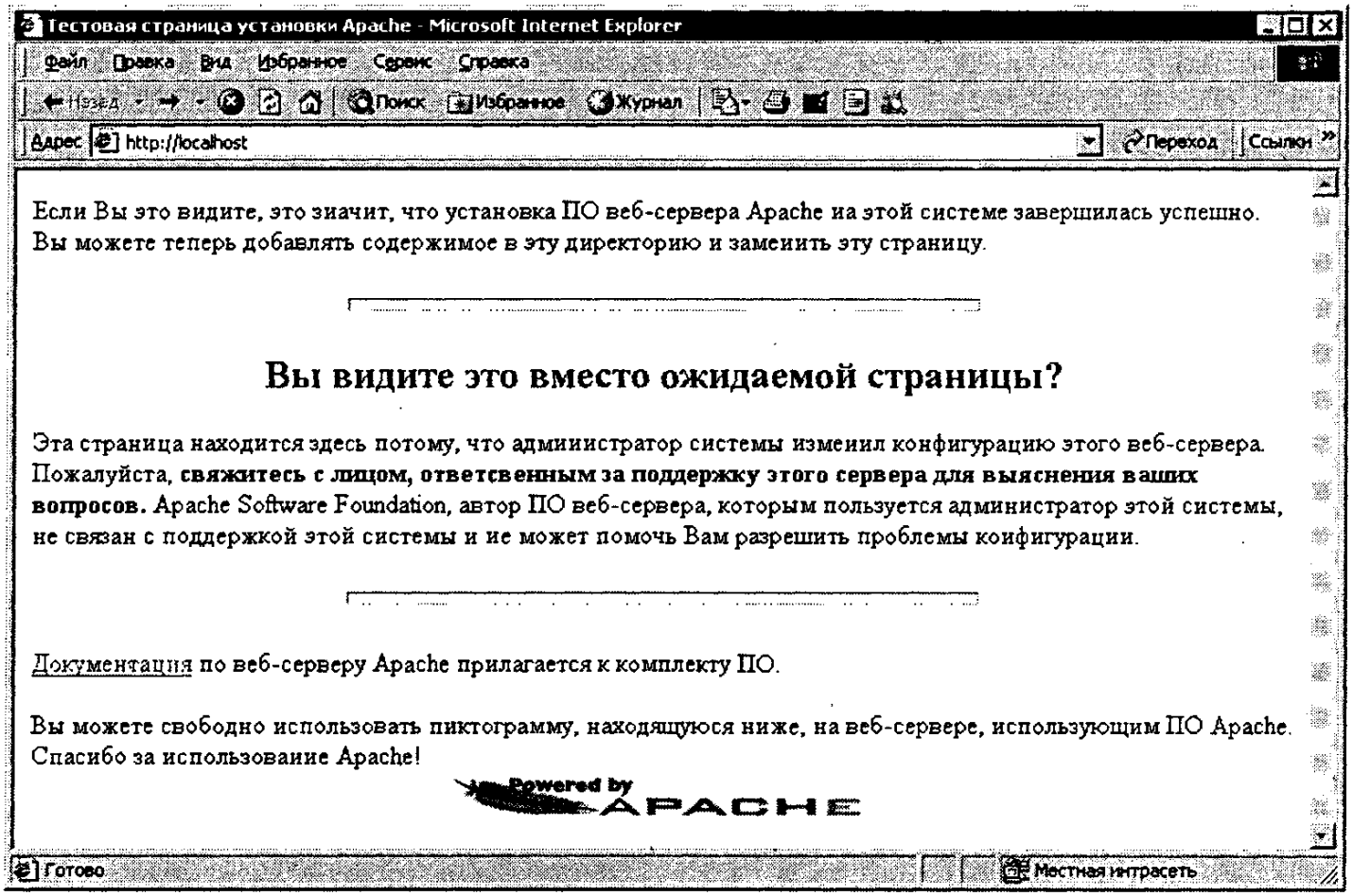

Рис. 2.4 ▾ Успешная установка Apache

7. Нажмите кнопку **Install**, после чего начнется установка Apache.
8. Для завершения нажмите кнопку **Finish**.

После того как завершится процесс установки Apache, произойдет автоматический запуск программы Apache Service Monitor и самого Web-сервера. Наберите в адресной строке браузера http://localhost/ и нажмите **Enter**. При этом должна появиться информация об успешной установке Web-сервера Apache (см. рис. 2.4).

Apache Service Monitor – это утилита, которая предназначена для запуска или прекращения работы Web-сервера. Ее иконка должна появиться в правом нижнем углу. По ней можно судить о состоянии Web-сервера на данный момент. Например, когда на иконке расположен зеленый треугольник, то Apache запущен.

PHP

Программный продукт: PHP v5.0.0
Официальный сайт: http://www.php.net/
Где скачать: http://www.php.net/downloads.php
Файл: php-5.0.0RC3-Win32.zip

1. Распакуйте файл (php-5.0.0RC3-Win32.zip) в папку C:\ Home_server\PHP5\.
2. Найдите в папке C:\Home_server\PHP5 файл с названием php.ini-dist. Переименуйте его в php.ini и скопируйте в папку установки Windows (в нашем случае C:\WINNT\).
3. Найдите файл libmysql.dll и скопируйте его в папку C:\WINNT\system32.

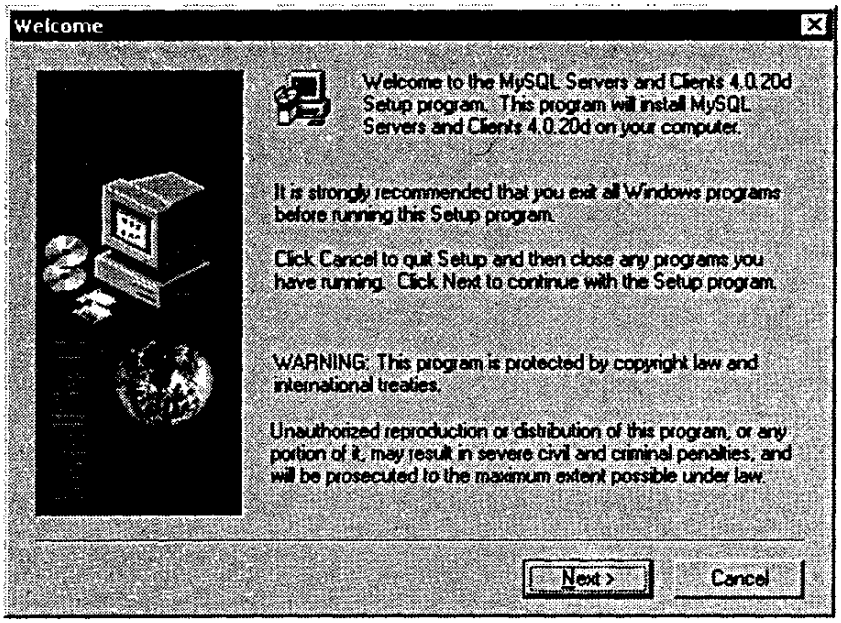

Рис. 2.5 ▼ Установка MySQL

MySQL

Программный продукт: MySQL v4.0.20d
Официальный сайт: http://www.mysql.com/
Где скачать: http://www.mysql.com/downloads/index.html
Файл: mysql-4.0.20d-win.zip

1. Распакуйте файл (mysql-4.0.20d-win.zip).
2. Запустите файл setup.exe. На экране появится окно установки (рис. 2.5). Нажмите кнопку **Next**.
3. Следующее окно имеет информационный характер. Ознакомьтесь с ним и нажмите кнопку **Next**.

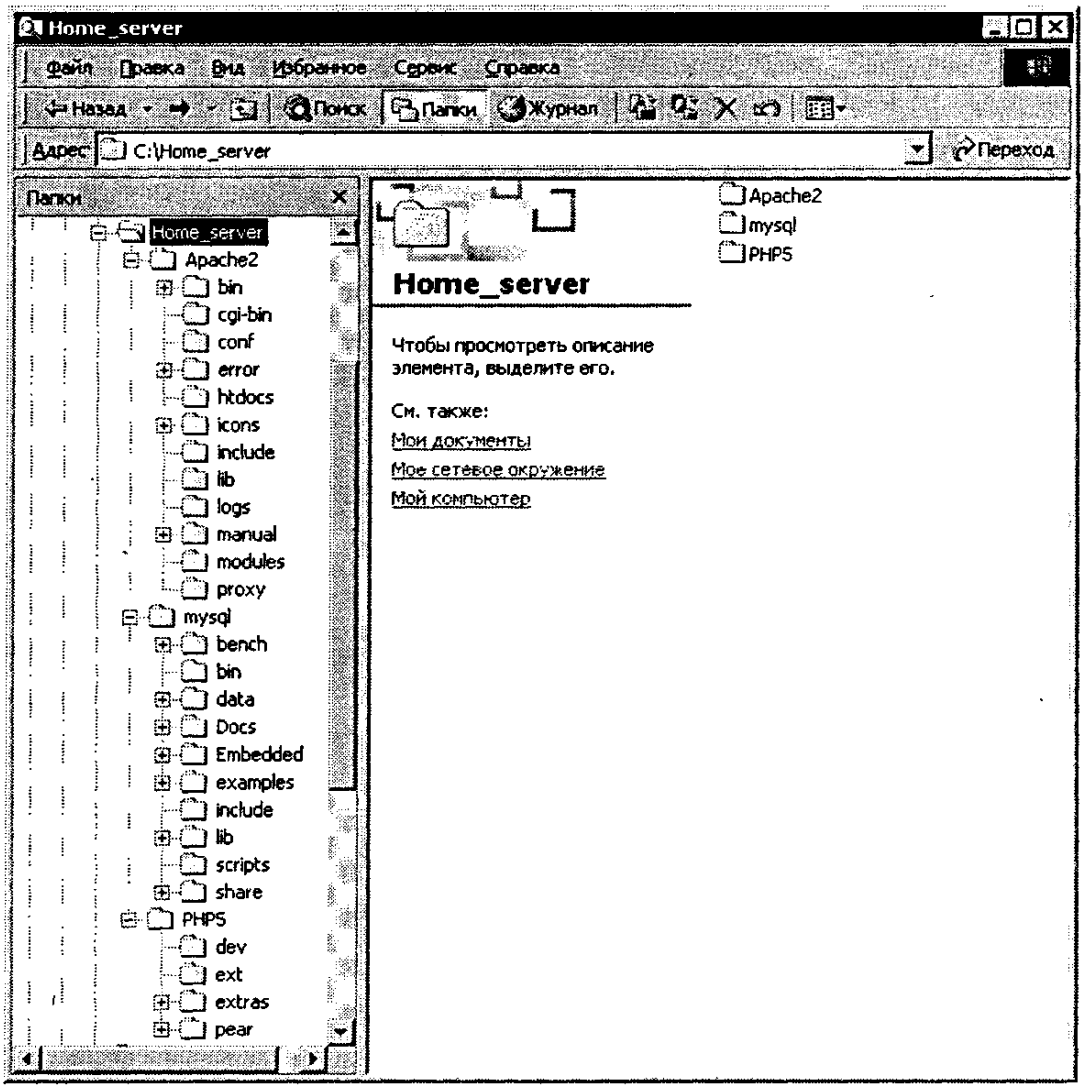

Рис. 2.6 ▾ Структура каталогов домашнего сервера

4. Далее нужно выбрать директорию, куда будет устанавливаться My SQL. По умолчанию программа установки предлагает путь C:\mysql. Поменяйте его на c:\home_server\mysql, так как все составляющие лучше располагать в одной папке, и нажмите кнопку **Next**.
5. Затем нужно выбрать тип установки. Поставьте флажок **Typical** и нажмите кнопку **Next**.
6. После установки MySQL нажмите кнопку **Finish**.

На этом установка программного обеспечения закончена. Структура каталогов домашнего сервера изображена на рис. 2.6.

Конфигурирование программ

После того как все необходимые программы установлены, нужно связать их воедино. Именно в этом и состоит назначение этапа конфигурирования. Напоминаем, что в этой книге приводится только один из вариантов установки и конфигурирования программного обеспечения. На самом деле этим делом занимаются специалисты, которые настраивают сервер в зависимости от решаемых задач. В нашем случае эти задачи являются стандартными, поэтому вариант настройки приводится самый элементарный. Итак, делаем очередной глоток кофе и продолжаем.

Apache

Для настройки конфигурации Apache нам потребуется только один файл – httpd.conf. Этот файл содержит в себе основные параметры сервера, с помощью которых определяется характер его работы. Найти его можно в папке C:\Home_server\Apache2\conf\ или следующем образом **Пуск ➤ Программы ➤ Apache HTTP Server 2.0.50 ➤ Configure Apache Server ➤ Edit the Apache httpd.conf Configuration File**.

1. Откройте файл конфигурации httpd.conf с помощью Блокнота Windows.
2. Найдите параметр `DocumentRoot`, который указывает на расположение HTML-файлов. Убедитесь в том, что он имеет значение

"C:/Home_server/Apache2/htdocs". Обратите внимание на то, что путь записывается с помощью знака /, а не \.

3. Далее нужно найти строчки, содержащие слово LoadModule. Поместите последней строкой следующее:

```
LoadModule php5_module "c:/Home_server/php5/php5apache2.dll"
```

С помощью этой команды загружаются модули для совместной работы Apache и PHP.

Обратите внимание на символ #, который часто встречается в рассматриваемом файле и служит для написания пояснений. Все, что следует за этим символом до конца строки, Apache игнорирует, поэтому внимательно следите за тем, чтобы к рассматриваемым настройкам не прилагались комментарии.

4. Последнее, что нужно сделать, – это найти строчки, содержащие AddType и добавить еще одну – AddType application/x-httpd-php .php. Тем самым мы указываем серверу расширение файлов, в которых находится PHP-код.
5. Сохраните изменения и закройте файл.

PHP

Для настройки PHP потребуется файл php.ini, который находится, если вы забыли, в папке C:\WINNT\. В нем содержатся основные параметры работы PHP.

1. Откройте файл php.ini с помощью Блокнота.
2. Найдите в нем параметр doc_root и поставьте ему значение "C:\Home_server\Apache2\htdocs". Обратите внимание, что в отличие от параметров Apache их значение записывается не через пробел, а с помощью знака равенства.
3. Теперь найдите параметр extension_dir и измените его на "C:\Home_server\PHP5\ext".
4. В завершении найдите строчку extension=php_mysql.dll и уберите символ точки с запятой (;), который служит аналогом # для Apache.
5. Сохраните изменения и закройте файл.

MySQL

На самом деле MySQL не требует особой настройки. Здесь мы просто расскажем о том, как запускать и завершать работу MySQL подобно Apache. Для этого создайте файл с расширением .bat (например, start_mysql. bat). Откройте содержимое этого файла с помощью Блокнота и вставьте следующие строки:

```
start c:\Home_server\mysql\bin\mysqld-nt.exe -u root
-standalone
exit
```

Сохраните изменения и закройте файл. Теперь, когда нужно будет запустить MySQL, просто дважды щелкните по созданному файлу.

Для завершения работы MySQL потребуется еще один bat-файл, содержание которого будет следующим:

```
start c:\Home_server\mysql\bin\mysqladmin -u root shutdown
exit
```

На этом заканчивается этап конфигурирования. Остается только протестировать установленные программы. Если вы четко следовали всем указаниям, то этот этап пройдет успешно.

Тестирование программ

Apache

1. Создайте файл с расширением .html (например, test_apache.html), который содержит всего одну строку: Hello, World!.
2. Скопируйте этот файл в папку C:\Home_server\Apa che2\htdocs\.
3. В адресной строке браузера наберите http://localhost/test_apache. html и нажмите **Enter**.
4. Результат смотрите на рис. 2.7.

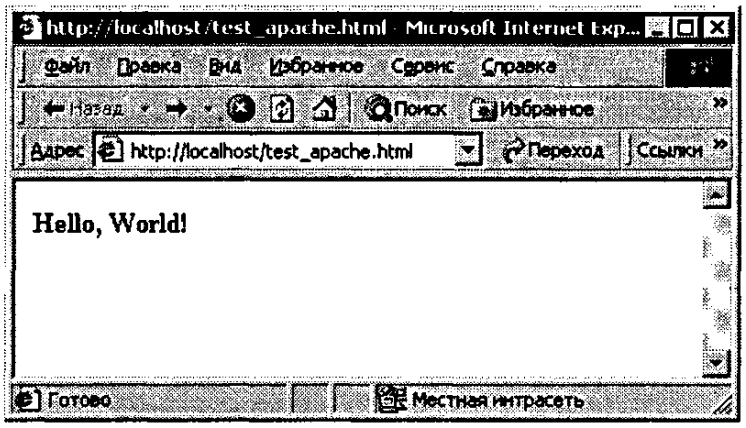

Рис. 2.7 ▾ Тестирование Apache

PHP

1. Создайте файл с расширением .php (например, test_php.php), который содержит следующий текст: `<?php echo "<b>Hello, PHP!</b>" ?>`.
2. Скопируйте этот файл в папку C:\Home_server\Apache2\htdocs\.

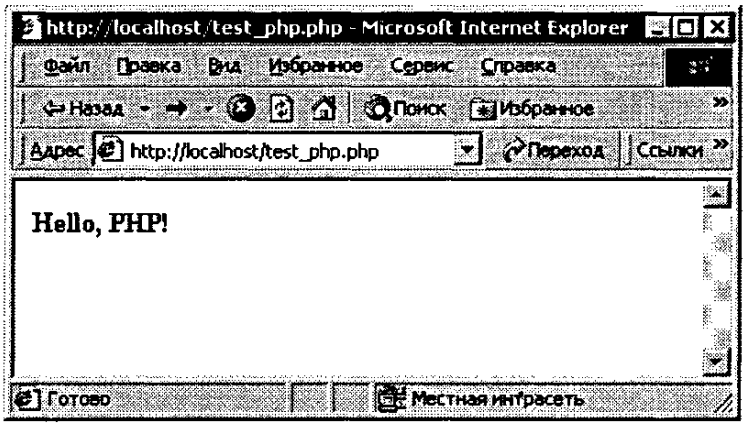

Рис. 2.8 ▾ Тестирование PHP

3. В адресной строке браузера наберите http://localhost/test_php.php и нажмите **Enter**.
4. Результат смотрите на рис. 2.8.

MySQL

1. Создайте файл с расширением .php (например, test_mysql.php), который содержит следующий текст:

```
<?php
if ($rc = mysql_connect("localhost","root","")) {
 echo "Hello, MySQL!";
}
else {
 echo "Error!";
}
?>
```

2. Скопируйте этот файл в папку C:\Home_server\Apache2\htdocs\.
3. В адресной строке браузера наберите http://localhost/test_mysql.php и нажмите **Enter**.
4. Результат смотрите на рис. 2.9.

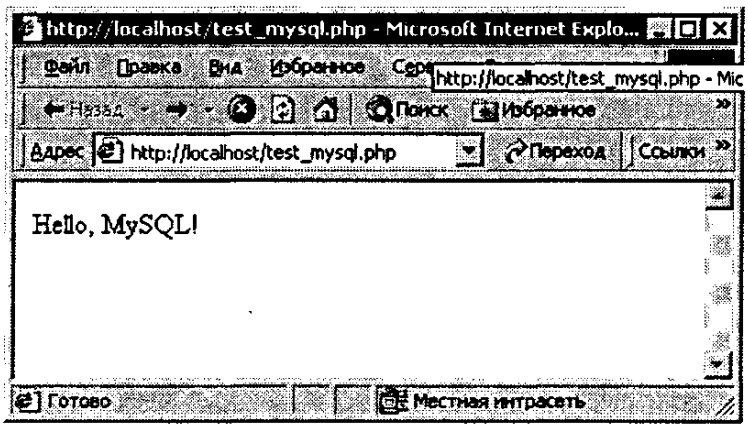

Рис. 2.9 ▼ Тестирование MySQL

Заключение к главе

Итак, в этой главе мы подробно разобрали этапы установки и конфигурирования программного обеспечения, а также протестировали его. Если в процессе тестирования у вас возникали ошибки, то внимательно проверьте правильность ваших предыдущих действий. Все примеры в книге выполнялись на сервере именно с теми настройками, которые приведены в этой главе.

Глава 3 Основы синтаксиса PHP

В предыдущей главе мы разобрали установку программного обеспечения для выполнения программ, написанных на PHP. Дело остается за малым – научиться создавать эти программы. Прежде всего нужно познакомиться с синтаксисом языка. Действительно, не зная слов и знаков препинания, человеку будет сложно составлять предложения, а тем более крупный тест. Итак, приступим.

Программа на PHP – это...

Как мы уже говорили в главе 1, программа на PHP (скрипт) – это просто текст. Поэтому для ее создания можно использовать обычный текстовый редактор (например, блокнот Windows).

Не будем откладывать дело в долгий ящик и приступим к непосредственному написанию первой программы. По сложившейся доброй традиции, результатом ее работы будет вывод сообщения «Hello, World!» в окне браузера.

Откройте текстовый редактор, введите туда строки из листинга 3.1 и сохраните как файл с расширением .php (например, hello.php). Поместите файл в корневой каталог вашего Web-сервера (в нашем случае это

C:\Home_server\Apache2\htdocs\) и наберите в адресной строке браузера http://localhost/hello.php. Перед тем как нажать **Enter**, убедитесь, что Apache запущен, о чем свидетельствует зеленый треугольник на иконке утилиты Apache Server Monitor в правом нижнем углу.

Листинг 3.1 ▾ Программа вывода сообщения

```
<?php
echo "Hello, World!";
?>
```

Результат выполнения программы смотрите на рис. 3.1.

Рис. 3.1 ▾ Вывод сообщения

Перейдем к подробному рассмотрению представленной программы. Обратите внимание на конструкцию <?php ... ?>, которая очень похожа на HTML-тег. С ее помощью выделяется PHP-код. Слово echo используется для вывода строки, которая находится в кавычках сразу после него (более подробно с этой командой мы познакомимся позже). Символ точка с запятой (;) выполняет такие же функции, как точка в конце предложения.

Выберите в меню браузера (в нашем случае Internet Explorer) **Вид** пункт **В виде HTML**. При этом откроется содержимое страницы в текстовом редакторе. Заметьте, что Web-сервер передал браузеру только строку «Hello, World!». Именно в этом состоит главная особенность серверного программирования, так как при вызове HTML-файлов они

пересылаются *без изменения*. В нашем случае сначала выполняется PHP-код, а затем результат отправляется браузеру.

Однако стоит помнить, что если код не заключить в специальные теги <?php...?>, то он будет передаваться без обработки PHP (см. листинг 3.2).

Листинг 3.2 ▼ Программа с текстом вне тегов PHP

```
<html>
<head>
   <title>Текст вне тегов PHP</title>
</head>
<body>
echo "Это не PHP-код";
<br>
<?php
echo "Hello, World!";
?>
<br>
echo "Это не PHP-код";
</body>
</html>
```

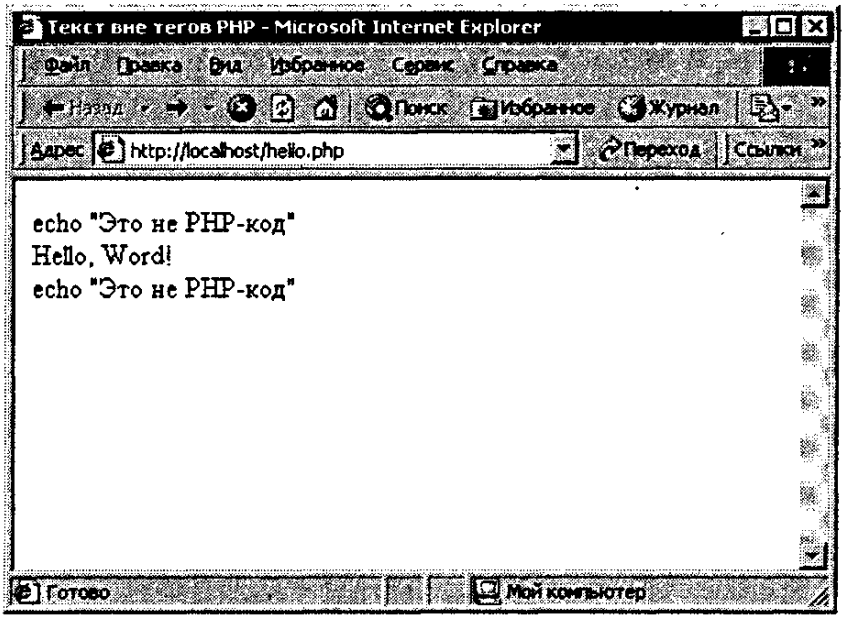

Рис. 3.2 ▼ Текст вне тегов PHP

Результат выполнения этой программы смотрите на рис. 3.2.

Обратите внимание, что в данном случае команда echo не выполняется в первой и последней строчке, а выводится как обычный текст. Это обстоятельство часто используется на практике для вывода больших сообщений (более подробно об этом читайте дальше в этой главе).

Вернемся к рассмотрению конструкции, обрамляющей PHP-код. Помимо уже известного вам тега <?php...?> встречаются еще три разновидности:

- <?...?>;
- <%...%>;
- <script language="php">...</script>.

Выбирая тот или иной вид конструкций, убедитесь в том, что настройки PHP позволяют вам их использовать. Например, короткие теги <?...?> не всегда включены по умолчанию (смотрите значение параметра short_ open_tag конфигурационного файла php.ini), поэтому их использование не рекомендуется. Конструкция <script language="php">...</script> так же, как и <?php...?> доступна всегда, но из-за своей громоздкости она применяется реже. Тег <%...%> был введен в версии PHP 4.0.3. Его работа зависит от параметра asp_tags. В нашей книге мы будем придерживаться тегов вида <?php...?>.

Стоит сказать еще несколько слов о закрывающем теге ?>. В PHP он воспринимается как символ точка с запятой (;), поэтому в конце последней строки его можно не ставить.

Профессиональная вставка

Рассмотрим еще один пример вывода строк в окно браузера (листинг 3.3).

Листинг 3.3 ▾ Профессиональная вставка

```
<?php
if ($expr)
{
echo "<b>вывод средствами PHP</b>";
}
```

```
else
{
?>
<b>Профессиональная вставка</b>
<?php
}
?>
```

В этой программе используется еще незнакомая вам конструкция `if … else`. Смысл ее состоит в том, что в зависимости от выражения `$expr` будет выполняться то или иное действие. В данном случае действиями являются выполнение инструкции `echo` и альтернативный вывод текста, называемый профессиональной вставкой. Суть ее заключается в том, что после закрывающего тега (`?>`) строки передаются для обработки браузером до тех пор, пока не встретится еще один открывающий тег(`<?php`). Профессиональная вставка в основном используется для вывода текста большого размера. Обратите внимание, что в нашем примере при значении `$expr` равном 1 будет выполняться только инструкция `echo`.

PHP и HTML

Так как текст, находящийся вне тегов и обрамляющий PHP-код, передается браузеру без обработки, то имеется возможность использовать HTML-теги (см. листинг 3.4).

Листинг 3.4 ▼ PHP-код в HTML-коде

```
<html>
<head>
   <title>PHP и HTML</title>
</head>
<body>
```

```
<b>
<?php
echo "Hello, World!";
?>
</b>
</body>
</html>
```

Результат выполнения этой программы представлен на рис. 3.3.

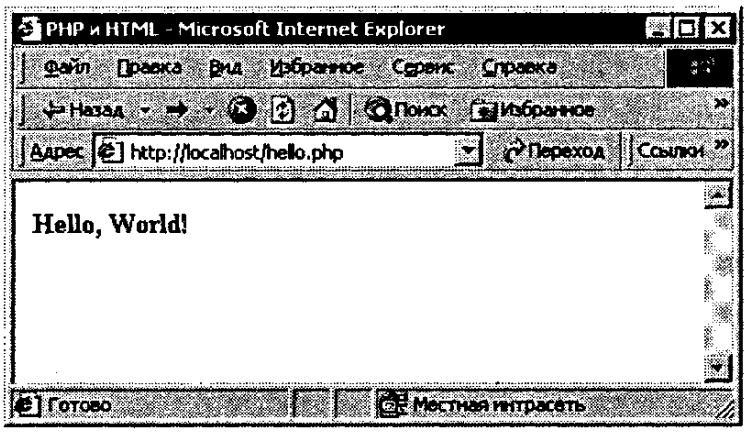

Рис. 3.3 ▾ PHP-код в HTML-коде.

Представленный пример программы является случаем вставки PHP-кода в HTML-код. Обратите внимание, что текст выводится жирным шрифтом, так как он находится внутри HTML-тегов ....

Помимо этого имеется возможность вставлять HTML-код в PHP-код (см. листинг 3.5).

Листинг 3.5 ▾ HTML-код в PHP-коде

```
<html>
<head>
   <title>PHP и HTML</title>
</head>
<body>
```

```
<?php
echo "<b>Hello, World!</b>";
?>
</body>
</html>
```

В данном случае теги ... находятся внутри конструкций <?php...?>, однако результат будет такой же, как в предыдущем примере. Единственное отличие заключается в том, что код работает немного медленнее.

Комментарии

Часто требуется внести пояснения в программный код или сделать небольшие пометки. Для этого применяют комментарии. Это специальные конструкции языка, позволяющие выделить ту часть кода, которая игнорируется при выполнении программы. Комментарии в PHP бывают однострочные и многострочные.

С помощью однострочных комментариев можно закомментировать все до конца строки или до закрывающего тега (?>). Рассмотрим в качестве примера листинг 3.6.

Листинг 3.6 ▼ Примеры комментариев в программе

```
<?php
//echo "Это однострочный комментарий PHP в стиле C++";
#echo "Это однострочный комментарий PHP в стиле Unix";
?>
// Эта строчка не является комментарием PHP
<!- Эта строчка является комментарием HTML ->
```

Обратите внимание на то, что при попытке закомментировать строку вне тегов <?php ... ?> у нас ничего не получится. Дело в том, что // и # – это конструкции языка PHP. Вне тегов они воспринимаются как обычный текст. В нашем случае можно воспользоваться комментариями HTML.

При использовании многострочных комментариев (/* ... */) игнорируется все, что находится внутри этой конструкции. Рассмотрим в качестве примера листинг.

Листинг 3.7 ▾ Ошибка использования многострочных комментариев

```
<?php
/*Текст внутри
этих конструкций
игнорируется*/
/* echo "Будет ошибка!!!"; /*Так нельзя*/ */
?>
```

Будьте внимательны при использовании многострочных комментариев. В данном примере произойдет ошибка! После нахождения открывающей части конструкции (/*), будет искаться ближайшая закрывающая (*/). В итоге не закомментированным остается только знак */, который вызовет ошибку.

В основном комментарии применяют для отладки и тестирования программы, а также для пояснения к коду. Надо сказать, что начинающие программисты не любят их использовать. Это естественно, так как задачи, которые перед ними стоят, обычно тривиальны и не требуют пояснений. Тем не менее, когда программа достигает огромных размеров, применение комментариев становится незаменимым.

Оформление кода программы

Самое главное, что требуется от программы, – это правильность выполнения поставленной задачи. Обычно оформление кода никого не интересует, кроме вас. Но теперь представьте, что условие задачи немного изменилось, и вам надо исправить программу. Естественно, если вы выполняете чей-то заказ, от вас потребуют сделать это в короткие сроки. Приведем в качестве примера листинг 3.8.

Листинг 3.8 ▼ Плохое оформление кода программы

```
<?php
$i=1;$j=2;if($i>1){if($j<1){echo"A";}else{echo"B";}}else{echo"C";}
?>
```

Этот фрагмент программы работает идеально. Я переставил местами всего два символа в этой строке, после чего при запуске программы возникала ошибка, и попросил своего друга-программиста ее исправить. У него это заняло около пяти минут. А теперь представьте, что таких строчек тысячи.

Оформление кода особенно важно, если работа ведется в команде, когда над программой работают несколько человек. Скорость и качество написания проекта напрямую зависит от взаимопонимания всех членов группы.

Все это и многое другое заставило программистов придерживаться каких-то стандартов при оформлении кода. Язык PHP не стал исключением. Для него был разработан PHP Coding Standard (автор Фредрик Кристиансен), которого мы будем придерживаться в нашей книге. Но стоит помнить, что это всего лишь рекомендации, а не требования.

В завершении этой главы дадим несколько советов, касающихся использования комментариев для оформления кода программы:

- для коротких пояснений используйте однострочные комментарии (см. листинг 3.9);

Листинг 3.9 ▼ Использование однострочных комментариев

```
<?php
// если $expr отлично от нуля, то вывести сообщение
if ($expr)
{
echo "Hello, world!"; // вывод сообщения
}
?>
```

- для развернутых пояснений используйте многострочные комментарии (см. листинг 3.10):

Листинг 3.10 ▾ Использование многострочных комментариев

```
<?php
/*
**********************************************************
*
*  В этой части программы содержится описание
*  функций для работы с базами данных
*
**********************************************************
*/
?>
```

- для комментирования больших блоков программ (при тестировании и отладки) старайтесь не использовать многострочные комментарии, так как может возникнуть ошибка, описанная в этой главе. Самый простой способ – использование конструкции (см. листинг 3.11);

Листинг 3.11 ▾ Комментирование больших блоков программы

```
<?php
if (0)
{
// большой блок программы
}
?>
```

- старайтесь придумывать комментарии, которые понятны не только вам, но и другим.

Заключение к главе

В этой главе мы познакомились с основами синтаксиса языка PHP. Напомним, что PHP-код обрабатывается Web-сервером, если он находится в файле с расширением, указанным в файле конфигурации Apache (в нашем случае *.php), и обрамлен специальными тегами, которых существует четыре вида. Настоятельно рекомендуем использовать теги вида `<?php ... ?>`, так как они почти всегда доступны. Все, что находится вне этих конструкций, в общем случае просто передается без изменения (исключение составляет профессиональная вставка).

Обратите внимание на тот факт, что PHP-код выполняется на стороне сервера, а результат отправляется клиенту. Еще раз выполните программу листинга 3.1, чтобы убедиться в этом.

Глава 4 Переменные, константы и типы данных

Наконец-то пришло время перейти от общего к частному. В этой главе мы познакомимся с основными элементами языка, без которых трудно представить себе серьезную программу на PHP, – имеются в виду переменные и константы. Если у вас есть опыт программирования и вы знакомы с этими понятиями, то все равно не следует пренебрегать данной главой, так как в PHP существует ряд особенностей при работе с ними.

Переменные – это...

В PHP, как и во многих других языках программирования, существует средство для хранения данных, которые могут изменяться в процессе выполнения программы, – это переменная. Любая переменная характеризуется именем (идентификатором) и значением.

Переменная – средство языка для хранения данных.

Для большей ясности разберем в качестве примера листинг 4.1.

Листинг 4.1 ▼ Пример переменной

```
<?php
$number = 5;     // присваиваем переменной значение 5
echo $number;    // вывод значения переменной $number
?>
```

В этом примере переменная имеет имя $number, а с помощью знака равенства (=) ее значение становится 5. Теперь вместо этого значения можно использовать имя $number, как это делается во второй строчке. В результате выполнения данной программы в окне браузера выведется число 5.

Но переменные способны не только хранить данные. По желанию их значение можно изменять (на то они и переменные) – листинг 4.2.

Листинг 4.2 ▼ Изменение значений переменной

```
<?php
$number = 18;              // присваиваем переменной значение 18
$number = $number + 2;     // увеличиваем значение переменной на 2
echo $number;              // выводит 20
?>
```

В этом случае значение переменной $number изменилось с 18 на 20 посредством прибавления 2. Итак, переменные могут не только хранить данные, но и допускают их изменение.

Обратимся к проблеме выбора имени переменной, так как в PHP существует ряд синтаксических правил на этот счет:

- любое имя переменной должно начинаться со знака доллара ($);
- после него может идти либо буква, либо знак подчеркивания (_), но не цифра (листинг 4.3);

Листинг 4.3 ▼ Выбор имени переменной

```
<?php
$2_var = 5;      // неправильное имя переменной
$_2_var = 5;     // так правильно
?>
```

➤➤ далее могут следовать буквы, цифры и символы подчеркивания в любой последовательности (знак пробела недопустим!);

➤➤ имена переменных чувствительны к регистру, то есть `$Number` и `$number` не будут эквивалентны.

Этот свод синтаксических правил является обязательным, так как отклонение от него вызовет ошибку, и программа просто не будет работать. Но так же существует ряд условных правил, которых большинство программистов стараются придерживаться. В нашем случае речь идет о PHP Coding Standard, о котором говорилось в главе 3. Согласно этому стандарту при выборе имени переменной используются символы в нижнем регистре, слова разделяются знаками подчеркивания (_) – листинг 4.4.

Листинг 4.4 ▼ Примеры названий переменных

```
<?php
$number;          // число
$color;           // цвет
$user_id;         // идентификационный номер пользователя
$max_woman_age;   // максимальный возраст женщины
?>
```

Старайтесь выбирать имя переменной так, чтобы по нему можно было понять смысл хранимой информации. Это помогает при работе с большими программами, когда количество переменных может быть больше сотни, а то и тысячи. И тем более это важно, если с вашим кодом будут работать другие люди. Но помните, что поговорка «кашу маслом не испортишь» в данном случае не подходит. Например, переменная `$it_is_variable_for_count_money` придаст вашему коду неструктурный вид и усложнит его восприятие.

Типы данных

В программировании существует такое понятие, как *тип данных*, который определяет множество допустимых значений и операций над ними.

Например, это могут быть целые или вещественные числа, строки различной длины и многое другое.

PHP поддерживает четыре скалярных (Integer, Double, Boolean, String) и два смешанных (Array, Object) типа данных. Рассмотрим некоторые из них подробно.

Integer

Тип данных Integer определяет множество целых чисел с определенными ограничениями по величине. Например, на 32-битных платформах диапазон составляет от –2 147 483 648 до 2 147 483 647 (листинг 4.5).

Листинг 4.5 ▼ Переменная типа Integer

```
<?php
$dec_int_number = 5;        // целочисленная переменная
echo 5                      // вывод целого числа
?>
```

Данные типа Integer могут быть представлены в различных системах счисления: восьмеричной, десятичной и шестнадцатеричной (листинг 4.6).

Листинг 4.6 ▼ Переменная типа Integer в различных системах счисления

```
<?php
// в десятичной системе все три числа равны 317
$oct_int_number = 0475;       // восьмеричные числа начинаются с 0
$dec_int_number = 317;        // эта запись десятичного числа
$hex_int_number = 0x13D       // шестнадцатеричные числа начинаются с 0x
?>
```

Double

Тип данных Double (или Float) соответствует множеству вещественных чисел, которые по-другому называются числами двойной точности или с плавающей точкой (листинг 4.7).

Листинг 4.7 ▾ Переменная типа Double

```
<?php
$first_double_number = 111.99;
$second_double_number = 4.5e3; // соответствует 4500
$third_double_number = 5E-5;   // соответствует 0,00005
?>
```

Переменные типа Double могут применяться при решении задач с большими числами (порядка 1.8e308), а также для расчетов высокой точности (до 14 знаков после точки).

Boolean

Boolean является самым простым типом данных. Он представляет множество, состоящее всего из двух значений: TRUE (истина) и FALSE (ложь) – листинг 4.8.

Листинг 4.8 ▾ Переменная типа Boolean

```
<?php
$boolean_var = TRUE;   // переменная логического типа
?>
```

Переменные типа Boolean обычно применяются для выяснения ложности или истинности какого-либо значения (об этом будет подробно рассказано позже). Стоит отметить, что значения TRUE и FALSE не чувствительны к регистру, то есть можно писать так, как показано в листинге 4.9.

Листинг 4.9 ▾ Особенности переменных типа Boolean

```
<?php
$boolean_var = False; // FALSE не чувствителен к регистру
?>
```

Тип данных Boolean появился в PHP 4.

Array

Тип данных Array рассматривается в главе 8.

String

В PHP тип данных String соответствует строке символов, длина которой практически не ограничена (листинг 4.10).

Листинг 4.10 ▼ Переменная типа String

```
<?php
$stroka = "Hello, World";      // переменная типа String
$nul_stroka = "";              // PHP поддерживает пустые строки
$stroka = 'Hi';                // строка может быть и в одинарных кавычках
?>
```

Более подробно об этом типе данных читайте в главе 9.

Object

Тип данных Object применяется в объектно-ориентированном программировании. В данной книге он не рассматривается.

Другие типы данных

Помимо скалярных и смешанных, PHP поддерживает два специальных (resource, NULL) типа данных, а также несколько псевдотипов (mixed, number, callback). Так как на практике они применяются очень редко, подробно рассматривать в этой книге мы их не будем.

Определение переменных

Во многих языках программирования (Pascal, Delphi и др.) при определении переменных нужно обязательно их объявлять (задавать конкретный тип данных), причем если попытаться присвоить значение переменной не соответствующее назначенному типу данных, то при компиляции программы будет возникать ошибка. В этом плане язык PHP более либерален. Здесь не обязательно задавать тип данных, так как он автоматически определяется, когда мы присваиваем значение

(смотрите примеры разобранные выше). Более того, в одной программе переменная может быть, например, числом и строкой. Тем не менее, такая вольность требует от программиста большей концентрации внимания, так как приходится следить за текущим типом данных. Для этого обычно применяют функцию (это понятие подробно разбирается в главе 7) gettype() – листинг 4.11.

Листинг 4.11 ▼ Определение типа данных

```
<html>
<head>
   <title>Определение типа данных</title>
</head>
<body>
<?php
$test_var = 15;           // переменная целого типа
echo gettype($test_var);  // выводит integer
echo "<br>";              // перевод строки
$test_var = 12.33;        // переменная вещественного типа
echo gettype($test_var);  // выводит double
echo "<br>";
$test_var = TRUE;         // переменная логического типа
echo gettype($test_var);  // выводит boolean
echo "<br>";
$test_var = "Hi";         // переменная строгого типа
echo gettype($test_var);  // выводит string
?>
</body>
</html>
```

Как вы, наверное, уже догадались, чтобы получить тип данных переменной, нужно указать ее имя в круглых скобках функции gettype().

Если требуется проверка переменной на соответствие определенному типу данных, то применяются функции `is_integer()`, `is_double()`, `is_string()`, `is_array()`, `is_object()` и `is_bool()` – листинг 4.12.

Листинг 4.12 ▼ Проверка соответствия определенному типу данных

```
<html>
<head>
   <title>Проверка соответствия определенному типу данных</title>
</head>
<body>
<?php
<?php
$test_var = 15;                 // переменная целого типа
echo is_integer($test_var);     // выводит 1
echo "<br>";
$test_var = 12.33;              // переменная вещественного типа
echo is_double($test_var);      // выводит 1
echo "<br>";
$test_var = TRUE;               // переменная логического типа
echo is_bool($test_var);        // выводит 1
echo "<br>";
$test_var = "Hi";               // переменная строгого типа
echo is_integer($test_var);     // выводит 0
?>
</body>
</html>
```

В случае соответствия типа переменной выводится 1, иначе – 0.

Изменение типа данных

В PHP имеется возможность изменять тип данных переменной. В этом случае происходит преобразование типов данных из одного в другой (листинг 4.13).

Листинг 4.13 ▾ Изменение типа данных

```
<html>
<head>
   <title> Изменение типа данных </title>
</head>
<body>
<?php
$test_var = 12.63;                  // присваиваем значение 12.63
echo gettype ($test_var);           // выводит double
echo "<br>";
echo $test_var;                     // выводит 12.63
echo "<br>";
settype ($test_var, string);        // устанавливаем тип String
echo gettype ($test_var);           // выводит string
echo "<br>";
echo $test_var;                     // выводит 12.63
echo "<br>";
settype ($test_var, integer);       // устанавливаем тип
echo gettype ($test_var);           // выводит integer
echo "<br>";
echo $test_var;                     // выводит 12
echo "<br>";
settype ($test_var, boolean);       // устанавливаем тип
```

```
echo gettype ($test_var);        // выводит boolean
echo "<br>";
echo $test_var;                  // выводит 1
echo "<br>";
?>
</body>
</html>
```

Итак, вначале мы присвоили значение переменной `$test_var` `12.63`, и автоматически ее тип стал Double, о чем свидетельствует вывод результата функции `gettype()` в окне браузера. Затем преобразовали тип переменной в String и опять вывели значение (теперь уже строка `12.63`). При переходе в Integer дробная часть отбрасывается и остается `12` (обратите внимание, что значение не округляется до большего целого!). И, наконец, при изменении типа переменной на Boolean результат становится `1`.

Нетрудно заметить, что преобразование типов данных осуществляется по определенным правилам. Приведем некоторые из них.

Преобразование в Boolean

Значения, преобразуемые в FALSE:

- целое число `0`;
- дробное число `0.0`;
- пустая строка или строка `0`.

Любые другие значения преобразуются в TRUE.

Преобразование в Integer

Преобразование осуществляется таким образом:

- значение FALSE преобразуется в 0, а TRUE – в 1;
- у вещественных чисел отбрасывается дробная часть (смотрите пример выше);
- если строка не начинается с цифры, то она преобразуется в 0. Иначе будет указанное целое число (листинг 4.14).

Листинг 4.14 ▾ Особенности преобразования в Integer

```
<html>
<head>
 <title> Особенности преобразования в Integer </title>
</head>
<body>
<?php
$test_var = "num10";
settype ($test_var, integer);
echo $test_var;                          // выводит 0
echo "<br>";
$test_var = "10num";
settype ($test_var, integer);
echo $test_var;                          // выводит 10
?>
</body>
</html>
```

Преобразование в String

Значение FALSE преобразуется в пустую строку, а TRUE – в 1.
Любые числа преобразуются в строку, содержащую цифры этих чисел, включая их степень.

Приведение типов данных

Приведение типов данных используется в тех случаях, когда изменение типа переменной не требуется (листинг 4.15).

Листинг 4.15 ▾ Приведение типов данных

```
<html>
<head>
   <title> Приведение типов данных </title>
```

```
</head>
<body>
<?php
$test_var = 12.63;
echo gettype ($test_var);     // выводит double
echo "<br>";
echo (integer)$test_var;      // выводит 12
echo "<br>";
echo gettype ($test_var);     // выводит double
?>
</body>
</html>
```

Обратите внимание, что преобразования типов данных функцией `settype()` и рассмотренной конструкцией эквивалентны. Отличие лишь в том, что `settype()` изменяет тип переменной, а рассмотренная конструкция создает временную копию требуемого типа.

Ссылки на переменные

В PHP есть такое понятие как *ссылка*. Для лучшего понимания, о чем идет речь, разберем в качестве примера листинг 4.16.

Листинг 4.16 ▼ Ссылки на переменные

```
<html>
<head>
 <title> Ссылки на переменные </title>
</head>
<body>
<?php
$a = 1;                  // $a имеет значение 1
```

```
$b = $a;                // в $b копируется значение $a
$c = &$a;               // $c является ссылкой на $a
$a = 5;                 // $a имеет значение 5
echo $a;                // выводит 5
echo "<br>";            // перевод строки
echo $b;                // выводит 1
echo "<br>";            // перевод строки
echo $c;                // выводит 5
?>
</body>
</html>
```

Итак, сначала мы присваиваем переменной $a значение 1. Затем копируем это значение в переменную $b. Теперь $a и $b имеют одинаковые значения, но они никак не связаны друг с другом, то есть при изменении $a переменная $b останется прежней, и наоборот. В следующей строчке используется еще не встречавшийся до этого символ &. В PHP он указывает на то, что создается ссылка $c на переменную $a. Теперь значения этих переменных будут постоянно равны друг другу. Этим и объясняется вывод значения (5, а не 1) переменной $c.

На самом деле применение ссылок на переменные не всегда себя оправдывает, так как выигрыш в производительности обычно невысокий, а шанс запутаться у неопытного программиста при этом резко возрастает. Поэтому не будем дальше углубляться в данную тему.

Динамические переменные

В PHP имеется возможность использовать переменные, имена которых содержат переменные. Это сложно понять с первого раза, поэтому сразу в качестве примера приведем листинг 4.17.

Листинг 4.17 ▼ Динамические переменные

```
<html>
<head>
   <title> Динамические переменные </title>
</head>
<body>
<?php
$name = "id";      // $name содержит строку "id"
$id = 5;           // $id содержит число 5
echo $$name;       // выводит 5
?>
</body>
</html>
```

Именно такие переменные, как $$name, называют динамическими. Они применяются, как правило, если требуется в ходе выполнения программы создавать много переменных. Рассмотрим листинг 4.18 в качестве примера.

Листинг 4.18 ▼ Альтернативные способы применения динамических переменных

```
<html>
<head>
<title>Альтернативные способы применения динамических переменных</title>
</head>
<body>
<?php
$name = "age";
$age = 18;
echo "<b>$$name</b><br>";     // выведет $age
echo "<b>${$name}</b><br>";   // выведет 18
```

```
echo "<b>${"age"}</b>" // выведет 18
?>
</body>
</html>
```

Здесь отражен еще один способ вывода динамической переменной. Если она находится внутри кавычек, то необходимо применять фигурные скобки ({}), иначе в окне браузера выведется строка $age. Также требуемого результата можно добиться с помощью фигурных скобок и строковой константы (смотри пример). Тем не менее применение динамических переменных, так же как и ссылок, может запутать начинающих (и не только их), поэтому продолжать не стоит.

Константы – это...

Употребляя в повседневной жизни слово «константа», мы имеем в виду постоянное значение. Это может быть число Пи (3,14) или температура кипения воды (100 °С). В PHP тоже имеется возможность использовать константы. Смысл их применения заключается в том, что обозначив определенное значение, мы можем использовать его на протяжении всего кода программы.

Константа – это неизменяемое значение.

Например, ваш друг Василий Пупкин создал Web-сайт и хочет, чтобы все знали имя администратора. При этом он находит самое простое и, на первый взгляд, верное решение (листинг 4.19).

Листинг 4.19 ▼ Вывод фамилии и имени администратора Web-сайта

```
<?php
echo "Администратор сайта: Пупкин Василий";      // вывод сообщения
?>
```

Давайте подумаем о последствиях принятия этого решения. Предположим, что Василий начал работать над другим проектом и переложил груз ответственности поддержания Web-сайта на ваши плечи.

Соответственно, надо изменить имя администратора. Скорее всего, вы станете искать по всем страницам сайта текст, содержащий строчку Пупкин Василий. После нахождения надо разобраться, менять ли ее на ваше имя или нет. Не трудно заметить, что решение, на первый взгляд, простой задачи занимает много времени и не гарантирует корректную работу программы. Все эти проблемы можно было бы избежать, если бы Василий применил константу для обозначения своего имени. Для этого надо выбрать имя константы (обычно его называют идентификатором), например ADMIN_NAME, после чего определить ее значение (в нашем случае Пупкин Василий).

Теперь решение задачи будет выглядеть так, как показано в листинге 4.20.

Листинг 4.20 ▼ Пример использования констант

```
<html>
<head>
   <title> Пример использования констант </title>
</head>
<body>
<?php
define("ADMIN_NAME", "Пупкин Василий");     // определение константы
echo "Администратор сайта: ";               // вывод сообщения
echo ADMIN_NAME;                            // вывод значения константы
?>
</body>
</html>
```

В этом случае идентификатор ADMIN_NAME будет заменен значением Пупкин Василий. Для того чтобы изменить имя администратора Web-сайта, потребуется только корректировка строчки с определением константы. Рассмотрим эту тему более подробно.

Определение констант

Для определения констант в PHP применяется функция define() – листинг 4.21.

Листинг 4.21 ▼ Определение констант

```
<html>
<head>
   <title> Определение констант </title>
</head>
<body>
<?php
define("ADMIN_NAME", "Пупкин Василий");        // фамилия и имя администратора
define("NUMBER_E", 2.71828);                   // число "e"
echo ADMIN_NAME;                               // выводит "Пупкин василий"
echo " знает, что число e*2 равно ";           // выводит строку
echo NUMBER_E*2;                               // выводит результат умножения
?>
</body>
</html>
```

Здесь использовались константы типа String и Double. Также можно определить логические и целочисленные константы. Заметим, что в этом примере константа используется при вычислениях. Однако стоит помнить, что ее значение не может быть изменено после определения. Например, выполнение программы, представленной в листинге 4.22, вызовет ошибку.

Листинг 4.22 ▼ Неправильное использование константы

```
<?php
define ("CONSTANT", 4);     // определение константы
CONSTANT = 2;               // попытка изменения значения константы
?>
```

Константы можно называть как прописными, так и заглавными буквами. Они чувствительны к регистру, то есть, например, ADMIN_NAME и ADMIN_name не эквивалентны. Как вы могли заметить, в нашем примере и во всех последующих, где встречаются константы, мы называем их именами в верхнем регистре, отделяя слова знаком подчеркивания (_). Тем самым мы придерживаемся PHP Coding Standard, о котором шла речь в предыдущей главе.

Заранее забегая вперед, надо сказать, что PHP включает несколько встроенных констант (подробнее о них читайте в разделе «Предопределенные константы»). Поэтому возникает опасность совпадения имен встроенной и вашей константы, что вызовет предупреждения об ошибке. Поэтому будьте внимательны при выборе имени. Чтобы узнать существует ли константа, можно использовать функцию defined(). Она возвращает 1, если константа определена, и 0 – в противном случае. Приведем небольшой пример использования функции defined() – листинг 4.23.

Листинг 4.23 ▼ Проверка на существование константы

```
<html>
<head>
   <title> Проверка на существование константы </title>
</head>
<body>
<?php
// проверка на существование константы
if (defined("CONSTANT"))
{
   // вывести сообщение, если константа существует
   echo "Константа определена"; }
?>
</body>
</html>
```

Предопределенные константы

В PHP заранее определен ряд констант. Они называются предопределенными. Их основное назначение – хранить информацию о системе. Например, PHP_VERSION и PHP_OS содержат соответственно версию PHP и название операционной системы, на которую установлен сервер.

Существуют так же несколько «волшебных» констант. Они могут менять свое значение в зависимости от их использования. Например, константы __LINE__ и __FILE__ содержат в себе соответственно номер строки и имя файла сценария. Обратите внимание, что эти константы можно писать как прописными буквами, так и заглавными. Листинг 4.24 – простой пример использования предопределенных констант.

Листинг 4.24 ▾ Предопределенные константы

```
<html>
<head>
   <title> предопределенные константы </title>
</head>
<body>
<?php
echo "Версия PHP: ";
echo PHP_VERSION;
echo "<br>";
echo "Операционная система: ";
echo PHP_OS;
echo "<br>";
echo "Номер строки:";
echo __LINE__;
echo "<br>";
echo "Имя запускаемого файла сценария: ";
echo __file__;
```

```
?>
</body>
</html>
```

Заключение к главе

В этой главе мы познакомились с такими понятиями как переменная, константа и тип данных. Их использование является неотъемлемой частью не только Web-программирования, но и программирования вообще. Поэтому настоятельно рекомендуется внимательно изучить эту главу.

Глава 5 Операторы

Итак, вы познакомились с такими понятиями, как константа и переменная. Теперь пришло время изучить операции, с помощью которых можно работать с их значениями.

Операторы – это...

В PHP, как и во многих других языках программирования, существуют *операторы*. На самом деле мы уже применяли операторы в предыдущих главах, но не акцентировали на этом внимание. Обычно их использование не вызывает трудностей у программистов, так как они зачастую очень похожи на те операции, которые мы делаем в повседневной жизни: в частности, речь идет об арифметических операторах, смысл которых известен нам со школьного возраста. Например, операции сложения, вычитания, умножения и деления практически идентичны тем, что используются в PHP.

Оператор – конструкция языка, предназначенная для получения нового значения.

Не сложно понять, что основной смысл операторов – получение нового значения. В свою очередь это значение обычно записывается

в новую переменную посредством оператора присваивания (=), который очень часто использовался в предыдущей главе для инициализации переменных.

Как вы наверно уже догадались, операторы бывают различных типов в зависимости от класса решаемых задач (сравнения, логические, поразрядные, строковые и другие).

Значения, к которым применяют операторы, обычно называют *операндами.* Например:

```
$num = CASH - 3;
```

В этом случае знак равенства (=) и знак вычитания (-) являются операторами, а переменная $num, константа CASH и число 3 – операндами. Всю эту комбинацию называют *выражением.*

Выражение – сочетание операторов и операндов.

Далее в этой книге будет использоваться именно данная терминология, поэтому обязательно уясните, что чем является.

Операторы также классифицируются по количеству операндов, на которые они действуют. Обычно мы встречаемся с бинарными операторами, такими как сложение, вычитание и др. Они задействуют два операнда. Но в PHP есть и унарные операторы (используют один операнд), и тернарные (три операнда). Мы еще подробно рассмотрим их в этой главе.

Оператор присваивания

Оператор присваивания, наверное, самый распространенный оператор в программировании. Он является бинарным и состоит из одного знака равенства (=). Приведем простейший пример его использования:

```
$number = 5;
```

В данном случае он задействует два операнда: переменную $number и число 5. Смысл его действия заключается в следующем. Оператор копирует значение правого операнда и записывает его в левый операнд. Поэтому убедитесь, что левый операнд может изменяться.

Рассмотрим пример:

```
6 = 4;              // будет ошибка
CONSTANTA = 3;      // тоже будет ошибка
```

Очевидно, что данные строки не будут выполняться в программе, так как константы нельзя изменять. Но это не означает, что константы не могут быть использованы в качестве правого операнда. Пример:

```
$number = CONSTANTA;   // ошибки не будет
```

В этом случае ошибка возникать не будет, так как переменная может изменяться.

Еще раз повторим, что использование операторов влечет за собой появление нового значения (в этом их суть). Оператор присваивания не является исключением, поэтому возможны конструкции такого вида:

```
$a = 2 + ($b = 3);     // $b имеет значение 3, $a - 5
```

В этом примере сначала переменной $b присвоится значение 3. Затем это выражение ($b = 3) примет значение правого операнда (3). После чего выполнится операция сложения, и в итоге переменная $a будет иметь значение 5. Но помните, что применение конструкций подобного вида не рекомендуется – они усложняют восприятие кода программы.

Арифметические операторы

Арифметические операторы выполняют операции, которые соответствуют тем, что существуют в математике. Например, оператор сложения (+) находит сумму двух операндов, а оператор вычитания (-) – разность. Перечислим все арифметические операторы PHP:

- сложения $a + $b;
- вычитания $a - $b;
- умножения $a * $b;
- деления $a / $b;
- остаток от деления $a % $b.

При выполнении операции деления, естественно, нужно отслеживать значение правого операнда (делителя), так как если он равен 0, возникает предупреждение об ошибке. При нахождении остатка от деления используйте целые числа, иначе результат может быть неверным.

Операторы отношения

Операторы отношения применяются для сравнения значений. Обычно сравнивают числа, но PHP позволяет применять их и к строкам. Выражения, содержащие операторы отношения (или сравнения), всегда имеют значение типа Boolean, то есть TRUE или FALSE. Они часто применяются в конструкциях типа `if..else` и подобных им. Об этом читайте в следующей главе.

Приведем все операторы отношения, доступные в PHP:

- `$a == $b` – проверка на равенство
  `$a` равняется `$b` - TRUE
  `$a` не равняется `$b` - FALSE;
- `$a != $b` – проверка на неравенство
  `$a` не равняется `$b` – TRUE
  `$a` равняется `$b` - FALSE;
- `$a < $b` – проверка на меньше
  `$a` меньше `$b` - TRUE
  `$a` больше либо равно `$b` - FALSE;
- `$a > $b` – проверка на больше
  `$a` больше `$b` - TRUE
  `$a` меньше либо равно `$b` - FALSE;
- `$a <= $b` – проверка на меньше либо равно
  `$a` меньше либо равно `$b` - TRUE
  `$a` больше `$b` - FALSE;
- `$a >= $b` – проверка на больше либо равно
  `$a` больше либо равно `$b` - TRUE
  `$a` меньше `$b` - FALSE;

- $a === $b – проверка на идентичность
  $a идентичны $b - TRUE
  $a неидентичны $b - FALSE.

Приведенные операторы очень простые, хотя пояснений требует, наверное, последний из них (===). Этот оператор появился только в PHP 4. Он требует от своих операндов не только одинаковых значений, но и совпадение типа данных (листинг 5.1).

Листинг 5.1 ▾ Оператор проверки на идентичность

```
<html>
<head>
   <title> Оператор проверки на идентичность </title>
</head>
<body>
<?php
$a = 4;          // переменная типа Integer
$b = "4";        // переменная типа String
echo "==: ";
echo ($a==$b);   // выводит 1 (TRUE)
echo "<br>";
echo "===: ";
echo ($a===$b); // выводит "" (FALSE)
?>
</body>
</html>
```

В данном случае значения переменных равны, но не равны их типы, поэтому они считаются неэквивалентными.

Логические операторы

Логические операторы выполняют операции булевской алгебры, то есть они работают со значениями типа Boolean. На самом деле в качестве операндов могут быть значения с другим типом данных. Дело в том, что они просто преобразовываются к типу Boolean по правилам, которые приводились в предыдущей главе. В результате выполнения логических операторов также получаются значения TRUE или FALSE. Рассмотрим каждый из них подробнее:

- $a && $b – логическое «и»;
- $a and $b – тоже логическое «и» (табл. 5.1);

Таблица 5.1 ▼ Логическое «и»

$a	$b	Результат
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

- $a || $b – логическое «или»;
- $a or $b – тоже логическое «или» (табл. 5.2);

Таблица 5.2 ▼ Логическое «или»

$a	$b	Результат
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

- $a XOR $b – исключающее «или» (табл. 5.3);

Таблица 5.3 ▾ Исключающее «или»

$a	$b	Результат
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

➳ ! $a – логическое «не» (табл. 5.4).

Таблица 5.4 ▾ Логическое «не»

$a	Результат
FALSE	TRUE
TRUE	FALSE

Наличие двух операторов на одну логическую операцию мы поясним в этой главе позже, когда будем рассматривать приоритетность.

Обратите внимание, что среди логических операторов есть унарный, то есть имеющий только один операнд (логическое «не»). В этой главе мы еще не раз будем встречаться с ними.

Логические операторы в сочетании с операторами отношения очень часто используются при ветвлении `if...else`, циклах `while...do` и в других конструкциях, где встречаются сложные логические условия.

Поразрядные операторы

Поразрядные операторы выполняют логические операции, но не между операндами, а между их разрядами в битовом представлении. Очень легко с этими операциями разобраться на примере листинга 5.2.

Листинг 5.2 ▼ Поразрядные операторы

```
<html>
<head>
   <title> Поразрядные операторы </title>
</head>
<body>
<?php
echo (10 & 5);  // выведет 0
?>
</body>
</html>
```

Давайте разберемся, почему получился такой удивительный результат. Битовое представление – это есть ни что иное как запись в двоичной системе счисления. Число 10 записывается как 1010, а 5 – 0101. Затем, начиная с младшего разряда, начинает выполняться логическое «и». В результате и получается число 0000, которое соответствует десятичному числу 0. Приведем все поразрядные операторы PHP:

- $a & $b – поразрядное «и»;
- $a | $b – поразрядное «или»;
- $a ^ $b – поразрядное исключающее «или»;
- ~ $a – поразрядное «не»;
- $a << $b – поразрядный сдвиг влево;
- $a >> $b – поразрядное сдвиг вправо.

В качестве операндов могут выступать не только числа, но и строки. В этом случае используется код ASCII каждого символа. Но надо сказать, что для такой возможности трудно найти применение, поэтому подробно тему применения поразрядных операторов к строкам рассматривать не будем.

Строковые операторы

Среди строковых операторов можно выделить оператор конкатенации (точка). Это сложно произносимое слово имеет довольно простой смысл – оператор просто соединяет две строки (листинг 5.3).

Листинг 5.3 ▾ Оператор конкатенации

```
<html>
<head>
   <title> Оператор конкатенации </title>
</head>
<body>
<?php
echo "I"." like "."PHP";     // выводит "I like PHP"
?>
</body>
</html>
```

Результатом конкатенации строк всегда будет значение типа String. Операнды могут быть различных типов, но в итоге они преобразуются в строку (листинг 5.4).

Листинг 5.4 ▾ Особенности работы оператора конкатенации

```
<html>
<head>
   <title> Особенности работы оператора конкатенации </title>
</head>
<body>
<?php
$b = 123 . 456;     // $b - строка
$a = $b . "789";       // $a тоже строка
```

```
echo $b;                // выведет 123456
echo "<br>";
echo $a;                // выведет 123456789
?>
</body>
</html>
```

На этом обзор основных операторов PHP закончен. Далее перечисляются менее используемые операторы с короткими пояснениями.

Другие операторы

Оператор подавления ошибок

Оператор подавления ошибок (@) применяется для отладки сценариев PHP. Если поставить его перед выражением, то любые, возникающие в нем ошибки или предупреждения, не будут выводиться в окне браузера (листинг 5.5).

Листинг 5.5 ▼ Подавление ошибок

```
<?php
@$a = 5/0;      // подавляет ошибку деления на 0
?>
```

Операторы увеличения и уменьшения

Операторы увеличения и уменьшения (инкремент и декремент) используются скорее для удобства и красоты, чем для эффективности. Выглядят они следующим образом: «++» и «--». Эти операторы соответственно увеличивают операнд и уменьшают его на единицу. Заметьте, что они являются унарными. Причем положение оператора может быть как слева, так и справа. В чем отличие разберемся на примере листинга 5.6.

Листинг 5.6 ▾ Особенности работы оператора конкатенации

```
<html>
<head>
   <title> Особенности работы оператора конкатенации </title>
</head>
<body>
<?php
$a = 1;
$b = 1;
$c = $a++ + 2;  // оператор справа
$d = ++$b + 2;  // оператор слева
echo $a;        // выводит 2
echo "<br>";
echo $b;        // выводит 2
echo "<br>";
echo $c;        // выводит 3
echo "<br>";
echo $d;        // выводит 4
?>
</body>
</html>
```

Итак, в первом случае оператор находится слева от операнда ($a). При этом сначала выполняется сложение, а затем уже увеличение операнда. Во втором случае все происходит наоборот. Этим и объясняется результат выполнения программы. Операторы инкремента и декремента очень часто используются в организации цикла for, который будет рассмотрен в следующей главе.

Сокращенная запись присвоения переменных

Эти операторы предназначены, наверное, для особо ленивых программистов. Они сочетают в себе оператор присваивания и арифметические операции, а также конкатенацию. Например, `$a += 1` эквивалентно выражению `$a = $a + 1`. Приведем полный список этих операторов:

- `$a += $b` эквивалент `$a = $a + $b;`
- `$a -= $b` эквивалент `$a = $a - $b;`
- `$a /= $b` эквивалент `$a = $a / $b;`
- `$a *= $b` эквивалент `$a = $a * $b;`
- `$a %= $b` эквивалент `$a = $a % $b;`
- `$a .= $b` эквивалент `$a = $a . $b.`

Приоритетность и ассоциативность

При использовании операторов нужно обязательно быть знакомым с такими понятиями, как *ассоциативность* и *приоритетность*. Иначе можно долго сидеть и думать, почему не получается верный результат вычислений.

На самом деле с этим понятием приоритетности мы сталкивались еще в школе на уроках математики. Например, нужно посчитать значение выражения 1+2*3. Если не обращать на порядок вычисления внимания, то мы сложили бы сначала 1+2=3 и умножили результат на 3. В итоге получили бы 9. Но со школы известно, что сначала следует выполнить операцию умножения, а уж потом сложения. И правильный ответ будет 7. Именно в порядке вычисления различных операций заключается смысл приоритетности.

Приоритетность – свойство, определяющее порядок вычисления различных операций.

Свойство ассоциативности используется в тех случаях, когда требуется выяснить порядок вычисления операций при одинаковых приоритетах (листинг 5.7).

Листинг 5.7 ▼ Порядок вычисления

```
<?php
$num = 27/9/3;  // эквивалентно (27/9)/3
```

```
echo $num;              // выведет 1
?>
```

В данном случае выведется число 1, так как выполнение операторов деления начинается слева. В этом случае говорят, что оператор ассоциативен слева.

Ассоциативность – свойство, определяющее порядок вычисления операций при одинаковых приоритетах.

В таблице 5.1 отражены приоритеты некоторых операторов PHP, а также их ассоциативность.

Таблица 5.5 ▾ Приоритеты операторов PHP

Операторы	Ассоциативность
++ -	Правая
/ * %	Левая
+ - .	Левая
<< >>	Левая
< <= => >	Не ассоциативны
== === !=	Не ассоциативны
&	Левая
^	Левая
|	Левая
&&	Левая
||	Левая
= += -= /= *= %= .=	Правая
and	Левая
xor	Левая
or	Левая

Заключение к главе

В этой главе мы познакомились с понятиями оператор, операнд и выражение. Также мы разобрали наиболее часто употребляемые операторы PHP, их приоритетность и ассоциативность, и рассмотрели множество примеров с их использованием.

Управляющие операторы PHP

В этой главе мы познакомимся с управляющими операторами языка. По сравнению с операторами, рассмотренными в предыдущей главе, они более функциональны. О широком применении этих операторов говорит тот факт, что они встречаются почти во всех серьезных проектах.

Управляющие операторы можно разбить на три группы: условные, цикла и безусловные. С их помощью программа может самостоятельно принимать решения, многократно выполнять определенные действия и прекращать работу, когда это необходимо.

Эта тема является очень важной для дальнейшего понимания материала, так как большинство крупных примеров в этой книге содержит управляющие операторы.

Условные операторы

Говоря об условных операторах, невольно вспоминаешь богатыря Илью Муромца из детской сказки, который стоит перед камнем и думает, куда ему ехать. В программировании «куда ехать» обычно определяется по некоторому заданному условию. Наверное, самый распространенный

случай использования условных операторов – это проверка пароля. Например, каждый день миллионы людей проверяют электронную почту. Но получить доступ к ней можно только при совпадении комбинации логин и пароль с той, которая зарегистрирована на почтовом сервере. Это и является условием, по которому будет определяться «куда ехать», то есть давать доступ к почте или нет.

После такой сказочной вводной части перейдем к конкретному изучению управляющих операторов, среди которых `if` является самым распространенным.

Оператор If

Действительно, этот оператор можно встретить во многих языках программирования. В этом плане PHP не стал исключением и предлагает очень простой и понятный синтаксис:

```
if (выражение) действие; // выполняется, если выражение равно TRUE
```

Вся конструкция начинается со слова `if`. Далее в круглых скобках записывается выражение типа Boolean. Если оно равно TRUE, то выполняется действие, которое следует сразу за закрывающей круглой скобкой, иначе действие игнорируется. Например, если вам больше шестнадцати лет, то вы можете получить паспорт (листинг 6.1).

Листинг 6.1 ▼ Условный оператор if

```
<html>
<head>
   <title> Условный оператор if </title>
</head>
<body>
<?php
if ( $age > 16 ) echo "А у вас есть паспорт?";
?>
</body>
</html>
```

Итак, при значении переменной $age (например, 18) больше 16 выражение в круглых скобках будет равно TRUE, а значит выполнится вывод сообщения. В противном случае окно браузера будет пустым.

Стоит сказать несколько слов о выражении в круглых скобках. На самом деле изначально оно может иметь не только булевское значение. Например, на практике можно встретить такие варианты, как, например, в листинге 6.2.

Листинг 6.2 ▾ Преобразование выражения в круглых скобках

```
<html>
<head>
   <title> Преобразование выражения в круглых скобках </title>
</head>
<body>
<?php
if ( 1 ) echo "Выводится";
if ( "1" ) echo "Выводится";
if ( "0" ) echo "Не выводится";
if ( 0 ) echo "Не выводится";
?>
</body>
</html>
```

Здесь мы видим строковые и числовые выражения, также могут быть массивы, объекты и непосредственно встроенные константы TRUE и FALSE. В данном случае все эти значения преобразуются к типу Boolean по правилам, рассмотренным в главе 4. Помните, что в итоге мы имеем дело только со значениями TRUE и FALSE. Также стоит отметить, что употребление выражений по типу отличных от Boolean, за исключением 0 и 1, не рекомендуется, так как при этом отсутствует наглядность.

Иногда мы встречаемся с задачами, в которых ложному выражению тоже соответствует определенное действие. Например, если

ваша подруга свободна в воскресенье, то вы пойдете с ней в кино, а если нет, то будете смотреть телевизор дома. Заметьте, что и в том, и в другом случае выполняется действие. В таких случаях применяют оператор `if` в сочетании с ключевым словом `else`:

```
if (выражение) действие;  // выполняется, если выражение равно TRUE
else действие;            // выполняется, если выражение равно FALSE
```

Если выражение имеет значение FALSE, то выполняется действие, которое следует за словом `else`. Приведем листинг 6.3 – простой пример сравнения двух чисел.

Листинг 6.3 ▼ Сравнение двух чисел

```
<html>
<head>
   <title> Сравнение двух чисел </title>
</head>
<body>
<?php
$a = 1;
$b = 0;
if ($a > $b) echo "a больше b";
else echo "a меньше, либо равно b";
?>
</body>
</html>
```

Допустим, что `$a` меньше `$b` (к примеру, `0<1`), тогда значение выражения будет равно FALSE, и выполнится действие, соответствующее `else`.

До этого времени у вас могло сложиться мнение, что при использовании оператора `if` выполняется всего одна команда. В этой книге применяется термин *действие*, которое подразумевает одну или множество команд. Например, можно вывести еще одно сообщение (листинг 6.4).

Листинг 6.4 ▾ Выполнение нескольких команд

```
<html>
<head>
   <title> выполнение нескольких команд </title>
</head>
<body>
<?php
$a = 1;
$b = 0;
if ($a > $b)
{
echo "a больше b";
echo "Это тоже будет выведено";
}
else echo "a меньше, либо равно b";
?>
</body>
</html>
```

Чтобы выполнить несколько команд, применяются фигурные скобки ({}). В нашем примере переменная $a больше $b, поэтому значение выражения ($a > $b) будет равно TRUE. В связи с этим выполнится соответствующий блок вывода двух сообщений. Если вы попытаетесь опустить фигурные скобки, то это либо вызовет ошибку, как в нашем примере, либо результат выполнения оператора if будет не тем, которого вы ожидаете (листинг 6.5).

Листинг 6.5 ▾ Особенности работы оператора if

```
<html>
<head>
```

```
    <title> Особенности работы оператора if </title>
</head>
<body>
<?php
$a = 1;
$b = 0;
if ($a > $b)
echo "a больше b";
else
echo "a меньше, либо равно b";
echo "Это будет выведено";
?>
</body>
</html>
```

В данном случае вывод второго сообщения в блоке `else` осуществится вне зависимости от значений переменных `$a` и `$b`, так как оно не относится к конструкции `if..else`. Чтобы избежать подобных ошибок, лучше всегда использовать фигурные скобки, даже если выполняется всего одна команда.

Если вы все-таки иногда пренебрегаете фигурными скобками, то вам необходимо знать еще одну особенность работы оператора `if`. Запомните, что ключевое слово `else` всегда относится к ближайшему `if` (листинг 6.6).

Листинг 6.6 ▼ Особенности работы оператора if

```
<html>
<head>
    <title> Особенности работы оператора if </title>
</head>
<body>
```

```
<?php
$a = 2;
$b = 0;
if ($a > $b)
if (($a - $b) == 1) echo "разность между a и b равна 1";
else echo "a меньше, либо равно b";
?>
</body>
</html>
```

В этом случае выведется a меньше либо равно b, так как else относится не к первому if, а ко второму. Для правильной работы этой программы нужно использовать фигурные скобки (листинг 6.7).

Листинг 6.7 ▾ Особенности работы оператора if

```
<html>
<head>
   <title> Особенности работы оператора if </title>
</head>
<body>
<?php
$a = 2;
$b = 0;
if ($a > $b)
{
if (($a - $b) == 1) echo "разность между a и b равна 1";
}
else echo "a меньше, либо равно b";
?>
</body>
</html>
```

Помните, что фигурные скобки помогают избежать путаницы при использовании оператора if. Также они делают код программы более понятным.

Elseif

Для расширения возможностей условного оператора if в PHP ввели конструкцию elseif. В общем случае она выглядит так:

```
if (выражение_1) действие;          // выполняется, если выражение_1
// равно TRUE
elseif (выражение_2) действие;      // выполняется, если
// выражение_1 равно FALSE и
// выражение_2 равно TRUE
else действие;                      // выполняется, если
// выражение_1 равно FALSE и
// выражение_2 равно FALSE
```

Итак, данная конструкция позволяет проверить альтернативные условия. В действительности она введена для того, чтобы избежать многократной вложенности операторов if. Приведем листинг 6.8 – простой пример использования elseif:

Листинг 6.8 ▼ Оператор Elseif

```
<html>
<head>
   <title> Оператор Elseif </title>
</head>
<body>
<?php
if ($a > $b)    // $a больше $b?
{
 echo "a больше b";        // если да, то выводим и остальное пропускаем
```

```
}
elseif ($a == $b)          // если нет, то $a равно $b?
{
 echo "a равно b";         // если да, то выводим и остальное пропускаем
}
else
{
 echo "a меньше b";        // если нет выводим
}
?>
</body>
</html>
```

С помощью этой программы можно выяснить отношение $a к $b. На самом деле `elseif` применяется не настолько часто, чтобы рассказывать о нем более подробно, поэтому перейдем к рассмотрению следующего оператора.

Switch

К сожалению, управляющий оператор `if` даже в сочетании с ключевыми словами `else` и `elseif` иногда может оказаться неэффективным при решении определенного класса задач. Например, в книжный магазин завезли несколько новых книг. Нужно написать PHP-приложение, которое выводит фамилию автора книги, указанную пользователем. Прежде чем ваши руки устремятся к клавиатуре, давайте немного поразмышляем о способе решения.

Итак, представим наше PHP-приложение в виде «черного ящика», то есть имеются входные и выходные данные, а что внутри пока неизвестно. На данный момент в нашем распоряжении есть оператор `if`, с помощью которого задача решается следующим образом. Так как от пользователя в программу поступает название книги, то можно последовательно сравнивать его со всеми имеющимися названиями и при совпадении выводить соответствующую фамилию (листинг 6.9).

Листинг 6.9 ▼ Оператор Switch

```
<html>
<head>
   <title> Оператор Switch </title>
</head>
<body>
<?php
$book_name = "Самоучитель по PHP";              // входные данные
if ( $book_name == "Самоучитель по Perl" )
{
    echo "Автор: Петров";                       // выходные данные
}
elseif ( $book_name == "Самоучитель по ASP" )
{
    echo "Автор: Иванов";                        // выходные данные
}
elseif ( $book_name == "Самоучитель по PHP" )
{
    echo "Автор: Сидоров";                      // выходные данные
}
?>
</body>
</html>
```

Сейчас нас не интересует способ поступления названия книги, поэтому оно просто хранится в переменной $book_name. Чтобы избежать многократного выполнения оператора if, мы использовали его в сочетании с ключевым словом elseif. В результате программа выведет фамилию Сидоров, так как выражение $book_name == "Самоучитель по PHP" имеет значение TRUE, а все предыдущие – FALSE.

Несложно заметить, что конструкции подобного типа очень громоздки, так как они скорее предназначены для малого количества сравнений. В данном случае применение оператора `if` выглядит искусственным.

Для более логичного и эффективного решения подобных задач применяется управляющий оператор выбора `switch`:

```
switch (выражение)
{
   case выражение : действие;
   break;
   ...
   case выражение : действие;
   break;

   default : действие;
}
```

Вся конструкция начинается со слова `switch`. Далее в круглых скобках следует выражение, которое, в отличие от выражения оператора `if`, может быть не только типа Boolean, но и Integer, Double и String. От его значения зависит, какие действия выполнять. В нашем случае это может быть переменная `$book_name`, так как именно по ней определяется какую фамилию выводить.

После закрывающей круглой скобки следует блок, в котором находятся выражения для сравнения и соответствующие им действия. Синтаксически это оформляется с помощью специального слова `case` и знака двоеточия (смотрите пример).

Оператор `break` служит для того, чтобы выйти из конструкции `switch` в случае, когда значения выражений совпали. Подробно о нем читайте дальше в этой главе.

После слова `default` следует действие, которое нужно выполнить, если совпадений не было. Например, вывести сообщение, что запрашиваемой книги нет в магазине.

Решение задачи с использованием оператора switch будет выглядеть так, как показано на примере листинга 6.10.

Листинг 6.10 ▼ Решение задачи с помощью оператора Switch

```
<html>
<head>
   <title> Оператор Switch </title>
</head>
<body>
<?php
$book_name = "Самоучитель по PHP";
switch ( $book_name )
{
   // выводится, если $book_name имеет значение "Самоучитель по Perl"
   case "Самоучитель по Perl": echo "Автор: Петров";
   // выход из конструкции switch
break;
   case "Самоучитель по ASP": echo "Автор: Иванов";
   break;
   case "Самоучитель по PHP": echo "Автор: Сидоров";
   break;
   // сообщение выведется, если не было совпадений
   default: echo "Такой книги в наличии нет";
}
?>
</body>
</html>
```

Надо отметить, что присутствие в конструкции switch слов default и break не является обязательным. Если опустить слово default, то в

случае, когда совпадений не было, никаких действий просто не выполняется. Интереснее дело обстоит с оператором break (листинг 6.11).

Листинг 6.11 ▾ Особенности оператора Break

```
<html>
<head>
   <title> Особенности оператора Break </title>
</head>
<body>
<?php
$book_name = "Самоучитель по Perl";
switch ( $book_name )
{
   case "Самоучитель по Perl": echo "Автор: Петров";
   case "Самоучитель по ASP": echo "Автор: Иванов";
   break;
   case "Самоучитель по PHP": echo "Автор: Сидоров";
   break;
   default: echo "Такой книги в наличии нет";
}
?>
</body>
</html>
```

В результате выведутся фамилии Петров и Иванов. Другими словами, если совпали выражения и опущен оператор break, то выполнятся все действия вплоть до следующего оператора break. В нашем случае это привело к неправильной работе программы. Но есть случаи, когда отсутствие break помогает избежать многократного повторения в конструкции switch. Например, Петров написал не только самоучитель по Perl, но и еще несколько книг, тогда задачу можно решить так, как показано на примере листинга 6.12.

Листинг 6.12 ▼ Особенности оператора Break

```
<html>
<head>
   <title> Особенности оператора Break </title>
</head>
<body>
<?php
$book_name = "Информатика в школе";
switch ( $book_name )
{
   case "Самоучитель по Perl":
   case "Информатика в школе":
   case "Программирование в Internet": echo "Автор: Петров";
   break;
   case "Самоучитель по ASP": echo "Автор: Иванов";
   break;
   case "Самоучитель по PHP": echo "Автор: Сидоров";
   break;
   default: echo "Такой книги в наличии нет";
}
?>
</body>
</html>
```

В этом случае если пользователь ввел название одной из трех книг Петрова, выведется его фамилия, так как отсутствует оператор break. Стоит отметить, что таким образом задача решается намного эффективнее, чем многократное повторение операторов break и вывода фамилии Петров.

Нельзя не сказать еще несколько слов о выражении, следующим за case. В отличие от многих других языков программирования, оно может иметь любой скалярный тип данных. Например, в решениях наших задач мы использовали строковые выражения.

Операторы цикла

Без операторов цикла вообще трудно себе представить программирование. Например, нужно вывести числа от 1 до 100. Есть идеи? Наверное, кроме как написать сто раз подряд конструкцию echo, больше ничего не приходит в голову. Чтобы эффективно решать подобные задачи как раз и применяют операторы цикла.

For

Синтаксис оператора цикла for выглядит следующим образом:

```
for (выражение_1; выражение_2; выражение_3) действие;
```

Вся конструкция начинается со слова for. Далее в круглых скобках следуют через точку с запятой три выражения, после чего записывается выполняемое действие. Итак, чтобы понять смысл этих выражений рассмотрим листинг 6.13.

Листинг 6.13 ▾ Оператор for

```
<html>
<head>
   <title> Оператор for </title>
</head>
<body>
<?php
for ($i = 1; $i <= 100; $i++)
{
   echo $i;
```

```
}
?>
</body>
</html>
```

Эта программа решает задачу вывода чисел от 1 до 100. Первое выражение задает начальные данные и определяется один раз перед входом в цикл: в данном случае переменной `$i` присваивается значение 1. Второе выражение определяет условие нахождения в цикле. Другими словами, до тех пор, пока оно равно TRUE, программа будет продолжать циклично выполнять вывод значения переменной. И наконец, третье выражение представляет собой действие, которое нужно выполнить по завершению итерации. Здесь это увеличение переменной `$i` на единицу.

Итак, рассмотрим ход выполнения этой программы. Сначала значением переменной `$i` становится 1. Затем производится операция сравнения (вычисляется второе выражение), после чего, если результат TRUE, выполняется вывод сообщения и третье выражение, иначе выходим из цикла. Следующая итерация начинается уже с вычисления логического выражения.

На самом деле применение цикла `for` на практике зачастую напрашивается само собой, поэтому в данной главе более не будем разбирать примеры, в которых его можно использовать. Вместо этого рассмотрим особенности выражений в круглых скобках.

Надо сказать, что разработчики PHP сделали цикл `for` настолько универсальным, что в нем можно уместить целые программы. Начнем с самого простого и рассмотрим листинг 6.14.

Листинг 6.14 ▼ Бесконечный цикл

```
<?php
for (;;)
{
}
?>
```

Если запустить этот код, то в лучшем случае через некоторое время выведется сообщение об ошибке, в котором будет сказано, что программа выполняется очень долго. В действительности эта программа представляет собой бесконечный цикл. Первое, что бросается в глаза, – отсутствие самих выражений, что допустимо в PHP. Если отсутствие первого и третьего выражения можно интерпретировать, как «ничего не делать», то второе означает TRUE. Именно поэтому цикл выполняется бесконечно. Каким образом выйти из такого цикла, мы разберем немного позже, когда будем рассматривать безусловные операторы.

Следующая особенность выражений оператора `for` – это то, что они могут состоять из нескольких выражений (листинг 6.15).

Листинг 6.15 ▾ Особенности оператора for

```
<html>
<head>
   <title> Особенности оператора for </title>
</head>
<body>
<?php
for ($i=1, $j=9;$i <= 9, $j >= 1; $i++, $j-)
{
   echo $i+$j;
   echo "<br>";
}
?>
</body>
</html>
```

Эта программа выводит девять раз подряд число 10. Заметим, что подвыражения разделяются между собой запятыми. Пояснений, наверное, требует только второе выражение, так как в других случаях просто

перечисляются несколько действий. В логическом выражении если хотя бы одно из подвыражений равно FALSE, то произойдет выход из цикла, что, по сути, эквивалентно логической операции «и». На этом завершается рассмотрение особенностей выражений. Хочется добавить, что несмотря на такие универсальные возможности оператора `for`, не стоит ставить перед ним сверхзадач. Помните, что чем сложнее конструкция, тем труднее найти в ней ошибку.

Пожалуй, единственное, о чем мы еще не говорили, так это о *теле цикла*. Под этим понятием подразумевают действие, выполняемое при каждой итерации. В данном случае оно, как и у оператора `if`, может состоять из одной или нескольких команд. В первом случае можно не применять фигурные скобки (однако, это не рекомендуется), а во втором они необходимы.

Foreach

Оператор цикла `foreach` появился в четвертой версии PHP. Так как он предназначен для работы с массивами, то его подробный разбор приведен в главе 8.

While

Цикл `while` можно назвать упрощенным циклом `for`, так как он выполняет часть его функций. Однако синтаксис очень похож на оператор `if`:

```
while (выражение) действие; // цикл выполняется, пока
// выражение равно TRUE
```

Конструкция начинается со слова `while`, после которого следует выражение, имеющее такой же смысл, как и у оператора `if`, то есть пока оно равно TRUE, цикл будет выполняться. Тем самым оператор `while` эквивалентен оператору `for` с пустым первым и третьим выражением. Единственное отличие состоит в том, что логическое выражение оператора `while` не может состоять из подвыражений. Например, код, представленный в листинге 6.16, вызовет ошибку.

Листинг 6.16 ▾ Оператор while

```
<?php
while ($i <= 9, $j >= 2)     // ошибка!
{
   $i++;
$j-;
}
?>
```

Для того чтобы задать сразу несколько условий, применяйте логические операторы, что естественнее, чем использование подвыражений (листинг 6.17).

Листинг 6.17 ▾ Особенности оператора while

```
<?php
$i = 1;
$j = 10;
while ($i <= 9 && $j >= 2)
{
   $i++;
$j-;
}
?>
```

Эта программа уже не будет вызывать ошибку. Ход работы оператора `while` следующий. Сначала проверяется условие, если логическое выражение равно TRUE, то выполняется тело цикла, иначе осуществляется выход из него. Примеры использования оператора `while` будут еще не раз встречаться в этой книге.

Do...while

Конструкция do...while используется в тех случаях, когда выполнение тела цикла необходимо хотя бы один раз. При этом условие выхода проверяется не в начале, а в конце итерации. Синтаксис цикла do...while следующий:

```
do действие; while (выражение);     // Действие выполняется,
                                    // хотя бы один раз.
                                    // выход осуществляется по FALSE
```

Итак, конструкция начинается со слова do, за которым следует тело цикла. Затем записывается слово while с логическим выражением в круглых скобках. Приведем в качестве примера листинг 6.18.

Листинг 6.18 ▼ Оператор do...while

```
<html>
<head>
   <title> Особенности оператора for </title>
</head>
<body>
<?php
do
{
   echo "Ты меня видишь!";
}
while (0);
?>
</body>
</html>
```

Итак, сначала выведется сообщение, а затем будет проверяться условие, которое в нашем случае равно FALSE, что говорит о выходе из цикла.

Когда используют конструкцию do...while, обычно допускают две ошибки. Первая заключается в следующем: многим кажется, что слова do и while могут выступать в роли фигурных скобок, то есть выделять блок команд. Это совершенно не так! Приведем в качестве примера листинг 6.19.

Листинг 6.19 ▾ Особенности оператора do...while

```
<?php
do
   echo 1;
   echo 2;      // эта строчка вызовет ошибку
while (0);
?>
```

Данная программа вызовет ошибку. Нужно обязательно помещать блок команд в фигурные скобки.

Вторая ошибка заключается в том, что программисты забывают поставить точку с запятой после закрывающей круглой скобки, обрамляющей логическое выражение. Она не обязательна лишь в том случае, когда за ней следует закрывающий тег ?> (об этом речь шла в главе 3).

На этом заканчивается обзор циклов. Главное надо понять принцип их работы, так как особенности применения в значительном объеме будут рассмотрены в других главах.

Безусловные операторы

Безусловные операторы в основном предназначены для работы с циклами. В некоторых случаях они существенно упрощают алгоритм программы и делают код более понятным. Тем не менее их применение считается нежелательным, так как в блоках программ, содержащих безусловные операторы, наиболее часто встречаются ошибки. Используйте их с особой осторожностью.

Break

Применение оператора break не ограничивается конструкцией switch. Он очень часто применяются в циклах, причем имеющих поисковый характер. Например, мы хотим выяснить, есть ли книги в магазине объемом более 400 страниц, причем нас не интересует их количество. Предположим, что у нас имеется список этих книг с соответствующей информацией. Начиная с первой книги, мы будем сравнивать количество страниц с числом 400. Как только мы встретили книгу, удовлетворяющую поисковому критерию, надо вывести соответствующее сообщение и выйти из цикла. Именно эту функцию и выполняет оператор break. Вспомним случай с бесконечным циклом for. С помощью break мы свободно можем из него выйти (листинг 6.20).

Листинг 6.20 ▼ Оператор break

```
<?php
$i=1;
for (;;)
{
   echo $i;
   $i++;
   if ($i>10) break;
}
?>
```

В этом случае «бесконечный» цикл выполнится всего 10 раз, так как после выполнения условия $i > 10 следует оператор break.

На практике очень часто применяют вложенные циклы, то есть циклы, выполняющиеся внутри тела другого цикла (листинг 6.21).

Листинг 6.21 ▼ Параметр оператора break

```
<html>
<head>
   <title> Особенности оператора break </title>
```

```
</head>
<body>
<?php
for ($i=1;$i<=10;$i++)
{
   for ($j=1;$j<=10;$j++)
   {
         $sum++;
         if (($j + ($i-1)*10) == 55) break 2;
   }
}
echo $sum;
?>
</body>
</html>
```

Отличительной чертой break в PHP является его параметр, указывающий на цикл, из которого нужно выйти. По умолчанию он равен 1, что означает выход из текущего цикла. В нашем примере мы использовали параметр, который равен 2. В результате выход произошел на 55 итерации (это количество считает переменная $sum), так как мы прервали внешний цикл. Попробуйте убрать этот параметр и подумайте, почему получается именно такой результат (95).

Continue

Оператор continue так же, как и break, применяется в циклах. Его выполнение приводит к незамедлительному переходу к следующей итерации (листинг 6.22).

Листинг 6.22 ▾ Оператор continue

```
<html>
<head>
```

```
    <title> Оператор continue </title>
</head>
<body>
<?php
for ($x=-5;$x<=5;$x++)
{
    if (!$x) continue;
    $y = 1/$x;
    echo "Значение переменной y рано "."$y"."<br>";
}
?>
</body>
</html>
```

Здесь мы выводим значение переменной $y в зависимости от $x. В случае, когда переменная $x будет равна 0, в окне браузера выведется сообщение об ошибке, но выполнение программы не прекратится, что в определенных условиях не устраивает. Чтобы избежать такой ситуации, можно применить оператор `if` в сочетании с ключевым словом `else`, а можно написать просто `continue`, и проблема будет решена.

Exit

Оператор `exit` применяется, если возникла такая ситуация, что дальнейшее выполнение приложения не имеет смысла. Например, программа выводит новости из базы данных. В случае если подсоединиться к ней невозможно, то вывод данных осуществить нельзя, поэтому обычно выводят сообщение об ошибке и выполняют оператор `exit` (листинг 6.23).

Листинг 6.23 ▼ Оператор exit

```
<html>
<head>
```

```
    <title> Оператор exit </title>
</head>
<body>
<?php
if ($error == 1)
{
    echo "Произошла критическая ошибка!";
    exit;
}
?>
</body>
</html>
```

Require и include

Обычно крупное приложение не содержат в одном файле, а разбивают на несколько частей. Например, часто используемые константы, переменные и функции лучше хранить в отдельном файле и использовать его содержание по необходимости. Именно эту функцию исполняют операторы `require` и `include`.

Require

Синтаксис оператора `require` очень простой:

```
require "имя файла";
```

Результатом его выполнения будет вставка текста указанного файла при запуске программы (не при выполнении!). Этот файл может содержать как PHP-код, так и просто текст (листинг 6.24 и 6.25).

Листинг 6.24 ▾ Файл globals.php

```
<?php
$a = "Переменная a находится в файле globals.php";
?>
```

Листинг 6.25 ▼ Запускаемая программа

```
<?php
require "globals.php";
echo $a;
?>
```

Include

Оператор include выполняет такие же действия, как и require, но с одним лишь отличием – он вставляет текст файла во время выполнения программы (листинг 6.26).

Листинг 6.26 ▼ Работа оператора include

```
<?php
for($i = 1; $i <=5; $i++)
{
include "file".$i.".php";
}
?>
```

В данном примере вставляется содержимое пяти файлов. Заметьте, что при использовании оператора require возникла бы ошибка.

Заключение к главе

В этой главе мы познакомились с управляющими операторами PHP. На данный момент главное, что для себя нужно уяснить, – принцип работы и особенности вышеизложенных конструкций. Способы их применения мы разберем позже, когда будем рассматривать приложения, часто встречающиеся на практике.

Глава 7 Функции

В предыдущих главах много раз встречалось понятие *функция*. Пришло время подробно разобрать, что это такое. Даже если вы опытный программист, внимательно прочтите эту главу, так как функции в PHP имеют очень много особенностей. Итак, приступим.

Функция – это...

На самом деле функции – это предмет целой исторической эпохи в программировании. Когда-то давным-давно программы представляли собой машинный код, с которым работал непосредственно процессор. Но человек постоянно пытался научить машину «говорить» с ним на одном языке. В результате непонятные для нас нолики и единички стали превращаться в команды, выполняющие одно действие. Но на этом развитие языков программирования не закончилось, так как с увеличением сложности решаемых задач количество команд возрастало в геометрической прогрессии. Для решения этой проблемы программисты создали структуру, с помощью которой можно было выделить блок команд и обозначить их определенным именем. Ее назвали *процедурой*. Это нововведение позволило резко сократить код программ и сделало его более

понятным. Именно с этого момента начинается эпоха процедурного программирования. Дальнейшее развитие привело к тому, что у процедур появились входные параметры, которые назвали аргументами. И наконец, появились функции, отличающиеся от процедур тем, что они могли не просто выполнять определенные действия, но и возвращать значения.

Тем не менее в PHP нет понятия процедуры. Вне зависимости есть ли возвращаемое значение или его нет, мы имеем дело с функцией (function).

Как вы, наверное, уже догадались, в PHP есть множество встроенных функций, которые решают широкий круг стандартных задач. Например, вывод сообщений, сортировка массива и т.д. Но на практике даже такого большого количества функций бывает мало. Поэтому в PHP, как и во многих других языках программирования, имеется возможность создавать функции самостоятельно непосредственно в коде программы. Их часто называют пользовательскими функциями. Именно им мы уделим особое внимание в этой главе.

Определение функций

Для начала приведем пример создания пользовательской функции. Очень часто на практике требуется вывести сообщение об ошибке, поэтому логично было бы организовать это с помощью функции (листинг 7.1).

Листинг 7.1 ▼ Пример функции

```
<html>
<head>
   <title> Пример функции </title>
</head>
<body>
<?php
```

```
function error_msg ($err_str)
{
   echo "<b>Ошибка!<br>Причина: " . $err_str . "</b>";
}
error_msg("вы ввели отрицательное число");
?>
</body>
</html>
```

Итак, описание пользовательской функции начинается со слова function. Затем следует ее имя и в круглых скобках через запятую указываются входные параметры (аргументы). В данном случае имя функции – error_msg, а аргумент – $err_str. Далее в фигурных скобках следует тело функции, в котором указываются команды на исполнение.

Разберем поэтапно ход работы такой программы. При вызове функции error_msg() строка вы ввели отрицательное число записывается в переменную $err_srt и выводится в теле функции.

Заметьте, что описание функции мы поместили прежде, чем вызвали ее. На самом деле, начиная с четвертой версии PHP, описание может располагаться в любом месте программы.

Мы уже встречали среди встроенных функций те, которые возвращают значения. При создании пользовательских функций тоже имеется такая возможность (листинг 7.2).

Листинг 7.2 ▾ Функция возведения в квадрат

```
<html>
<head>
   <title> Функция возведения в квадрат </title>
</head>
<body>
<?php
function mnog($num)
```

```
{
return $num*$num;
}
echo mnog(2);
?>
</body>
</html>
```

Эта простая функция подсчитывает квадрат числа, который передается в качестве параметра. Возвращение результата происходит посредством оператора return, который находится в теле цикла. Все,что записывается после него, функция передает программе. Также оператор return можно использовать для завершения работы функции (листинг 7.3).

Листинг 7.3 ▼ Функция возведения числа в степень –1

```
<html>
<head>
   <title> Функция возведения числа в степень -1 </title>
</head>
<body>
<?php
function invert($num)
{
if ($num == 0) return;
echo 1/$num;
}
echo invert(0);
?>
</body>
</html>
```

В этом примере если мы передадим в качестве входного параметра 0, то программа не выдаст предупреждения об ошибке, так как завершиться еще до выполнения операции деления.

Негласные правила при определении функций

В этой книге уже не раз упоминалось о том, что существует ряд негласных правил оформления кода. Для функций рекомендуется писать имена в нижнем регистре и разделять слова символом подчеркивания. Пример:

```
set_var();
print_msg();
inc();
```

Обычно если функция что-то возвращает, то ее название начинают со слова `get_`. Пример:

```
get_cfg_var();
get_class();
get_browser();
```

Также часто используют приставки `set_`, `show_`, `print_` и другие, которые позволяют по названию функции определить, что она делает. В наших примерах мы не всегда будем придерживаться этих правил, так как они несут другую смысловую нагрузку.

Аргументы функций

Особый интерес представляют аргументы функции, так как здесь необходимо четко понимать, что вы передаете и каким образом. Например, вы решили написать функцию, которая не просто увеличивает значение переменной на единицу, но и выводит его на экран (листинг 7.4).

Листинг 7.4 ▼ Функция увеличения числа на единицу и вывода его на экран

```
<html>
<head>
```

```
<title>Функция увеличения числа на единицу и вывода его на экран</title>
</head>
<body>
<?php
function inc_print($num)
{
$num++;
echo $num;
}
$a = 1;
echo inc_print($a);
echo "<br>" . $a;
?>
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 7.1

Видно, что функция вывела совершенно верный результат, но оставила переменную $a без изменений. Происходит это потому, что в функцию передается значение переменной. Эквивалентом в этом случае

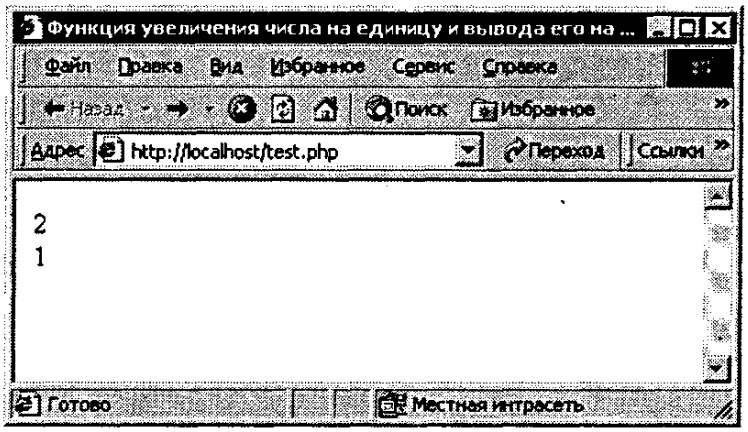

Рис. 7.1 ▼ Результат выполнения функции inc_print ()

была бы операция присваивания ($num = $a). Для того чтобы наша функция работала правильно, надо аргументы передавать по ссылке (листинг 7.5).

Листинг 7.5 ▾ Передача параметров по ссылке

```
<html>
<head>
   <title> Передача параметров по ссылке </title>
</head>
<body>
<?php
function inc_print(&$num)
{
$num++;
echo $num;
}
$a = 1;
echo inc_print($a);
echo "<br>" . $a;
?>
</body>
</html>
```

Результат выполнения данной программы смотрите на рис. 7.2.

На этот раз функция выполнила поставленную перед ней задачу. В программе мы добавили всего один уже знакомый нам символ & перед аргументом в определении функции. Напомним, что если мы имеем дело со ссылками, то изменение одной переменной влечет за собой изменение другой. Именно поэтому достигается нужный результат.

Стоит заметить, что при передаче аргументов по ссылке нельзя использовать литералы, константы и другие значения, которые не могут изменяться. Например, программа, представленная в листинге 7.6, выдаст сообщение об ошибке.

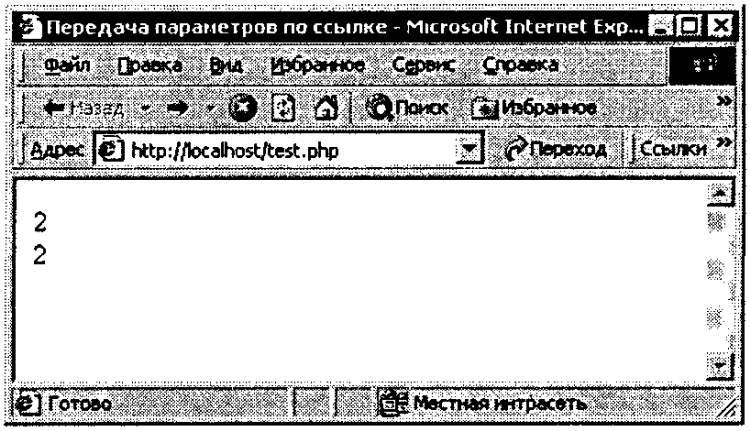

Рис. 7.2 ▼ Результат выполнения функции inc_print () с передачей параметра по ссылке

Листинг 7.6 ▼ Особенности передачи параметров по ссылке

```
<html>
<head>
   <title> Особенности передачи параметров по ссылке </title>
</head>
<body>
<?php
function inc_print(&$num)
{
$num++;
echo $num;
}
echo inc_print(123);   // вызовет ошибку
?>
</body>
</html>
```

Вы наверняка обратили внимание на встроенные функции с необязательными параметрами. Их также называют параметрами по умолчанию, что более точно характеризует эти аргументы. При определении

пользовательских функций они оформляются так, как показано в листинге 7.7.

Листинг 7.7 ▾ Функция с параметром по умолчанию

```
<html>
<head>
   <title> Функция с параметром по умолчанию </title>
</head>
<body>
<?php
function error_msg ($err_str, $flag_end = 0)
{
   echo "<b>Ошибка!<br>Причина: " . $err_str . "</b>";
   if ($flag_end == 1) exit;
}
error_msg("вы ввели отрицательное число");
echo "Продолжение программы";
?>
</body>
</html>
```

В этом случае аргументом по умолчанию является переменная `$flag_end`, которая определяет, выходить из программы при вызове функции или нет. Здесь мы вызываем функцию `error_msg()` без указания этого параметра. Поэтому значение переменной `$flag_end` принимается равным нулю по умолчанию, и мы не выходим из программы. Если значение передать явно, например `error_msg("вы ничего не ввели", 1)`, то выполнится оператор `exit`, и сообщение "Продолжение программы" выведено не будет.

Нужно сказать несколько слов о порядке расположения аргументов при определении функций. Размещайте параметры функции по умолчанию в конце списка аргументов, так как могут возникнуть некоторые проблемы при их вызове (листинг 7.8).

Листинг 7.8 ▼ Особенности передачи параметров по ссылке

```
<html>
<head>
   <title> Особенности передачи параметров по ссылке </title>
</head>
<body>
<?php
function error_msg ($flag_end = 0, $err_str)
{
   echo "<b>Ошибка!<br>Причина: " . $err_str . "</b>";
   if ($flag_end == 1) exit;
}
error_msg("вы ввели отрицательное число");
echo "Продолжение программы";
?>
</body>
</html>
```

В результате выполнения этой программы PHP сообщит, что пропущен второй параметр.

Область видимости переменных

При активной работе с пользовательскими функциями вы обязательно столкнетесь с проблемой области видимости переменных. Что это такое, лучше пояснить на примере листинга 7.9.

Листинг 7.9 ▼ Область видимости переменных

```
<html>
<head>
   <title> Область видимости переменных </title>
```

```
</head>
<body>
<?php
function inc()
{
$num++;
}
$num = 1;
inc();    // вызываем функцию
echo $num;       // выведет 1
?>
</body>
</html>
```

Результатом выполнения этой программы будет вывод числа 1. Другими словами, расположенные в теле функции и в основной программе переменные совершенно не связаны друг с другом, хотя имеют одинаковые имена. В этом случае говорят, что переменная, находящаяся внутри тела функции, имеет локальную область видимости, а та, которая располагается в основной программе, – глобальную. Локальные переменные объявляются внутри тела функции и недоступны извне. Глобальные переменные могут объявляться как в основной программе, так и в теле функции, но делается это с помощью специального оператора `global` (листинг 7.10).

Листинг 7.10 ▾ Глобальные переменные

```
<html>
<head>
   <title> Глобальные переменные </title>
</head>
<body>
<?php
```

```
function inc()
{
   global $num;
$num++;
}
$num = 1;
inc();          // вызываем функцию
echo $num;      // выведет 2
?>
</body>
</html>
```

В этом случае выводится число 2, так как переменная $num в теле функции имеет глобальную область видимости.

Время жизни переменных

Такое понятие, как время жизни переменных, тоже очень распространено в программировании. Например, время жизни глобальных переменных начинается с того момента, как их объявили, и заканчивается в двух случаях. Либо их уничтожили непосредственно в программе, например с помощью функции unset(), либо завершилась работа сценария (листинг 7.11).

Листинг 7.11 ▼ Удаление переменных

```
<html>
<head>
   <title> Удаление переменных </title>
</head>
<body>
<?php
```

```
$global_var = 1;
unset($global_var);
echo $global_var;
?>
</body>
</html>
```

В результате программа выводит пустую строку, так как функция `unset()` уничтожает переменную `$global_var`.

Иначе дело обстоит с локальными переменными, так как время их жизни напрямую зависит от продолжительности выполнения пользовательской функции. Однако встречаются такие ситуации, когда нужно сохранять значения локальных переменных. Для этого применяют специальный оператор `static` (листинг 7.12).

Листинг 7.12 ▾ Использование статических переменных

```
<html>
<head>
   <title> Использование статических переменных </title>
</head>
<body>
<?php
function print_msg($msg)
{
   static $num = 1;
   echo $num .") ". $msg . "<br>";
         $num++;
}
print_msg("Начало установки программы");
print_msg("Продолжение установки программы");
print_msg("Завершение установки программы");
```

```
?>
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 7.3.

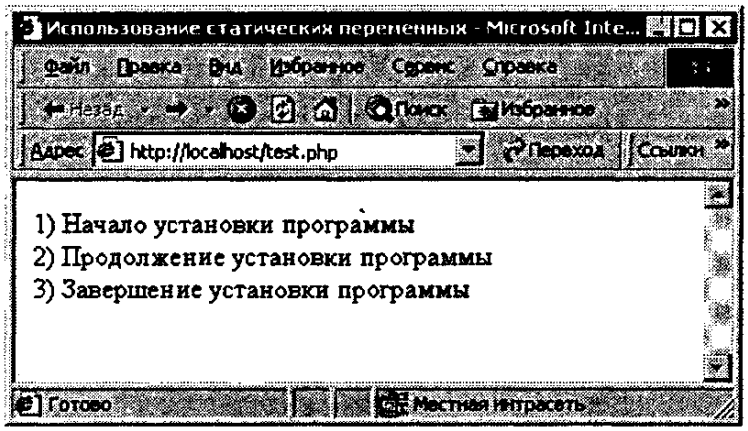

Рис. 7.3 ▼ Применение оператора static

Из примера видно, что переменная `$num` определяется один раз при первом вызове и не уничтожается после завершения работы функции, так как во втором вызове мы имеем дело со значением предыдущего вызова.

Рекурсия

При объяснении решения задач математики очень часто применяют словосочетание «по индукции», которое многих ставит в тупик, так как это понятие является не самым простым. В программировании тоже имеется своеобразная индукция, называемая рекурсией.

Например, попробуем написать функцию с помощью обычного цикла `for`, возвращающую факториал числа. Из курса математики известно, что он вычисляется следующим образом: `n!=1*2...*(n-1)*n`. Причем `0!=1`, а факториала отрицательных чисел не бывает (листинг 7.13).

Листинг 7.13 ▼ Функция нахождения факториала числа без рекурсии

```
<html>
<head>
```

```
    <title> Функция нахождения факториала числа без рекурсии </title>
</head>
<body>
<?php
function factorial($num)
{
    if ($num < 0)
    {
        return 0;
    }
    if ($num == 0)
    {
        return 1;
    }
    for ($i = 1, $sum = 1; $i <= $num; $i++)
    {
        $sum = $sum * $i;
    }
    return $sum;
}
echo factorial(3);     // выведет 6
?>
</body>
</html>
```

Итак, задача решена, как говорится, «в лоб». В случае передачи отрицательных значений функция будет возвращать 0. Это реализуется с помощью первого оператора `if`. Далее если передано число 0, то возвращаем единицу. И наконец, в цикле `for` вычисляем факториал при любых других значениях.

При решении задач с помощью рекурсии требуются рассуждения иного рода. Например, факториал числа можно вычислить следующим образом:

n! = 1, если n=0

n! = n*(n-1)!, если n>0

Заметьте, что второе утверждение содержит в себе знак факториала. Именно в этом и состоит основной смысл рекурсии. При решении определенной задачи мы используем то, что нам нужно найти. В программировании функцию называют рекурсивной, если в ее описании присутствует обращение саму на себя.

Итак, вернемся к рассуждению о факториале. При решении задач с помощью рекурсии всегда надо иметь два типа утверждений: базисное и рекурсивное. В нашем случае базисным утверждением является n! = 1, если n = 0. Как вы, наверное, уже догадались, работая с рекурсивным утверждением, мы должны перейти к базисному и получить результат. Теперь попробуем реализовать все это на практике (листинг 7.14).

Листинг 7.14 ▼ Функция нахождения факториала числа с рекурсией

```
<html>
<head>
   <title> Функция нахождения факториала числа с рекурсией </title>
</head>
<body>
<?php
function factorial($num)
{
   if ($num < 0)
   {
         return 0;
   }
   if ($num == 0)
```

```
    {
        return 1;
    }
    return $num*factorial($num-1); // рекурсивный вывод
}
echo factorial(3);                 // выведет 6
?>
</body>
</html>
```

В этом случае первые два условия аналогичны примеру, разобранному выше. Но теперь вместо цикла for мы записываем рекурсивное выражение, в котором функция вызывает сама себя. В нашей программе это происходит до тех пор, пока в качестве входного параметра не будет число 0, при котором функция возвратит 1. Причем это значение возвратится не в основную программу, а в тело предыдущей функции. Эта функция, в свою очередь, возвратит свое значение в тело следующей функции и т. д., пока нужное значение не возвратится в основную программу.

Динамический вызов функций

В PHP имеется возможность использовать вместо имени функции переменные (листинг 7.15).

Листинг 7.15 ▾ Динамический вызов функции

```
<html>
<head>
   <title> Динамический вызов функции </title>
</head>
<body>
<?php
```

```
function error_msg ($err_str)
{
   echo "<b>Ошибка!<br>Причина: $err_str</b>";
}
$start_function = "error_msg";
$start_function("вы ввели отрицательное число");
?>
</body>
</html>
```

Такой вызов функции называют динамическим. Эта программа выполнит такие же действия, как при вызове функции `error_msg()`. На первый взгляд эта особенность PHP может показаться, мягко говоря, излишней. Однако динамический вызов функции часто применяется в случаях, когда нужно выяснить какую функцию выполнять при определенных условиях (листинг 7.16).

Листинг 7.16 ▼ Использование динамического вызова функций

```
<html>
<head>
   <title> Использование динамического вызова функций </title>
</head>
<body>
<?php
function error_msg ($err_str)
{
   echo "<b>Error!<br>Text: $err_str</b>";
}
function warring_msg ($war_str)
{
```

```
    echo "<b>Warring!<br>Text: $war_str</b>";
}
function information_msg ($inf_str)
{
    echo "<b>Information!<br>Text: $inf_str</b>";
}
$type_msg = "err";
$msg = "вы ввели отрицательное число";
switch ($type_msg)
{
    case "err" : $start_function = "error_msg";
    break;
    case "war" : $start_function = "warring_msg";
    break;
    case "inf" : $start_function = "information_msg";
    break;
}
$start_function($msg);
?>
</body>
</html>
```

Заключение к главе

В конце этой главы хочется дать два совета по написанию пользовательских функций. Во-первых, не ставьте перед вашей функцией глобальных задач. Она должна быть маленькой, но полезной. Это простое правило позволит быстро устранить возникающую ошибку. Обычно

программисты говорят, что функция должна умещаться на экране монитора (приблизительно 20–30 строк). Поверьте, это очень удобно при написании программы.

Во-вторых, прежде чем начинать писать свои функции, убедитесь в том, что среди встроенных функций подобных вашей нет – встроенные обычно работают куда быстрее и надежнее, чем пользовательские.

Глава 8 Массивы

Очень часто в книгах о языках программирования тему массивов разбирают в отдельной главе, причем не в самой маленькой по объему. Такое положение обусловлено прежде всего тем, что массивы значительно облегчают жизнь программистам. Представьте себе, что приложение работает с данными, которые содержат в себе тысячи записей. Такое очень часто встречается на практике: например, работа с базой данных телефонных номеров. До этого момента мы разбирали переменные, которые позволяли хранить единственное значение. Естественно, с их помощью можно работать с любым количеством данных, но только сложно себе представить человека, способного разобраться в коде, содержащем тысячи переменных. В такой программе очень велика вероятность появления ошибки, а ее нахождение будет подобно поиску иголки в стоге сена. Именно по этим и другим причинам почти во всех языках программирования существуют переменные способные хранить в себе множество значений. Одной из них является массив.

Массив – это...

Мы уже говорили, что существует такой тип данных, как Array. Переменные этого типа называются массивами. По сравнению с другими

языками программирования PHP и в этом случае проявляет себя очень либерально. Другими словами, почти не существует жестких рамок при работе с массивами, что позволяет программисту проявить свою фантазию. Но помните, что любая вольность требует от вас большой концентрации внимания при написании программы.

Итак, массив представляет собой набор элементов, каждый из которых имеет *значение* и *ключ* (индекс). Значение - это данные, которые хранит элемент массива, а по ключу мы можем обратиться к нему. Для лучшего понимания о чем идет речь, представьте себе шкафчики для одежды, которые обычно находятся в спортивных раздевалках. Чтобы человек не забыл, где он оставил свою одежду, на дверцу наносят порядковый номер. Так вот, шкафчик - это элемент массива, его содержимое (одежда) - значение, а порядковый номер - ключ. Эта простая аналогия очень часто позволяет понять суть массивов.

Синтаксис массивов почти не отличается от других языков программирования. Пример:

```
$mas[2];
```

В этом примере мы обращаемся к элементу массива с именем `$mas`, который имеет ключ в виде числа 2. Надо отметить, что правила выбора имени массива такие же, как и у обычных переменных. Ключ записывается в квадратных скобках.

Обычно программисты называют массивом упорядоченную совокупность однотипных данных. Особенность массивов PHP заключается в том, что элементы одного массива могут быть разного типа данных. Возвращаясь к нашим шкафчикам, нужно сказать, что в них, например, можно хранить не только одежду, но и предметы спортивной утвари (мячи, ракетки и тому подобное).

Для человека привычнее все элементы массива нумеровать, то есть в качестве ключа использовать число. Но иногда встречаются такие ситуации, когда удобно вместо числа применять строку. Например, на каждом шкафу писать имя его владельца. В качестве ключа в PHP может быть значение либо Integer, либо String.

На этом особенности массивов PHP не исчерпываются. Изучив эту главу до конца, вы поймете, что для них действительно почти не существует жестких рамок.

Инициализация массивов

Итак, если вы более или менее поняли что такое массив, то можно приступать к его инициализации или созданию. Для этого PHP предлагает два способа: присвоение значений и с помощью функции `array()`.

Присвоение значений

Для того чтобы создать массив, можно просто присвоить значение его элементу подобно тому, как мы делали это с обычными переменными:

```
$closets[3] = "Майка";
```

В результате выполнения этой строки если массив `$closets` (шкафчики) еще не существует, то он будет создан. Его первый и пока единственный элемент будет содержать строку `Майка`. Ключом (который, кстати, может и отсутствовать) в данном случае будет число 3. Пример:

```
$closets[] = "Майка";
```

Здесь в квадратных скобках мы ничего не поставили. В этом случае если такого массива еще не существует, то по умолчанию ключом первого элемента будет число 0. Другими словами, эта запись будет эквивалентна следующему:

```
$closets[0] = "Майка";
```

Как мы уже говорили, в качестве ключа может быть строка. Тогда инициализация массива будет выглядеть следующим образом:

```
$closets["Петров"] = "Майка";
```

В этом случае мы создали массив с одним элементом, ключ которого строка `Петров`.

В PHP существует особенность, связанная с использованием пустого ключа (`[]`). Пример:

```
$closets[] = "Майка";
$closets[] = "Кроссовки";
$closets[] = "Шорты";
```

Как мы уже говорили, первой строчкой создается массив с одним элементом, ключ которого является число 0. При выполнении следующей строчки PHP добавит еще один элемент (добавление всегда происходи к концу массива), ключ которого на единицу больше, то есть 1. Другими словами все это эквивалентно следующему:

```
$closets[0] = "Майка";
$closets[1] = "Кроссовки";
$closets[2] = "Шорты";
```

Общий механизм подсчета очередного ключа при использовании пустых квадратных скобок нужно отдельно пояснить, так как у вас могло сложиться ложное мнение по этому поводу. Допустим массив уже существует, тогда ищется максимальный числовой ключ и именно к нему прибавляется единица. Пример:

```
$closets[10] = "Майка";
$closets[1] = "Кроссовки";
$closets["Петров"] = "Шорты";
$closets[] = "Брюки"; // эквивалентно $closets[11] = "Брюки";
```

В этом случае четвертым элементом будет строка брюки, а ключ будет равен 11, так как максимальное значение ключа до этого было число 10. Если числовые ключи отсутствуют, то по умолчанию он будет равен 0. Пример:

```
$closets["Петров"] = "Шорты";
$closets[] = "Брюки"; // эквивалентно $closets[0] = "Брюки";
```

Но и эти рассуждения, честно говоря, не всегда верны. Отклонение от общего правила может возникнуть при использовании функции unset(), которую мы рассмотрим позже.

Не трудно заметить, что употребление пустых квадратных скобок, мягко говоря, может запутать даже опытного программиста, поэтому не рекомендуется их применять.

Функция array()

Другим способом инициализации массивов является функция array() и выглядит это так:

```
$closets = array(1 => "Майка",2 => "Кроссовки",3 => "Шорты");
```

Функция array() в качестве входных параметров принимает пары ключ–значение и возвращает требуемый массив. Ключ и значение разделяются оператором =>. Пары разделяются запятыми. В приведенном примере создается массив, состоящий из трех элементов с ключами 1, 2 и 3 и значениями соответственно Майка, Кроссовки, Шорты.

По желанию ключи можно не указывать, тогда их значение будет определяться по умолчанию. Пример:

```
$closets = array("Майка", "Кроссовки", "Шорты");
```

В этом случае индексация будет начинаться с 0, поэтому рассматриваемая строчка будет эквивалентна этой:

```
$closets = array(0 => "Майка",1 => "Кроссовки",2 => "Шорты");
```

Надо отметить, что правила определения ключей по умолчанию здесь эквивалентны тем, которые мы рассматривали чуть выше. И не стоит забывать, что в качестве ключей могут быть и строки:

```
$closets = array("Петров"=>"Майка","Иванов"=>"Кроссовки","Сидоров"=>"Шорты");
```

Вывод массивов

После того как вы освоили инициализацию массивов, нужно научиться их использовать. Для начала попробуем просто вывести массив на экран (листинг 8.1).

Листинг 8.1 ▼ Вывод массива

```
<html>
<head>
   <title> вывод массива</title>
</head>
<body>
<?php
$closets = array(0 => "Майка",1 => "Кроссовки",2 => "Шорты");
echo $closets;
?>
</body>
</html>
```

В результате в окне браузера выведется слово Array! Эта, наверное, самая распространенная ошибка, которую допускают начинающие программисты, знакомясь с массивами. Конечно, ошибки в выполнении программы здесь нет, так как она просто выводит слово, которое указывает, что данная переменная является массивом. Просто в этом случае программист не видит разницы между переменной скалярного и смешанного типа (к последнему и относятся массивы).

Наверное, самым удобным способом, которым можно вывести массив, является функция print_r(). Эта функция может вывести значения массива с ключами (листинг 8.2).

Листинг 8.2 ▼ Вывод массива с помощью функции print_r()

```
<html>
<head>
   <title> вывод массива с помощью функции print_r()</title>
</head>
<body>
<pre>
```

```
<?php
$closets = array(0 => "Майка",1 => "Кроссовки",2 => "Шорты");
print_r ($closets);
?>
</pre>
</body>
</html>
```

На самом деле возможности функции print_r() куда шире, чем просто вывод массивов, но сейчас не имеет смысла приводить ее полное описание, так как материал по многим нужным для этого темам еще не пройден.

Обход массивов

Функция print_r() позволяет посмотреть на весь массив целиком, но не может выделить отдельные его части. Представьте себе задачу, когда требуется вывести из массива элементы, удовлетворяющие определенному условию. В этом случае функция print_r() не сможет нам помочь. Для решения подобных задач применяют операторы цикла. Но и здесь нас ожидают подводные камни. Сначала разберем самый простой случай, когда массив в качестве ключей имеет последовательный ряд чисел. Воспользуемся оператором цикла for, так как его запись является очень компактной и практичной. Единственное, что нам надо выяснить, – это количество элементов в массиве. На практике очень часто для решения данной проблемы применяют функцию count(). В качестве входного параметра для нее является массив, а возвращает она количество элементов в нем (листинг 8.3).

Листинг 8.3 ▾ Обход массива

```
<html>
<head>
   <title> Обход массива </title>
```

```
</head>
<body>
<?php
// инициализация массива
$closets = array(0 => "Майка",1 => "Кроссовки",2 => "Шорты");
$len_mass = count ($closets); // вычисление количества элементов
for ($i = 0; $i < $len_mass; $i++)
{
   echo $closets[$i] . "<br>"; // вывод элемента
}
?>
</body>
</html>
```

Итак, предположим, что ключ первого элемента в нашем массиве будет 0. Таким образом, мы проходим по всем элементам массива и выводим их. В этот цикл уже можно добавить определенное условие, предъявляемое к элементам массива.

У вас может возникнуть вопрос о присутствии переменной `$len_mass` в программе, так как функцию `count()` можно компактно уместить во второе выражение цикла `for`. Сразу хочется сказать, что делать этого не следует, потому что при этом возникают два негативных момента. Во-первых, функция `count()` будет выполняться при каждой итерации, что увеличивает время обработки, а во-вторых, количество элементов массива может поменяться в теле цикла.

К сожалению (а может и к счастью), рассматриваемый массив является лишь частным случаем. На практике обычно встречаются массивы с непоследовательной индексацией. Да и применение цикла `for` сегодня является неактуальным, так как есть специальная конструкция `foreach`, которая появилась только в четвертой версии PHP. В общем случае она выглядит так, как показано в листинге 8.4.

Листинг 8.4 ▾ Конструкция foreach

```
foreach ($massiv as $key => $value)
{
   // действия
}
```

В этой конструкции мы видим три переменные, среди которых $massiv - это просматриваемый массив. Переменные $key и $value (названия этих переменных вы выбираете сами) содержат соответственно ключ и значение. Итак, поясним ход работы этого оператора цикла на примере листинга 8.5.

Листинг 8.5 ▾ Обход массива с помощью конструкции foreach

```
<html>
<head>
   <title> вывод Обход массива с помощью конструкции forach </title>
</head>
<body>
<?php
$closets=array("Петров"=>"Майка","Иванов"=>"Кроссовки","Сидоров"=>"Шорты");
foreach ($closets as $key => $value)
{
   echo $key . " " . $value . "<br>"; // вывод элемента
}
?>
</body>
</html>
```

Итак, в первой строчке мы создаем массив из трех элементов. Заметьте, что индексация производится с помощью строк. Далее следует конструкция, которая начинается со слова foreach. Затем в круглых скобках

помещается рассматриваемый массив, после которого записывается специальное слово as. После него следует пара ключ–значение, разделяемая оператором =>. Итак, на первой итерации переменной $key присваивается ключ первого элемента, а переменной $value - его значение. При следующей итерации в переменные $key и $value запишутся соответственно ключ и значение следующего элемента. И так далее, пока массив не будет пройден полностью.

Естественно, у вас может возникнуть вопрос о способе реализации этого цикла, так как механизм перемещения по массиву остается скрытым от программиста. Дело в том, что любой массив помимо ключей и значений имеет внутренний указатель (pointer) или курсор, с помощью которого можно узнать, какой элемент мы сейчас рассматриваем. Этим указателем, естественно, можно управлять, то есть переносить его от одного элемента к другому. Для работы с указателем применяются функции list(), each(), count() и др. Приведем краткий обзор этих функций.

count()

Функция count() устанавливает указатель на первый элемент массива, который является для нее входным параметром. Возвращает она значение первого элемента массива (листинг 8.6).

Листинг 8.6 ▼ Работа функции count()

```
<html>
<head>
   <title> Работа функции count() </title>
</head>
<body>
<?php
$closets=
array("Петров"=>"Майка", "Иванов"=>"Кроссовки", "Сидоров"=>"Шорты");
echo reset($closets); // выводит слово "Майка"
?>
```

```
</body>
</html>
```

each()

Функция each() возвращает массив, который содержит ключ и значение элемента, на который указывает курсор. Причем ключ индексируется числом 0, а значение – 1. Затем функция смещает указатель на один элемент вправо (листинг 8.7).

Листинг 8.7 ▼ Работа функции each()

```
<html>
<head>
   <title> Работа функции each() </title>
</head>
<body>
<pre>
<?php
$closets=array("Петров"=>"Майка","Иванов"=>"Кроссовки","Сидоров"=>"Шорты");
print_r (each($closets)); // выводит массив
?>
</pre>
</body>
</html>
```

list()

На самом деле конструкция list() не является функцией, так как ее работа сильно отличается от работы функции. Поясним на примере листинга 8.8.

Листинг 8.8 ▼ Работа конструкции list()

```
<html>
<head>
```

```
    <title> Работа конструкции list() </title>
</head>
<body>
<?php
$closets = array(0 => "Майка",1 => "Кроссовки",2 => "Шорты");
list($thing_1, $thing_2, $thing_3) = $closets;
echo $thing_1; // выводит "Майка"
echo "<br>";
echo $thing_2; // выводит "Кроссовки"
echo "<br>";
echo $thing_3; // выводит "Шорты"
?>
</body>
</html>
```

Конструкция `list()` позволяет записать в переменные, которые находятся в круглых скобках, значения элементов в массиве. Но помните, что она работает с элементами проиндексированными числами, причем начиная строго с нуля, иначе переменные останутся пустыми.

До появления цикла `foreach` функции `list()`, `each()`, `count()` очень часто использовали вместе, чтобы просмотреть массив с непоследовательной индексацией (листинг 8.9).

Листинг 8.9 ▼ Обход массива посредством list(), each() и count()

```
<html>
<head>
    <title> Обход массива посредством list(), each() и count() </title>
</head>
<body>
<?php
$closets=
```

```
array("Петров"=>"Майка","Иванов"=>"Кроссовки","Сидоров"=>"Шорты");
reset($closets); // установка указателя на первый элемент
while (list($key, $value) = each($closets))
{
   echo $key . " " . $value . "<br>"; // вывод элемента
}
?>
</body>
</html>
```

Не сложно догадаться, что этот пример по своей сути представляет собой цикл foreach. Итак, сначала мы устанавливаем указатель на первый элемент массива с помощью функции count(). Затем в цикле начинаем последовательно выводить значения элементов вместе с их ключами. Обратите внимание на логическое выражение оператора while. Функция each() возвращает массив с данными для вывода, а конструкция list() записывает их переменные $key и $value. Происходит это до тех пор, пока функция each() не возвратит пустой массив, что эквивалентно FALSE.

Именно эту сложную конструкцию заменяет цикл foreach, который скрывает от программиста реальный ход работы программы и предоставляет удобный интерфейс. Обратите внимание, что оператор foreach перед началом обхода тоже устанавливает указатель на первый элемент и не возвращает его обратно после завершения цикла.

Стоит отметить, что в конструкции foreach по желанию значение ключа можно опустить. Тогда мы будем работать только со значениями элементов (листинг 8.10).

Листинг 8.10 ▾ Обход массива посредством конструкции foreach без ключа

```
<html>
<head>
   <title> Обход массива посредством конструкции foreach без ключа </title>
</head>
```

```
<body>
<?php
$closets=
array("Петров"=>"Майка","Иванов"=>"Кроссовки","Сидоров"=>"Шорты");
foreach ($closets as $value)
{
   echo $value . "<br>"; // вывод значения
}
?>
</body>
</html>
```

В этом случае программа выведет слова Майка, Кроссовки, Шорты.

Операторы массивов

В главе 5 мы рассматривали операторы применительно к переменным скалярного типа. Тем не менее для массивов тоже существует ряд операторов, которые можно, а порой и нужно, использовать.

Сложение массивов

Сложение (иногда эту операцию называют слиянием) массивов происходит посредством оператора суммы + (листинг 8.11).

Листинг 8.11 ▼ Сложение массивов

```
<html>
<head>
   <title> Сложение массивов </title>
</head>
<body>
<pre>
<?php
```

```
$closets_1=
array("Петров"=>"Ботинки","Иванов"=>"Футболка");
$closets_2=
array("Петров"=>"Майка","Иванов"=>"Кроссовки","Сидоров"=>"Шорты");
$closets_sum = $closets_1 + $closets_2;
print_r($closets_sum);
$closets_sum = $closets_2 + $closets_1;
print_r($closets_sum);
?>
</pre>
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 8.1.

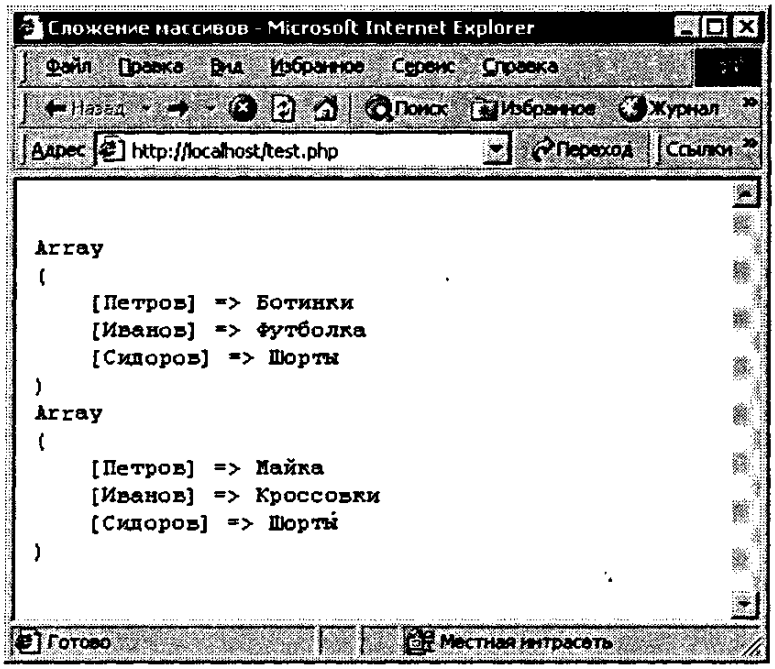

Рис. 8.1 ▾ Результат сложения массивов

В первом случае мы видим, что результирующий массив содержит в себе два элемента первого и один элемент второго массива. Чтобы

понять этот результат, нужно знать общий принцип сложения массивов. Результирующий массив состоит целиком из первого слагаемого, к которому добавляются элементы второго, имеющие отличные индексы. В связи с этим в первом случае отсутствуют элементы Майка и Кроссовки, а во втором - Ботинки и Футболка.

Сравнение массивов

Сравнение массивов производится с помощью привычных для нас операторов отношений. Наибольший интерес в этом плане вызывают операторы равенства (==) и эквивалентности (===).

Массивы считаются равными, в том случае, если каждый элемент одного массива имеет один равный ему во втором, и наоборот. Равенство элементов подразумевает соответственно совпадение ключа и значения. Порядок расположения элементов при этом не играет роли (листинг 8.12).

Листинг 8.12 ▼ Сравнение массивов

```
<html>
<head>
   <title> Сравнение массивов </title>
</head>
<body>
<?php
$closets_1=array("Ботинки","Футболка");
$closets_2=array("1" => "Футболка","0" => "Ботинки");
if ($closets_1 == $closets_2)
{
   echo "Массивы равны";
}
else
{
```

```
    echo "Массивы неравны";
}
?>
</body>
</html>
```

Заметьте, что индексы второго массива находятся в двойных кавычках, а значит имеют тип String. Но в данном случае сравниваются значения, а не типы, поэтому программа все равно выведет сообщение о равенстве массивов. То же самое относится к значениям элементов в массиве.

Иначе дело обстоит с оператором эквивалентности, который требует от своих операндов не только равенства значений, но и одинаковый порядок следования элементов в массиве (листинг 8.13).

Листинг 8.13 ▾ Эквивалентность массивов

```
<html>
<head>
   <title> Эквивалентность массивов </title>
</head>
<body>
<?php
$closets_1=
array("Ботинки","Футболка");
$closets_2=
array("1" => "Футболка","0" => "Ботинки");
if ($closets_1 === $closets_2)
{
   echo "Массивы эквивалентны";
}
else
```

```
{
   echo "Массивы неэквивалентны";
}
?>
</body>
</html>
```

В этом случае программа выведет, что массивы неэквивалентны.

Наряду с разобранными операторами существуют также операторы неравенства (!=) и неэквивалентности (!==), которые работают аналогично, но результат выполнения будет противоположный.

Модифицирование массивов

Очень часто на практике требуется работать не просто со значениями, которые хранятся в массиве, а с его элементами в целом, то есть добавлять и удалять их.

Добавление элементов массива

На самом деле в этой главе мы уже добавляли элементы к массиву и делали это с помощью оператора присваивания. Но более эффективный способ заключается в применении функции `array_push()`, которая добавляет один или несколько элементов к концу массива (листинг 8.14).

Листинг 8.14 ▼ Добавление элементов массива

```
<html>
<head>
   <title> Добавление элементов массива </title>
</head>
<body>
<pre>
<?php
```

```
$closets = array("Ботинки", "Футболка");
Array_push($closets, "Кеды", "Шорты");
print_r ($closets);
?>
</pre>
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 8.2.

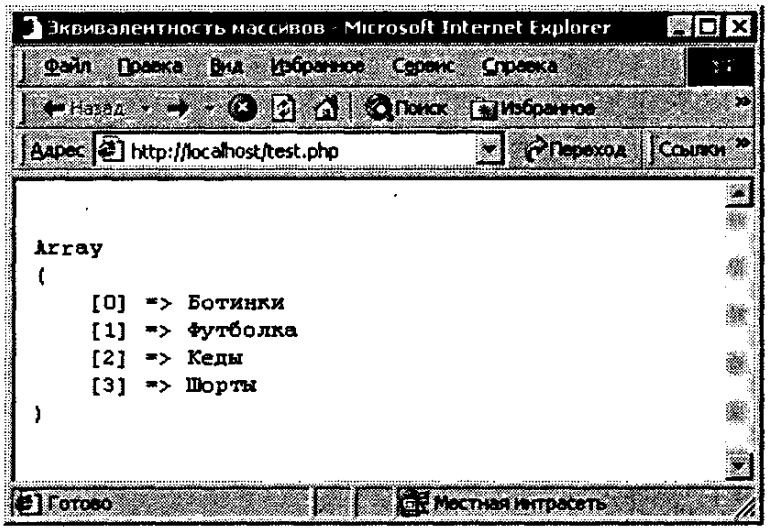

Рис. 8.2 ▾ Результат выполнения функции array_push()

Удаление элементов массива

Удаление элементов массива можно тоже производить различными способами. В этой главе мы разберем два из них. Во-первых, удалить элемент массива можно с помощью функции `unset()`. На самом деле она предназначена для уничтожения переменных, но на практике часто используется при работе с массивами (листинг 8.15).

Листинг 8.15 ▾ Удаление элементов массива

```
<html>
<head>
```

```
    <title> Удаление элементов массива </title>
</head>
<body>
<pre>
<?php
$closets = array("Ботинки", "Футболка", "Шорты");
unset($closets[1]);
print_r ($closets);
?>
</pre>
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 8.3.

В этом примере мы удалили элемент с индексом 1.

Подобно функции `array_push()` есть функция `array_pop()`, которая удаляет последний элемент массива (листинг 8.16).

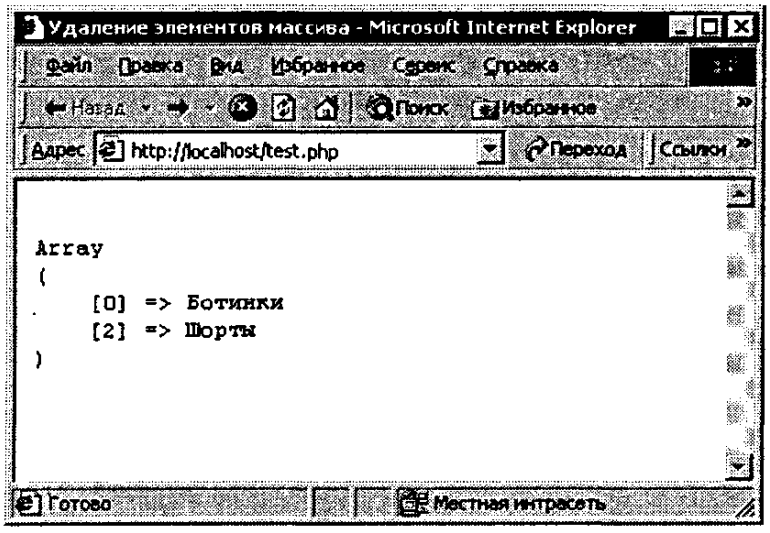

Рис. 8.3 ▼ Результат выполнения функции unset()

Листинг 8.16 ▾ Удаление последнего элемента массива

```
<html>
<head>
   <title> Удаление последнего элемента массива </title>
</head>
<body>
<pre>
<?php
$closets = array("Ботинки", "Футболка", "Шорты");
array_pop($closets);
print_r ($closets);
?>
</pre>
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 8.4.

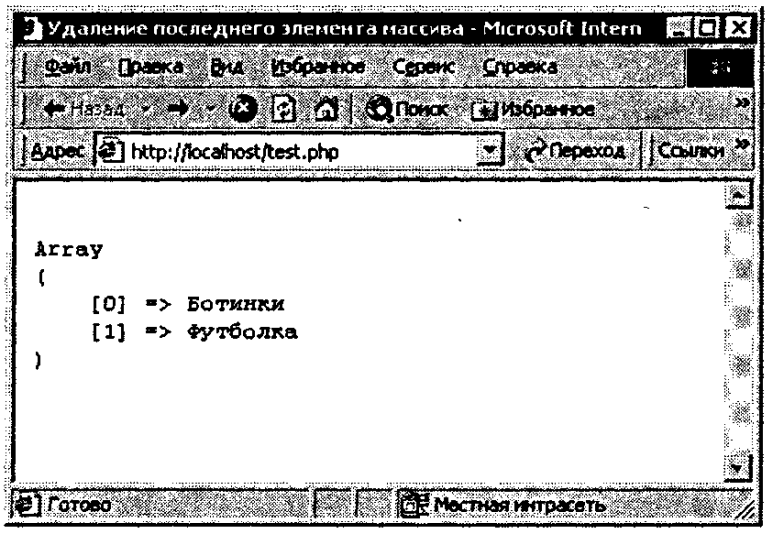

Рис. 8.4 ▾ Результат выполнения функции array_pop()

Сортировка массивов

Теперь вы можете удалять, добавлять и изменять элементы массива. Но на практике зачастую приходится работать не с отдельными элементами, а с массивом в целом. В данном случае речь пойдет о сортировке. Давно прошли те времена, когда нужно было самостоятельно придумывать наилучший алгоритм, например для вывода чисел в порядке возрастания. Сегодня для этого есть очень много полезных функций, с работой которых мы и познакомимся.

Самая простая и, наверное, самая распространенная функция сортировки – `sort()`. Она располагает элементы массива в алфавитном порядке (листинг 8.17).

Листинг 8.17 ▼ Сортировка массива

```
<html>
<head>
   <title> Сортировка массива </title>
</head>
<body>
<?php
$closets = array(0 => "Шорты",1 => "Майка",2 => "Кроссовки");
sort($closets);
foreach ($closets as $key => $value)
{
   echo $key . " " . $value . "<br>"; // вывод элемента
}
?>
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 8.5.

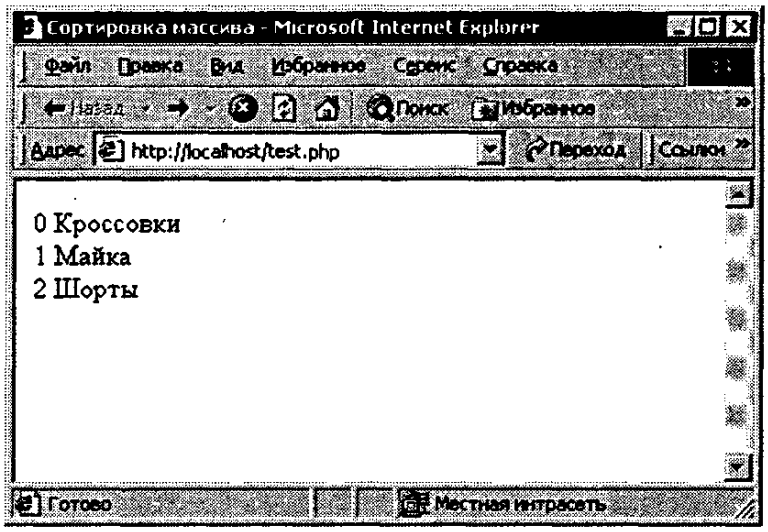

Рис. 8.5 ▾ Результат сортировки функцией sort()

В этом примере индексы остались неизменны, но это лишь частный случай. На самом деле, какие бы у нас не были ключи до сортировки, функция `sort()` сама индексирует элементы, начиная с нуля (листинг 8.18).

Листинг 8.18 ▾ Особенности сортировки массива с помощью функции sort()

```
<html>
<head>
   <title> Особенности сортировки массива с помощью функции sort()</title>
</head>
<body>
<?php
$closets = array(3 => "Шорты",4 => "Майка",1 => "Кроссовки");
sort($closets);
foreach ($closets as $key => $value)
{
   echo $key . " " . $value . "<br>"; // вывод элемента
}
?>
```

```
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 8.6.

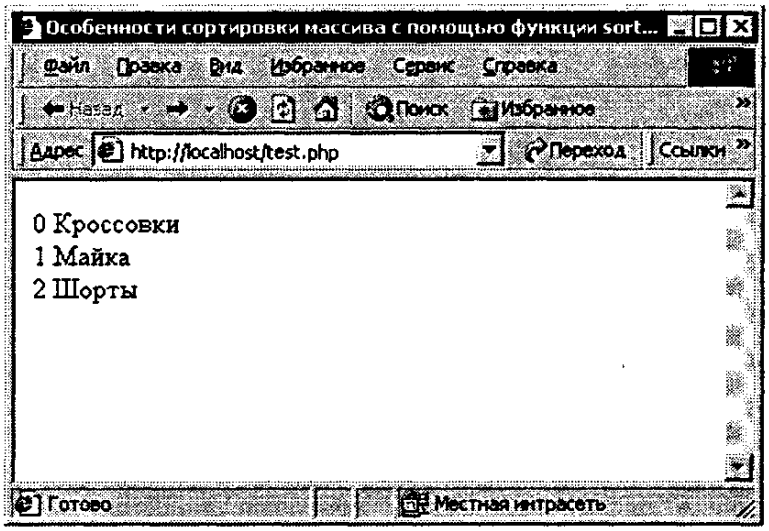

Рис. 8.6 ▼ Результат сортировки функцией sort()

Как видим, результат точно такой же, как и в первом случае. В связи с этим стоит быть осторожнее при сортировке массивов, где индексация имеет значение. Например, если вы в качестве ключей используете строки, то функция `sort()` все равно заменит их числами.

Еще одна особенность этой функции - необязательный параметр (флаг), который указывает на тип сортируемых значений (листинг 8.19).

Листинг 8.19 ▼ Использование необязательных параметров

```
<html>
<head>
   <title> Использование необязательных параметров </title>
</head>
<body>
<?php
$closets_1 = array(0 => 2,1 => 3,2 => "Кроссовки");
```

```
sort($closets_1, SORT_NUMERIC);
foreach ($closets_1 as $key => $value)
{
   echo $key . " " . $value . "<br>"; // вывод элемента
}
$closets_2 = array(0 => 2,1 => 3,2 => "Кроссовки");
sort($closets_2, SORT_STRING);
foreach ($closets_2 as $key => $value)
{
   echo $key . " " . $value . "<br>"; // вывод элемента
}
?>
</body>
</html>
```

В первом случае функция воспринимает все значения как числа, поэтому Кроссовки оказались на месте первого элемента (строка преобразовалась в число 0). Во втором случае значения сортируются как строки, поэтому на этот раз строка Кроссовки оказалась на месте последнего элемента. Существует так же третий флаг - SORT_REGULAR, который указывает на обычную сортировку.

Стоит отметить, что эти необязательные параметры функции sort() появились только в четвертой версии PHP.

Как мы уже говорили, функцию sort() неудобно применять к массивам, где нужно сохранять значение ключей. Для решения этой проблемы применяют функцию asort(), которая работает по аналогичному принципу, но не изменяет индексы элементов (листинг 8.20).

Листинг 8.20 ▾ Сортировка массива с помощью функции asort()

```
<html>
<head>
```

```
    <title> Сортировка массива с помощью функции asort() </title>
</head>
<body>
<?php
$closets = array(3 => "Шорты",4 => "Майка",1 => "Кроссовки");
asort($closets);
foreach ($closets as $key => $value)
{
    echo $key . " " . $value . "<br>"; // вывод элемента
}
?>
</body>
</html>
```

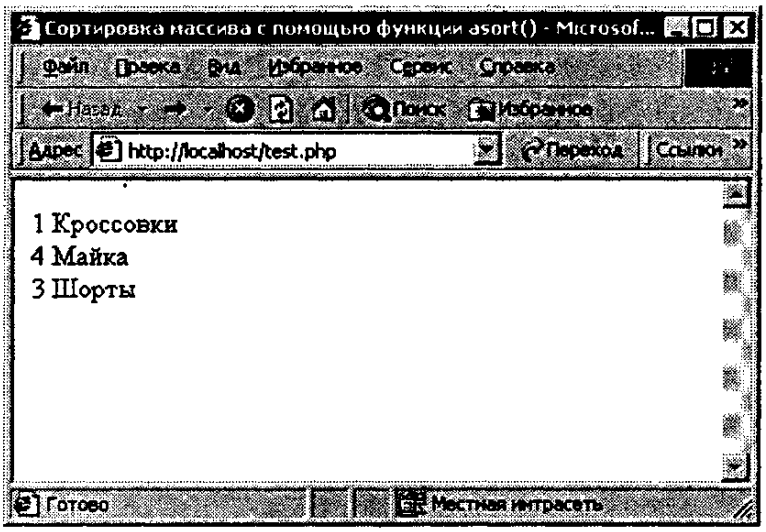

Рис. 8.7 ▼ Результат сортировки функцией asort()

Результат выполнения этой программы смотрите на рис. 8.7.

Как видим, результаты сортировок одинаковые за исключением того, что индекса элементов остались прежними. Необязательные параметры также присутствуют в этой функции и работают по такому же принципу.

Часто встречаются случаи, когда надо отсортировать элементы массива в обратном порядке. Для этого применяются функции `rsort()` и `arsort()`, которые в пояснениях не нуждаются, так как работают аналогично разобранным функциям.

Если есть сортировка элементов массива по значению, то логично было бы добавить сортировку по ключу. Именно эту задачу выполняет функция `ksort()` – листинг 8.21.

Листинг 8.21 ▾ Сортировка массива с помощью функции ksort()

```
<html>
<head>
   <title> Сортировка массива с помощью функции ksort() </title>
</head>
<body>
<?php
$closets=
array("Петров"=>"Майка","Иванов"=>"Кроссовки","Сидоров"=>"Шорты");
ksort($closets);
foreach ($closets as $key => $value)
{
```

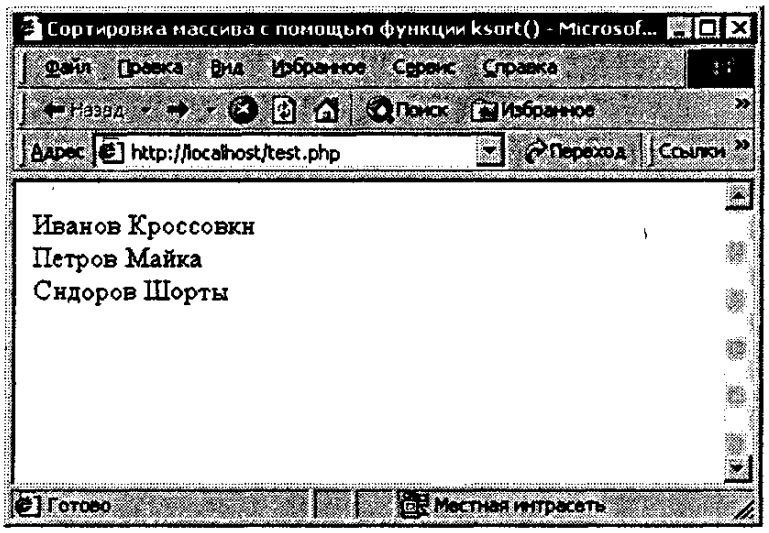

Рис. 8.8 ▾ Результат сортировки функцией ksort()

```
    echo $key . " " . $value . "<br>"; // вывод элемента
}
?>
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 8.8.

Наверное, стоит добавить, что существует функция `krsort()`, которая, как вы поняли, сортирует массив в обратном порядке.

Многомерные массивы

До этого момента в качестве элементов массива мы использовали только числа и строки. На самом деле элементом может быть все что угодно, даже другие массивы. Выглядит это следующим образом:

```
$lang = array("Петров"=> array ("Английский", "Испанский", "Немецкий"),
 "Иванов"=> array ("Французский", "Итальянский"),
 "Сидоров"=> array ("Немецкий"));
```

Это пример инициализации двумерного массива, где в качестве элементов выступают обычные (одномерные) массивы. Он содержит информацию о знании иностранных языков. На самом деле мы можем еще более усложнить структуру, если в качестве элементов внутреннего массива возьмем еще один массив, тогда это уже будет трехмерный массив. При желании так можно продолжать очень долго, но на практике обычно на этом останавливаются.

Обращение к элементам внутреннего массива происходит следующим образом:

```
$lang["Петров"][0];
```

В данном случае мы обращаемся к первому элементу массива, который, в свою очередь, является первым элементом массива `$lang`, то есть к строке `Английский`.

Обход многомерных массивов можно также производить с помощью конструкции foreach. При этом мы будем иметь дело с вложенными циклами (листинг 8.22).

Листинг 8.22 ▾ Вывод многомерных массивов

```
<html>
<head>
   <title> вывод многомерных массивов </title>
</head>
<body>
<?php
$lang = array("Петров"=> array ("Английский", "Испанский", "Немецкий"),
"Иванов"=> array ("Французский ", "Итальянский"),
"Сидоров"=> array ("Немецкий"));
foreach ($lang as $value)
{
```

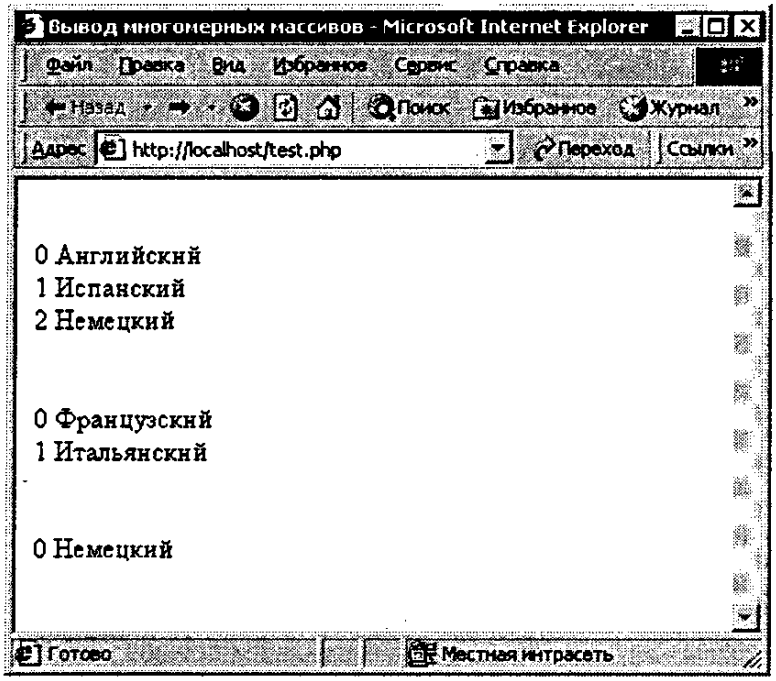

Рис. 8.9 ▾ Результат вывода многомерных массивов

```
    echo $key . "<br>";
    foreach ($value as $sub_key => $sub_value)
    {
        echo $sub_key . " " . $sub_value . "<br>"; // вывод элемента
    }
    echo "<br>";
}
?>
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 8.9.

Итак, во внешнем цикле переменная $key будет принимать значения индексов внешнего массива, то есть строки Петров, Иванов, Сидоров, а переменная $value - внутренние массивы. Переменные $sub_key и $sub_value работают уже непосредственно с ключами и значениями внутренних массивов.

Преобразование в массив

В главе 4 приводились примеры различных преобразований типов данных. Естественно, что тип Array не был разобран, так как тогда база знаний еще не позволяла этого сделать. Приведем пример преобразования известных нам типов данных в массив (листинг 8.23).

Листинг 8.23 ▼ Преобразование в массив

```
<html>
<head>
   <title> Преобразование в массив </title>
</head>
<body>
<pre>
```

```
<?php
$var_int = 123;
$var_double = 12.12;
$var_string = "World";
$var_boolean = TRUE;
print_r((array)$var_int);
echo "<br>";
print_r((array)$var_double);
echo "<br>";
print_r((array)$var_string);
echo "<br>";
print_r((array)$var_boolean);
?>
</pre>
</body>
</html>
```

Во всех случаях создается массив с единственным элементом, ключ которого равен нулю, а значение соответствует преобразуемой переменной. Относительно других типов данных можно сказать, что преобразование происходит точно так же.

Заключение к главе

В этой главе мы разобрали одну из самых важных структур в программировании - массив. Помните, что его основная задача работать с группой данных. Всегда старайтесь использовать цикл `foreach` для обхода массивов, так как на данный момент он является самой мощной и в то же время простой конструкцией в PHP. Здесь приведена лишь часть функций для работы с массивами, остальные вы сможете найти в справочниках по PHP.

Строки

Многие зададутся вопросом, почему о строках рассказывается в отдельной главе. Действительно, на первый взгляд может показаться, что их функциональность невелика, так как это всего лишь переменные скалярного типа данных String. Но в PHP эффективная работа со строками требует глубокого понимания этой темы.

Строка – это...

Как уже было сказано выше, в PHP существует скалярный тип данных String, переменные которого называют строками. По своей сути строка – это просто набор символов.

Определить строку можно следующими способами:

```
$stroka_1 = "Hello, World!"; // с помощью двойных кавычек
$stroka_2 = 'Hello, World!'; // с помощью одинарных кавычек
```

Итак, в данном случае строка создается с помощью оператора присваивания и двойных и одинарных кавычек.

Кроме того, строку можно создать с помощью heredoc-синтаксиса, описание которого здесь не приводится из-за специфического применения.

Несложно заметить, что определение строки почти ни чем не отличается от определения переменных других скалярных типов. Тем не менее проблемы начинают возникать, например, если вам требуется вывести одинарную или двойную кавычку (листинг 9.1).

Листинг 9.1 ▾ Неправильное определение строки

```
<html>
<head>
   <title> Неправильное определение строки </title>
</head>
<body>
<?php
$str = 'Он сказал: 'Привет!''; // эта строчка вызовет ошибку
echo $str;
?>
</body>
</html>
```

В данном случае в окне браузера выведется ошибка, которая заключается в том, что неправильно определена строка. Это происходит потому, что строкой считается набор символов между первой и ближайшей к ней кавычкой. Для решения этой проблемы применяют экранирование символов с помощью обратной косой черты (\). Тогда наша программа будет выглядеть так, как показано на примере листинга 9.2.

Листинг 9.2 ▾ Правильное определение строки

```
<html>
<head>
   <title> Правильное определение строки </title>
</head>
<body>
<?php
```

```
$str = "Он сказал: \"Привет!\''; // так правильно
echo $str;
?>
</body>
</html>
```

В данном случае программа выполнится без ошибок. Однако это всего лишь самый простой пример. На практике часто требуется, чтобы строки содержали в себе перевод строки, табуляцию и другие символы форматирования текста. Но такое возможно только при использовании двойных кавычек (листинг 9.3).

Листинг 9.3 ▼ Использование специальных символов

```
<html>
<head>
   <title> Использование специальных символов </title>
</head>
<body>
<pre>
<?php
// выводит: Одинарные кавычки \n
echo 'Одинарные кавычки \n';
echo "<br>";
// выводит: Двойные кавычки
echo "Двойные кавычки \n";
?>
</pre>
</body>
</html>
```

Несложно заметить, что в строках с одинарными кавычками специальные символы не распознаются, и это не единственное различие.

Обработка переменных внутри строк

Одна из самых главных особенностей строк, определенных с помощью двойных кавычек, – это возможность обрабатывать переменные внутри них (листинг 9.4).

Листинг 9.4 ▾ Обработка переменных внутри строк

```
<html>
<head>
   <title> Обработка переменных внутри строк </title>
</head>
<body>
<?php
$str = "Иван";
// выведет: Привет, Иван!
echo "Привет, $str!";
?>
</body>
</html>
```

В данном случае вместо подстроки `$str` подставляется значение переменной `$str`. Происходит это по следующей схеме. Подстрока будет считаться переменной, если она образует правильное имя переменной (смотрите главу 4). Например, в строке `Hello, $strs!` переменной будет считаться `$strs`, а не `$str` как в предыдущем примере. Если все-таки требуется распознать именно переменную `$str` в этой строке, то надо применить фигурные скобки (листинг 9.5).

Листинг 9.5 ▾ Особенности обработки переменных внутри строк

```
<html>
<head>
```

```
    <title> Особенности обработки переменных внутри строк </title>
</head>
<body>
<?php
$str = 'Ivan';
// выведет: Hello, Ivans!
echo "Hello, {$str}s!";
?>
</body>
</html>
```

В том случае если вам понадобится вывести знак доллара (например, для вывода имени переменной), можно использовать одинарные кавычки или экранировать его.

Листинг 9.6 ▼ Вывод знака доллара

```
<html>
<head>
    <title> вывод знака доллара </title>
</head>
<body>
<?php
$str = "Hello";
// выведет: Переменная имеет имя $str
echo 'Переменная имеет имя $str';
echo "<br>";
// выедет: Переменная имеет имя $str
echo 'Переменная имеет имя \$str';
?>
```

```
</body>
</html>
```

В обоих случаях переменная в строках не определяется.

Полезные функции

В PHP содержится огромное количество функций для работ со строками. Наверное, не имеет смысла описывать все эти функции, так как это обычно присуще различным справочникам. Рассмотрим только те из них, которые часто применяются на практике.

Вывод строк

В PHP имеется много способов вывода строк. Для начала разберем самый простой из них – конструкция языка echo.

Конструкция echo встретилась нам уже в главе 3, когда нужно было вывести сообщение Hello, World!. Вам может показаться странным отсутствие круглых скобок, обрамляющих строку. На самом деле echo можно использовать как с круглыми скобками, так и без них. Просто во всех примерах подчеркивается тот факт, что echo является конструкцией языка, а не функцией.

Еще одна особенность echo заключается в том, что с помощью нее можно выводить несколько сообщений (листинг 9.7).

Листинг 9.7 ▾ Вывод строк с помощью echo

```
<html>
<head>
   <title> вывод строк с помощью echo </title>
</head>
<body>
<?php
```

```
$name = "Ann";
// выводит "Hello, Ann!"
echo "Hello, ", $name,"!";
?>
</body>
</html>
```

Форматированный вывод строк

До этого мы выводили строки в окне браузера с помощью команды echo. Однако в PHP имеется еще несколько способов, например с помощью функции printf(). Функция printf() очень похожа (хотя бы по названию) на ту, которая существует в языке Си. Она выводит строку в определенном формате, который задает программист (листинг 9.8).

Листинг 9.8 ▼ Форматный вывод строки

```
<html>
<head>
   <title> Форматный вывод строки </title>
</head>
<body>
<?php
$str = "Число 8 в двоичном представлении: %b";
// выводит: Число 8 в двоичном представлении: 1000
printf($str, 8);
?>
</body>
</html>
```

Первым аргументом функции printf() является строка для вывода. Ее формат определяется с помощью сочетания специальных символов. В данном случае это символ %, который всегда ставится первым, и буква

b, определяющая двоичный формат вывода целого числа. Это число передается в качестве следующего входного параметра. Заметьте, что сначала оно преобразуется к целому числу, а затем выводится в двоичном представлении.

В строке имеется возможность вставить несколько различных комбинаций специальных символов, при этом их количество должно совпадать с числом параметров функции printf(), кроме первого (листинг 9.9).

Листинг 9.9 ▾ Вывод числа в двоичном и восьмеричном представлении

```
<html>
<head>
   <title> вывод числа в двоичном и восьмеричном представлении </title>
</head>
<body>
<?php
$str = "Двоичное и восьмеричное представление числа 12: %b и %o";
// выведет: Двоичное и восьмеричное представление числа 12: 1100 и 14
printf($str, 12, 12);
?>
</body>
</html>
```

Как вы, наверное, догадались, буква o указывает на вывод целого числа в восьмеричном представлении. Полный список специальных символов представлен в табл. 9.1.

Таблица 9.1 ▾ Специальные символы

Символ	Описание
b	Параметр преобразуется в целое и выводится в виде двоичного числа
c	Параметр преобразуется в целое и выводится в виде символа с соответствующим кодом ASCII
d	Параметр преобразуется в целое и выводится в виде десятичного числа со знаком

Таблица 9.1 ▼ Специальные символы (окончание)

Символ	Описание
u	Параметр преобразуется в целое и выводится в виде десятичного числа без знака
f	Параметр преобразуется в вещественное число и выводится в виде двоичного числа
o	Параметр преобразуется в целое и выводится в виде восьмеричного числа
s	Параметр преобразуется в строку
x	Параметр преобразуется в целое и выводится в виде шестнадцатеричного числа (в нижнем регистре)
X	Параметр преобразуется в целое и выводится в виде шестнадцатеричного числа (в верхнем регистре)

По желанию можно задать ширину поля для вывода значения параметра (листинг 9.10).

Листинг 9.10 ▼ Вывод строки в поле определенной ширины

```
<html>
<head>
   <title> вывод строки в поле определенной ширины </title>
</head>
<body>
<pre>
<?php
$str = "%10s";
printf($str, "Hello");
?>
</pre>
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 9.1.

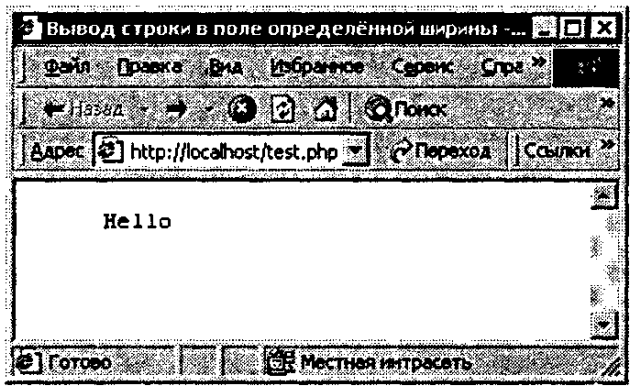

Рис. 9.1 ▼ Форматный вывод строк

Обратите внимание, что выравнивание текста происходит по правому краю поля. Таким способом очень удобно выводить табличные данные. Попробуете поэкспериментировать самостоятельно.

Если требуется выравнивание по левому краю, то нужно использовать такую же запись, но со знаком минус (-) – листинг 9.11.

Листинг 9.11 ▼ Выравнивание по левому краю

```
<html>
<head>
   <title> выравнивание по левому краю </title>
</head>
<body>
<pre>
<?php
$str = "%-10s";
printf($str, "Hello");
?>
</pre>
</body>
</html>
```

Помимо этого, с помощью функции printf() можно задавать точность выводимого числа, автоматически дополнять недостающие символы и многое другое. Более полную информацию об этой функции вы сможете найти в любом справочнике по PHP. Мы же перейдем к краткому рассмотрению других функций PHP для работы со строками.

Длина строки

На практике очень часто требуется узнать длину строки. Для этого в PHP имеется специальная функция strlen(), принимающая в качестве входных параметров строку и возвращающая ее длину виде целого числа (листинг 9.12).

Листинг 9.12 ▼ Определение длины строки

```
<html>
<head>
   <title> Определение длины строки </title>
</head>
<body>
<?php
// строка
$str = "Hello, World!";
// длина строки
$len = strlen($str);
// посимвольный вывод строки
for ($i=0;$i<=$len;$i++)
{
   echo $str[$i];
   echo "<br>";
}
?>
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 9.2.

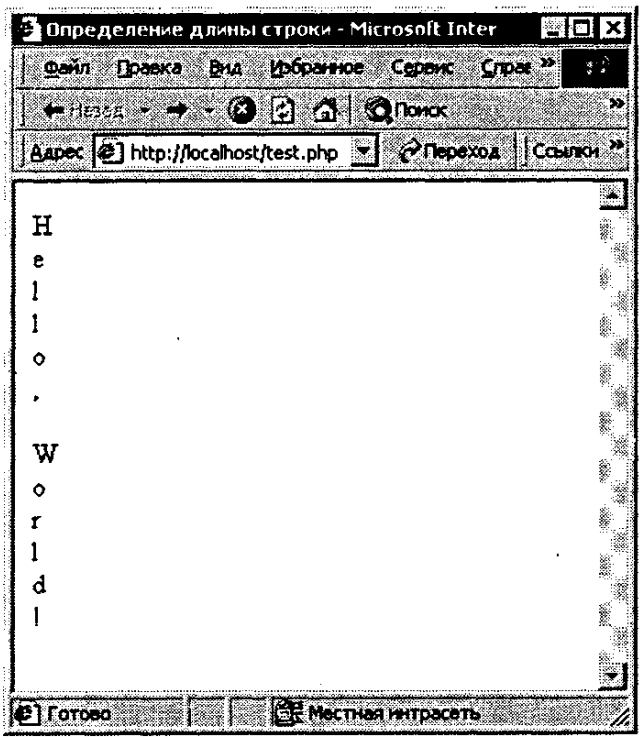

Рис. 9.2 ▼ Вертикальный вывод строки

Первое, что бросается в глаза, – это присутствие квадратных скобок, так как они использовались при работе с массивами. Здесь нет ничего странного, просто в PHP, как и во многих других языках программирования, строку можно воспринимать как массив символов, проиндексированный целыми числами, начиная с нуля.

Итак, вернемся к нашему примеру. Сначала мы инициализируем строку, затем с помощью функции `strlen()` получаем ее длину в виде целочисленного значения, которое запишем в переменную `$len`. Далее создаем простой цикл `for` на `$len` итераций. У многих начинающих программистов часто возникает соблазн не использовать дополнительную переменную для хранения длины строки. Действительно, функцию `strlen()` можно записать прямо в определении цикла `for`. Например: `for ($i=0;$i<=strlen($str);$i++){...}`. Тем не менее, категорически не рекомендуется так делать. Во-первых, этот код работает медленнее, чем с

использованием дополнительной переменной, так как приходится постоянно вызывать функцию strlen(). Во-вторых, длина строки может меняться в теле цикла, соответственно, поменяется количество итераций, что, вероятно, повлияет на ход программы.

В теле цикла нашей программы мы выводим один символ и выполняем перевод строки. Еще раз хочется обратить ваше внимание, что нумерация символов происходит с нуля.

Поиск подстроки в строке

Иногда бывает нужным выяснить, содержится ли подстрока в строке. Например, есть список фамилий, среди которых требуется найти Иванова. Осуществить это можно с помощью функции strstr(). Принцип ее работы рассмотрим на примере листинга 9.13.

Листинг 9.13 ▼ Поиск подстроки в строке

```
<html>
<head>
   <title> Поиск подстроки в строке </title>
</head>
<body>
<?php
// строка
$str = "Петров, Иванов, Сидоров";
// подстрока
$substr = "Иванов";
if (!strstr($str, $substr))
{
   echo "Фамилия не найдена";
}
else
```

```
{
   echo "Фамилия найдена";
}
?>
</body>
</html>
```

Первый параметр функции является строкой, где производится поиск подстроки, которая в свою очередь передается вторым параметром. В нашем примере строкой является переменная `$str`, а подстрокой – `$substr`. Если результат поиска отрицательный (Иванов отсутствует в списке фамилий), то функция возвращает значение FALSE. Если совпадение имеется, то функция возвращает часть строки, начинающуюся с найденной подстроки.

Чистка строк

Когда пользователь вводит данные, очень часто он ставит лишние пробелы. Для удаления пробельных символов из начала и конца строки имеется функция `trim()` – листинг 9.14.

Листинг 9.14 ▾ Удаление пробельных символов

```
<html>
<head>
   <title> Удаление пробельных символов </title>
</head>
<body>
<?php
$str = " Иванов Иван Иванович ";
$new_str = trim($str);
// выводит "Иванов Иван Иванович"
echo $new_str;
?>
```

```
</body>
</html>
```

В данном примере функция `trim()` возвращает строку с удаленными пробелами.

Надо заметить, что разобранные полезные функции являются всего лишь «каплей в море». Описание всех функций для работы со строками вы сможете найти в любом справочнике по PHP.

Заключение к главе

В этой главе мы рассматривали строки. Теперь вы знаете, как можно определить строку и как обрабатываются переменные и специальные символы внутри строки.

Работа с HTML-формами

Мы уже говорили, что PHP позволяет работать со многими структурами будь то электронная почта, файлы, базы данных и т.д. Однако рассмотрение этих и других возможностей PHP мы начнем с более простого примера – HTML-формы. Наверное, многие из читателей встречали HTML-формы, посещая Web-сайты сети Internet. Это может быть заполнение анкет с данными о пользователе, интерфейс гостевой книги и др. В данной главе мы рассмотрим схему передачи данных из HTML-формы в PHP-сценарий, структуры хранения этих данных, а также несколько полезных примеров использования HTML-форм.

HTML-форма – это...

HTML-форма – это средство языка HTML для ввода и передачи данных. Хотя они напрямую не относятся к PHP, для лучшего понимания материала этой главы рассмотрим основные принципы синтаксиса их создания. Любая форма начинается с тега `<form>` и заканчивается `</form>`. Внутри открывающего тега могут указываться атрибуты, которые определяют параметры HTML-формы. Каждый из них имеет название и оп-

ределенное значение. Нас будут интересовать атрибуты action и method. Внутри HTML-формы (между тегами <form> и </form>) обычно помещают поля для ввода текста с клавиатуры, списки, кнопки и т.д. Их подробное описание можно найти в любой литературе, посвященной языку HTML.

Вернумся к рассмотрению тега <form>. Атрибут этого тега action указывает на файл, в котором будут обрабатываться данные из HTML-формы (листинг 10.1).

Листинг 10.1 ▼ HTML-форма

```
<html>
<head>
   <title> HTML-форма </title>
</head>
<body>
<form action="test.php">
<input type="Text" name="text">
<input type="Submit" value="Go!"
</form>
</body>
</html>
```

В этом примере создается HTML-форма, которая содержит поле для ввода текста и кнопку для отправки данных. Если пользователь нажмет на кнопку из этой HTML-формы, то запустится файл с названием test.php, куда определенным способом передадутся данные из HTML-формы. Обратите внимание, что файл test.php и файл, содержащий HTML-форму, должны находиться в одной папке.

Передача данных HTML-формы

Рассмотрим атрибут method тега <form>. Он указывает, каким способом передавать данные из формы. Всего существует два метода: GET и POST.

Если использовать метод GET, то данные передаются посредством добавления их в конец строки запроса. Например, создайте файл test.html со следующим содержанием (листинг 10.2).

Листинг 10.2 ▾ Передача данных методом GET

```
<html>
<head>
   <title> Передача данных методом GET </title>
</head>
<body>
<form action="test.php" method="get">
<input type="Text" name="text">
<input type="Submit" value="Go!"
</form>
</body>
</html>
```

Затем нам понадобится файл с именем test.php, который будет запускаться при нажатии кнопки **Go!**. Его содержание пока нас не интересует, поэтому оставьте его пустым. Запустите файл test.html в браузере,

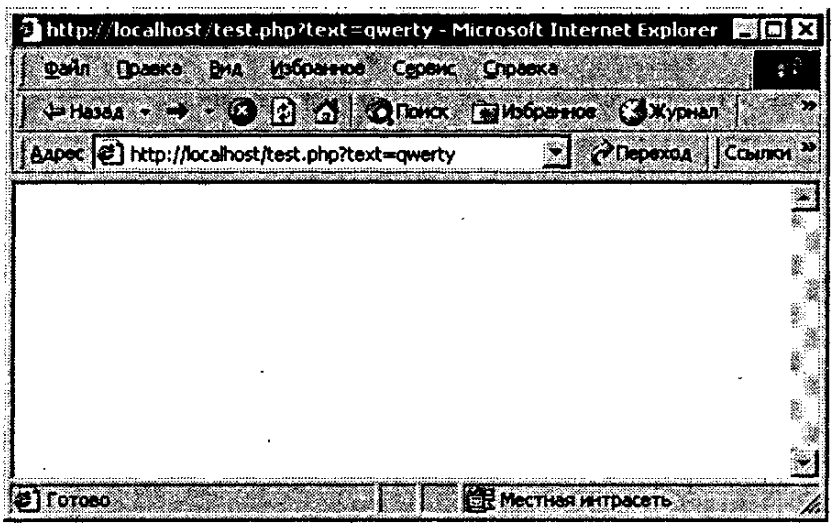

Рис. 10.1 ▾ Передача данных методом GET

введите в поле текст (например, «qwerty») и нажмите кнопку **Go!**. Результат смотрите на рис. 10.1.

Обратите внимание на адресную строку браузера. После знака вопроса следует выражение `text=qwerty`. Другими словами, пользователь может видеть, какие данные передаются серверу. Метод GET используется по умолчанию.

Если применить метод POST, то результат будет иным (см. рис. 10.2).

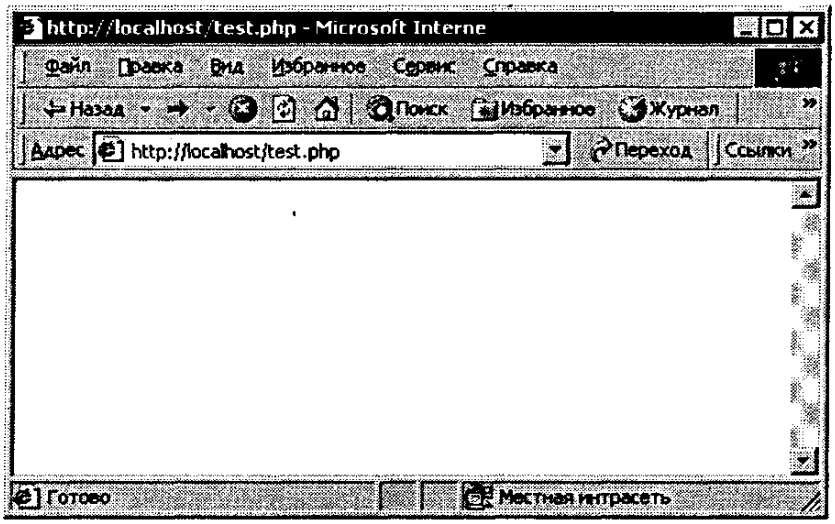

Рис. 10.2 ▼ Передача данных методом POST

В этом случае передаваемые данные скрыты от пользователя. В принципе особой разницы использования того или иного метода передачи данных нет. Единственное, что стоит отметить, это ограничение на объем передаваемой информации при методе GET.

Получение данных

Вплоть до версии PHP 4.2.0 при получении данных из HTML-формы не возникало никаких проблем, так как все было просто и естественно. Например, сделаем содержание файла test.php таким, как показано на примере листинга 10.3.

Листинг 10.3 ▾ Получение данных

```
<html>
<head>
   <title> Получение данных </title>
</head>
<body>
<?php
echo $text;
?>
</body>
</html>
```

Если бы мы работали с версией PHP 4.2.0, то результат нажатия кнопки **Go!** был бы таким, как на рис. 10.3.

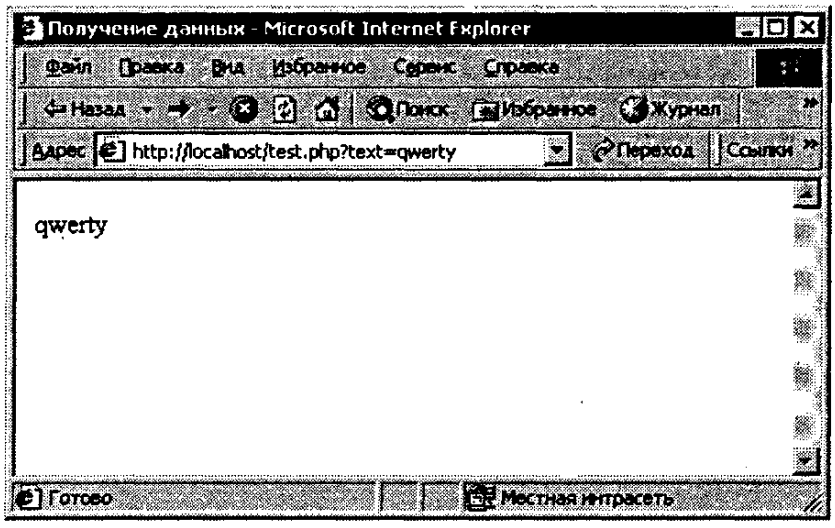

Рис. 10.3 ▾ Получение данных

Обратите внимание, что текст поля с названием text стал значением переменной `$text`. Если говорить более грамотно, то при запуске test.php автоматически инициализируются переменные с таким же именем, как у элементов формы. Однако если вы попробуете проделать

тоже самое в версии более поздней, то результат будет приблизительно такой, как на рис. 10.4.

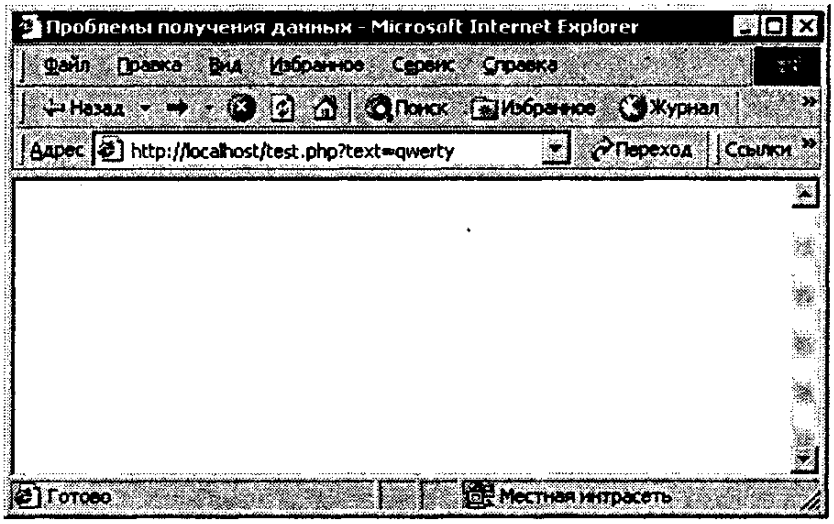

Рис. 10.4 ▼ Проблемы получения данных

Как видите, в этом случае текст из поля HTML-формы не передается. Все дело в одной установке конфигурационного файла php.ini, название которой register_globals. До версии PHP 4.2.0 включительно ее значение по умолчанию было on. При такой установке в самом начале выполнения PHP-скрипта, запущенного из формы, инициализировались глобальные переменные по описанному выше принципу. Однако разработчики в целях безопасности решили изменить значение по умолчанию на off – теперь передаваемые данные нужно доставать другим путем, немного сложнее прежнего.

Суперглобальные массивы $_GET и $_POST

Доступ к данным из HTML-формы при register_globals = off может осуществляться посредством суперглобальных массивов $_GET и $_POST. Использование того или иного массива зависит от метода передачи данных. Например, рассмотрим HTML-форму листинга 10.1. В этом случае по умолчанию метод передачи данных выбирается GET. Программа,

которая будет обрабатывать введенную пользователем информацию, приведена в листинге 10.4.

Листинг 10.4 ▾ Передача данных

```
<html>
<head>
   <title> Пердача данных </title>
</head>
<body>
<?php
echo $_GET['text']; // выводит значение поля text
?>
</body>
</html>
```

Итак, если выбирается метод передачи данных GET, то создается элемент массива `$_GET`, ключ которого имеет название `text`. При использовании POST данные заносятся в массив `$_POST`.

Заключение к главе

В этой главе мы познакомились с методами передачи данных – GET и POST. Так же рассмотрели принципы работы при различных установках конфигурационного файла php.ini. Обязательно заранее узнайте настройки Web-сервера, где вы предполагаете размещать ваши PHP-сценарии.

Глава 11 Работа с файлами и каталогами

Когда есть вопрос, на который надо ответить в сжатые сроки, когда не можешь найти последнюю песню любимой группы, когда требуется срочно отправить информацию, первая мысль, наверное, будет о сети Internet. Действительно, такое огромное количество данных, пожалуй, не найдешь ни в одной библиотеке мира. И все это благодаря возможности постоянно обмениваться информацией.

В такой ситуации организация хранения данных всегда была и будет одной из основных задач не только программирования, но и других областей. В сети Internet для этого обычно используют файлы и базы данных.

Несмотря на то что базы данных являются более удобными хранилищами данных, файлы до сих пор не утратили своей актуальности. Тем более что при малых количествах информации (счетчик посещений, гостевая книга) лучше использовать именно файлы, так как применение баз данных в этом случае будет похоже на стрельбу из пушки по воробьям.

В этой главе вы познакомитесь с основными возможностями PHP при работе с фалами и каталогами. Знания в этой области помогут вам на первом этапе в разработке многих PHP-приложений.

Работа с файлами

Файл – это...

Файл – это именованная область на носителе информации. Обычно файлы делятся на текстовые и бинарные. Текстовые файлы состоят из строк, которые заканчиваются переводом строки. Бинарные соответственно имеют иную структуру.

Открытие файлов

Чтобы начать работу с файлом, его нужно открыть. Для этого обычно применяют функцию `fopen()` – листинг 11.1.

Листинг 11.1 ▼ Открытие файла для чтения

```
<?php
// открытие файла для чтения
$file_pionter = fopen ("info.txt", "r");
?>
```

В качестве входных параметров функции `fopen()` выступает строка с именем файла и специальный признак, по которому определяется режим открытия файла (об этом чуть позже). В данном случае если файла не существует, то выведется сообщение об ошибке. Функция возвращает дескриптор файла, который имеет значение типа source. В случае неудачной попытки открытия файла она возвратит FALSE.

Вернемся ко второму параметру, который, как мы говорили, указывает режим открытия файла. В приведенном примере `r` означает то, что файл открывается только для чтения, то есть мы не сможем его изменять. Описания основных режимов открытия файлов смотрите в табл. 11.1.

Таблица 11.1 ▼ Режимы открытия файлов

Режим	Описание
r	Файл открывается только для чтения. При попытке открыть несуществующий файл выводится сообщение об ошибке. Если файл открылся удачно, то указатель текущей позиции устанавливается в начало

Таблица 11.1 ▼ Режимы открытия файлов

Режим	Описание
r+	Файл открывается одновременно для чтения и записи. Указатель текущей позиции устанавливается в начало файла. При этом запись в файл будет происходить поверх уже существующих данных, поэтому будьте осторожны при работе в этом режиме
w	Файл открывается для записи. При этом все данные, которые были в нем, уничтожаются. Если файла с таким именем не существует, то он создается
w+	Действия аналогичные режиму w, но файл открывается для чтения и для записи
a	Файл открывается для записи. При этом его указатель устанавливается в конец файла. Если файла не существует, то выводится сообщение об ошибке
a+	Файл открывается для чтения и записи. При этом его указатель устанавливается в конец файла. Если файла не существует, то он создается

Рассмотрим более подробно параметр, указывающий имя файла. В приведенном примере подразумевается, что файл info.txt, находится в одной папке с файлом PHP-сценария. Если файл имеет иное расположение, то нужно задать соответствующие папки (листинг 11.2).

Листинг 11.2 ▼ Открытие файла для чтения и записи

```
<?php
// путь указывается относительно корневой папки Web-сервера
// в этом случае это C:\Home_server\Apache2\htdocs
$file_pionter = fopen ("files/data/info.txt", "r+");
?>
```

Также вы можете задать абсолютный путь к файлу, хотя это не рекомендуется (листинг 11.3).

Листинг 11.3 ▼ Открытие файла для чтения и записи с указанием абсолютного пути

```
<?php
// задаем абсолютный путь
$file_pionter = fopen ("C:\\Home_server\\files\\info.txt", "r+");
?>
```

Заметьте, что для корректного задания пути символ \ должен удваиваться.

Также функция `fopen()` позволяет открывать файлы, находящиеся на других серверах. Происходит это посредством протоколов HTTP и FTP (листинг 11.4).

Листинг 11.4 ▼ Открытие файла для чтения посредством протоколов HTTP и FTP

```
<?php
// соединение с http сервером
$http_fp = fopen ("http://www.php.net/ "r");
// соединение с ftp сервером
$ftp_fp = fopen ("ftp://user:password@example.ru/info.txt", "w");
?>
```

При работе через HTTP и FTP протоколы нужно учитывать несколько особенностей. Например, если используется HTTP-соединение, то файл можно открыть только для чтения, а при FTP – нельзя одновременно открыть файл для чтения и записи.

Закрытие файлов

Если работа с файлом завершена, то его стоит закрыть. Эта операция осуществляется с помощью функции `fclose()`, которая в качестве входного параметра принимает дескриптор файла (листинг 11.5).

Листинг 11.5 ▼ Закрытие файла

```
<html>
<head>
   <title> Закрытие файла </title>
```

```
</head>
<body>
<?php
// открытия файла info.txt только для чтения
$file_pionter = fopen ("info.txt", "r");
// закрытие файла
if (fclose($file_pionter))
{
   echo "Закрытие файла произошло удачно";
}
else
{
   echo "Ошибка закрытия файла";
}
?>
</body>
</html>
```

Функция `fclose()` возвращает булевское значение, которое равно TRUE при удачном закрытии файла и FALSE в противном случае.

Чтение и запись файлов

Чтение из файлов в PHP можно осуществить различными способами. Например, с помощью функции `fread()`. Она принимает дескриптор файла – число, которое определяет длину строки из файла и возвращает эту строку (листинг 11.6).

Листинг 11.6 ▼ Вывод строки из файла

```
<html>
<head>
   <title> Вывод строки </title>
```

```
</head>
<body>
<?php
// открытие файла
$file_pionter = fopen ("info.txt", "r") or die ("Ошибка открытия файла");
// чтение файла пяти символов
$txt = fread($file_pionter, 5) or die ("Ошибка чтения файла");
// закрытие файла
fclose($file_pionter) or die ("Ошибка закрытия файла");
// вывод строки
echo $txt;
?>
</body>
</html>
```

Обратите внимание на конструкцию `or die()`. Функция `die()` является эквивалентом безусловного оператора `exit`, который разбирался в главе 6. Так как приоритет оператора `or` ниже чем `=`, то сначала будет выполняться операция присваивания, а затем, при условии, что левый операнд равен FALSE, – функция `die()`, которая позволит выйти из программы. Заметьте, что без использования конструкции `or die()` выводится сообщение об ошибке, но программа продолжает выполняться дальше.

Вернемся к рассмотрению функции `fread()`. В данном примере из файла info.txt читаются первые пять символов, которые записываются в виде строке в переменную `$txt`. Если при чтении достигнут конец файла, функция прекращает свое выполнение и возвращает строку, содержащую ровно столько символов, сколько прочиталось (несмотря на то, что длина строки может быть меньше запрашиваемой).

В PHP имеется похожая по выполнению функция `fgets()`, которая читает символы из файлов в количестве равном заданному минус единица

и возвращает их. Чтение прекращается не только при достижении конца файла, но и если встречается символ перевода строки \n (листинг 11.7).

Листинг 11.7 ▼ Работа функции fgets()

```
<html>
<head>
   <title> Работа функции fgets() </title>
</head>
<body>
<?php
// открытие файла
$file_pionter = fopen ("info.txt", "r") or die ("Ошибка открытия файла");
// чтение файла четырех символов
$txt = fgets($file_pionter, 5) or die ("Ошибка чтения файла");
// закрытие файла
fclose($file_pionter) or die ("Ошибка закрытия файла");
// вывод строки
echo $txt;
?>
</body>
</html>
```

Эту функцию очень удобно применять в том случае, когда нужно просмотреть файл по строкам.

Для записи строк в файл используются функции fwrite() и fputs(). По сути, они выполняют одно и тоже, отличаясь разве что названиями. В качестве входных параметров выступают дескриптор файла, который нужно открыть в режиме записи, и строка для записи в файл. Также имеется еще один необязательный аргумент в виде целочисленного значения, которое указывает на количество символов строки, записываемых в файл. Если его опустить, то она запишется полностью. Аналогично

другим разобранным функциям fwrite() и fputs() возвращают TRUE при удачной записи и FALSE в противном случае (листинг 11.8).

Листинг 11.8 ▾ Запись строки в файл

```
<html>
<head>
   <title> Запись строки в файл </title>
</head>
<body>
<?php
// открытие файла на запись
$file_pionter = fopen ("info.txt", "w") or die ("Ошибка открытия файла");
// запись строки в файл
fwrite($file_pionter, "Hello, World!\n") or die ("Ошибка записи файла");
// закрытие файла
fclose($file_pionter) or die ("Ошибка закрытия файла");
?>
</body>
</html>
```

Копирование, удаление и переименование файлов

При пользовании компьютером вы копировали и удаляли файлы, а также изменяли их названия. Подобные манипуляции часто востребованы в PHP-приложениях.

Для копирования файлов применяются функция copy(), работа которой предельно проста. Вам нужно задать две строки: имя файла и имя файла копии (листинг 11.9).

Листинг 11.9 ▾ Копирование файла

```
<?php
// копирование файла
```

```
copy ("info.txt", "files/log.txt") or die("Ошибка копирования файла");
?>
```

В данном случае файл info.txt копируется в паку files и переименовывается в log.txt. Если файл с таким именем уже существует, то он перезаписывается. Когда копирование по какой-то причине не возможно, функция возвращает FALSE, иначе – TRUE.

Для удаления файла используется функция `unlink()`, пример использования которой представлен в листинге 11.10.

Листинг 11.10 ▼ Удаление файла

```
<?php
// удаление файла
unlink ("info.txt") or die("Ошибка удаления файла");
?>
```

Несложно понять, что функция принимает имя файла и возвращает логическое значение по принципу, который встречался уже не раз в этой главе. После выполнения этой программы файла info.txt в корзине не будет, так как он удаляется безвозвратно.

Переименование файла (в PHP оно эквивалентно перемещению) осуществляется с помощью функции `rename()`. Нужно ввести первоначальное и новое имя файла (листинг 11.11).

Листинг 11.11 ▼ Переименование файла

```
<?php
// переименование файла
rename("files/log.txt","info.txt")or die("Ошибка переименования файла");
?>
```

Стоит отметить, что, в отличии от функции `copy()`, в случае если файл info.txt уже существует, то переименования не произойдет и функция возвратит FALSE.

Получение информации о файлах

Во всех примерах, рассмотренных выше, мы почти ничего не знали о файле, кроме его названия. Однако иногда полезно получить информацию о его размере, о дате последнего изменения и вообще существует ли файл с указанным именем. Итак, рассмотрим все по порядку.

Для начала нужно узнать существует ли файл. Для этого используется функция `file_exists()`. Она возвращает TRUE, если файл существует, в противном случае – FALSE (листинг 11.12).

Листинг 11.12 ▾ Проверка существования файла

```
<html>
<head>
   <title> Проверка существования файла </title>
</head>
<body>
<?php
// проверка существования файла
if (file_exists("info.txt"))
{
   echo "Файл существует";
}
else
{
   echo "Файл не существует";
}
?>
</body>
</html>
```

Однако вы можете обмануться, так как любая папка может иметь имя info.txt. Поэтому дополнительно нужно использовать функцию `if_file()` – листинг 11.13.

Листинг 11.13 ▼ Проверка на файл

```
<html>
<head>
   <title> Проверка на файл </title>
</head>
<body>
<?php
if (is_file("info.txt"))
{
   echo "Это файл";
}
else
{
   echo "Это не файл";
}
?>
</body>
</html>
```

Иногда полезно бывает знать, можно ли читать или изменять файл. Например, создадим файл с именем file.txt в корневом каталоге нашего Web-сервера. Среди его атрибутов поставим флажок **Только для чтения** – в этом случае мы не сможем открыть его в режиме записи. Если мы не знаем атрибуты файла, то проверку на запись и чтение можно осуществить с помощью функций `is_writable()` и `is_readable()` соответственно (листинг 11.14).

Листинг 11.14 ▾ Проверка на запись и чтение

```
<html>
<head>
   <title> Проверка на запись и чтение </title>
</head>
<body>
<?php
// проверка на запись
if (is_writable("info.txt"))
{
   echo "Файл возможно записать";
}
else
{
   echo "Файл невозможно записать";
}
// проверка на чтение
if (is_readable("info.txt"))
{
   echo "Файл возможно читать";
}
else
{
   echo "Файл невозможно читать";
}
```

```
?>

</body>

</html>
```

В этом примере функция `is_writable()` возвратит FALSE, а `is_readable()` – TRUE.

Файловый указатель

При работе с файлами нужно обязательно учитывать положение его указателя. Для этого в PHP существует несколько функций. Разберем некоторые из них.

Для установки указателя в начало файла используется функция `rewind()`, которая принимает файловый дескриптор и возвращает булевское значение (TRUE – удачное выполнение, иначе – FALSE).

Если требуется перенести указатель на конкретное место в файле, стоит использовать функцию `fseek()`. В качестве входных параметров выступают дескриптор файла и целочисленное значение, определяющее число байт или символов от указанной точки. Функция задается еще одним необязательным параметром:

- SEEK_SET (или 0) – от начала файла (по умолчанию);
- SEEK_CUR (или 1) – от текущей позиции;
- SEEK_END (или 2) – от конца файла.

Стоит обратить внимание, что в случае неудачи функция возвращает –1 (в отличие от многих других функций), при успешном завершении – 0.

Чтобы узнать текущую позицию файлового указателя, нужно использовать функцию `ftell()`. Ей передается дескриптор файла, а она возвращает целочисленное значение, которое определяет положение указателя относительно начала файла.

На практике часто используется функция `feof()`, с помощью которой определяется положение указателя на конце файла.

Работа с каталогами

Каталог – это...

Каталог – это, по сути, тот же файл, который хранит в себе информацию о подкаталогах и файлах. Обычно выделяют следующие понятия: текущий и родительский каталог. Текущим называют каталог, в котором мы находимся в данный момент, а родительский – это тот, в котором находится текущий.

Открытие и закрытие каталогов

Работа с каталогами так же, как и с файлами, обычно начинается с их открытия. Для этого используют функцию `opendir()`. Она принимает путь к каталогу в виде строки и возвращает его дескриптор (листинг 11.15).

Листинг 11.15 ▾ Открытие каталога

```
<?php
// открытие каталога
$dir_pointer = opendir("/files/data");
?>
```

При указании пути возможно использование строк . и . . . Одна точка обозначает текущий каталог, а две точки – родительский. Например, если текущий каталог Apache2/htdocs, то для открытия Apache2 нужно указать . . .

После работы с каталогом, как и после работы с файлами, его нужно закрыть (однако если вы этого не сделаете, ничего страшного не случится) с помощью функции `closedir()`. Она принимает дескриптор каталога и ничего не возвращает(листинг 11.16).

Листинг 11.16 ▾ Открытие и закрытие каталога

```
<?php
// открытие каталога
```

```
$dir_pointer = opendir("/files/data");
// закрытие каталога
closedir($dir_pointer);
?>
```

Чтение каталогов

После того как мы получили дескриптор, можно использовать его для просмотра содержимого каталога или, по-другому, его чтения (листинг 11.17).

Листинг 11.17 ▼ Чтение каталога

```
<html>
<head>
   <title> Чтение каталога </title>
</head>
<body>
<?php
// открытие каталога
$dir_pointer = opendir(".");
// чтение каталога
while (($res = readdir($dir_pointer))!==FALSE)
{
   // вывод имен файлов и папок
echo $res."<br>";
}
// закрытие каталога
closedir($dir_pointer);
?>
</body>
</html>
```

В этом примере используется функция readdir(). Она принимает дескриптор каталога, возвращает строку с именем файла (включая расширение) или подкаталога. При последующих вызовах мы будем получать другие имена до тех пор, пока все элементы каталога не будут перебраны. В этом случае функция возвратит значение FALSE. Порядок вывода будет зависеть от файловой системы.

Обратите внимание, что в цикле while мы использовали оператор !== (операнды не эквивалентны), а не != (операнды не равны). Делается это по следующей причине. Допустим, что среди имен файлов и подкаталогов имеется такое: 0. В этом случае при использовании оператора != выполнение цикла прекратилось бы раньше времени.

Однако разработчики PHP посчитали, что данный способ чтения каталогов является громоздким. Поэтому в пятой версии появляется новая функция scandir(). Она принимает дескриптор каталога и возвращает массив, элементы которого содержат имена файлов и подкаталогов (листинг 11.18).

Листинг 11.18 ▼ Вывод содержимого каталога

```
<html>
<head>
   <title> Вывод содержимого каталога </title>
</head>
<body>
<pre>
<?php
// родительский каталог
$dir = "..";
$arr = scandir($dir);
// вывод массива
print_r($arr);
?>
```

```
</pre>
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 11.1.

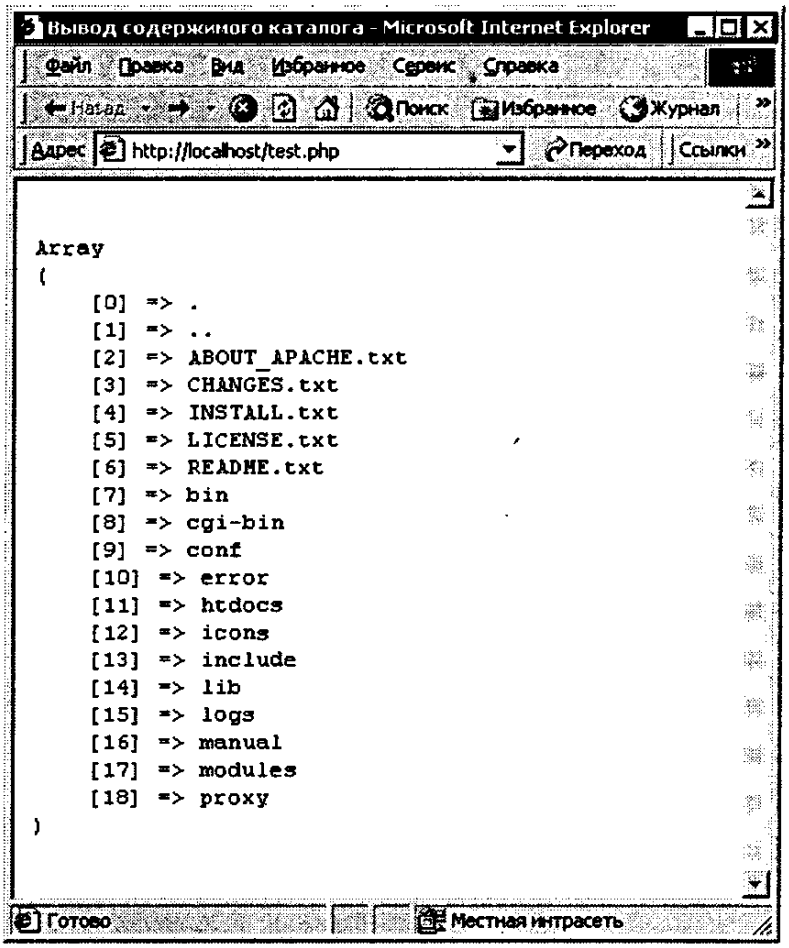

Рис. 11.1 ▼ Работа функции scandir()

Сортировка массива производится в алфавитном порядке по возрастанию. Однако имеется возможность это изменить с помощью необязательного параметра. Если его указать равным единице, то сортировка массива будет производиться в алфавитном порядке по убыванию (листинг 11.19).

Листинг 11.19 ▼ Вывод содержимого каталога в алфавитном порядке по убыванию

```
<html>
<head>
```

```
    <title> Вывод содержимого каталога в алфавитном порядке по убыванию </title>
</head>
<body>
<pre>
<?php
// родительский каталог
$dir = "..";
$arr = scandir($dir, 1);
// вывод массива
print_r($arr);
?>
</pre>
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 11.2.

Создание и удаление каталогов

Язык позволяет создавать и удалять не только файлы, но и каталоги. Для этого используют функции `mkdir()` и `rmdir()` соответственно (листинг 11.20).

Листинг 11.20 ▾ Создание и удаление какатола

```
<?php
// создание каталога
mkdir("dir_1");
mkdir("dir_2");
// удаление каталога
rmdir("dir_1");
?>
```

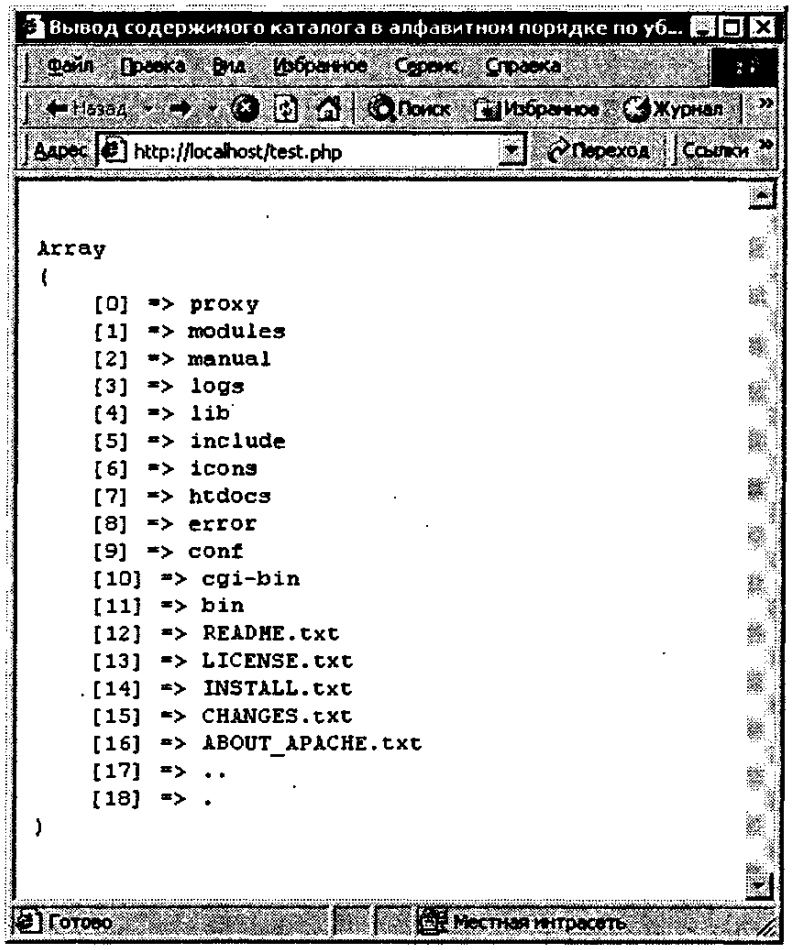

Рис. 11.2 ▼ Работа функции scandir() с указанием необязательного параметра

Стоит обратить внимание, что функция `rmdir()` может удалять только пустые каталоги, иначе выведется сообщение об ошибке, и функция возвратит FALSE.

Заключение к главе

В этой главе мы разобрали основные возможности PHP при работе с файлами и каталогами.

Глава 12 Работа с базами данных

Издавна люди искали надежное средство для хранения информации. Сначала это были наскальные рисунки, затем появилась письменность. Люди стали записывать мысли с помощью букв и знаков препинания на листах бумаги. С появлением компьютера ситуация резко изменилась: теперь огромное количество данных может храниться на жестком диске в виде текстовых файлов. Но и это не удовлетворило потребности человека. Поэтому ученые разработали специальную структуру хранения информации, называемую базой данных. Сегодня базы данных применяются почти во всех сферах человеческой жизни, поэтому разработчики PHP не могли не включить функции для работы с ними.

Базы данных – это...

База данных – это хранилище данных. Конечно, существует еще множество развернутых определений этого понятия, но здесь они не внесут большей ясности.

Так как базы данных представляют собой только данные, нужно было разработать программное обеспечение, способное работать с ними. Таким образом, появились системы управления базами данных (СУБД).

Как мы уже отмечали в главе 1, одной из наиболее популярных СУБД является MySQL. Надо заметить, что PHP работает с базами данных именно посредством СУБД.

Базы данных бывают нескольких видов, которые отличаются друг от друга структурой хранения данных. Сегодня реляционная модель баз данных является самой популярной среди всех прочих. Поэтому мы будем рассматривать именно ее.

Любая реляционная база данных состоит из *таблиц*, которые представляют собой именованные двумерные матрицы. В таблице выделяют поля (столбцы) и записи (строки). Пример таблицы смотрите на рис. 12.1.

id	name	author	num_pages
1	Самоучитель PHP	Иваиов Петр Сергеевич	222
2	Самоучитель Perl	Мелешко Виталий Николаевич	333
3	Справочник по математике	Дитятева Аниа Викторовна	415
4	Справочиик по физике	Доброхотова Татьяна Петровна	312

Рис. 12.1 ▼ Пример таблицы

В данном случае таблица хранит информацию о книгах, о чем говорит ее название (book). С помощью полей мы задаем характеристики книг: id (идентификационный номер), name (название), author (автор), num_pages (количество страниц). Смысл записи состоит в выделении характеристик отдельной книги. Обратите внимание на поле id, которое представляет собой идентификационный номер книги. Другими словами, несколько книг не могут одновременно иметь одинаковые значения этой характеристики. Такие поля называют ключами. Подробно с этими и другими понятиями мы разберемся позже на примерах в этой главе.

Обычно работа с базами данных осуществляется посредством запросов языка SQL (Structured Query Language). С помощью них можно добавлять и удалять таблицы, вносить новые данные и изменять старые, выбирать данные по определенному критерию и многое другое. В свою очередь язык PHP позволяет установить соединение с сервером базы данных, выбрать одну из них для работы, а также посылать SQL-запросы. Рассмотрим все эти возможности более подробно.

PHP и базы данных

Итак, начнем непосредственное изучение функций PHP для работы с базами данных с того, что создадим небольшой опытный образец, с которым будем работать на протяжении всей этой главы.

Соединение с сервером базы данных

В первую очередь нужно убедиться, что сервер базы данных работает. Для этого запустите файл с расширением .bat, содержание которого мы рассматривали в главе 2. После этого в Диспетчере задач Windows должен появиться процесс с названием mysql-nt.exe (смотрите рис. 12.2). Напомним, что при работе над примерами в этой книге использовалась операционная система Windows 2000, поэтому не пугайтесь, если у вас будет что-то отличаться.

Чтобы начать работу с базой данных, нужно подключиться к серверу, где она находится. В нашем случае это будет локальный сервер базы

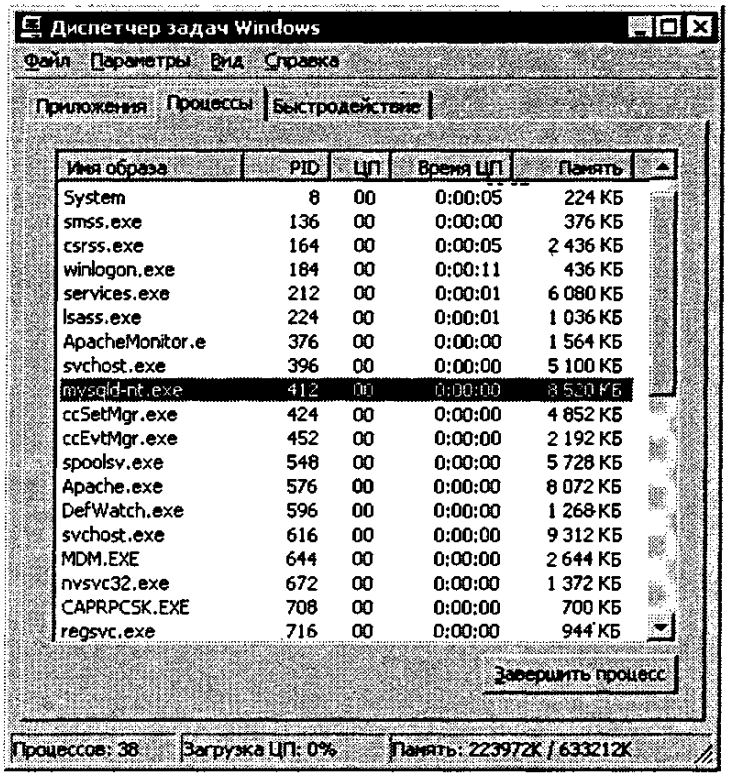

Рис. 12.2 ▾ Процесс сервера базы данных в Диспетчере задач Windows

данных MySQL. В PHP для этого есть две функции: mysql_connect() и mysql_pconnect(). Рассмотрим в качестве примера листинг 12.1.

Листинг 12.1 ▼ Соединение с сервером базы данных

```
<html>
<head>
   <title> Соединение с сервером базы данных </title>
</head>
<body>
<?php
// соединение с сервером базы данных
$link = mysql_connect("localhost", "root", "");
if (!$link)
{
   echo "Не могу соединиться с сервером базы данных";
   exit();
}
echo "Соединение с сервером базы данных произошло успешно";
// закрытие соединения с сервером базы данных
mysql_close($link);
?>
</body>
</html>
```

В этом примере функция mysql_connect() открывает соединение с сервером базы данных с именем localhost для пользователя root с паролем в виде пустой строки. При успешном выполнении она возвращает значение типа resource, которое является указателем на соединение с MySQ, в противном случае – значение FALSE. Как вы, наверное, догадались, дальнейшая работа с базой данных осуществляется с помощью полученного указателя. Если соединение с сервером базы данных более

не требуется, то его можно закрыть с помощью функции mysql_close(). Эту функцию можно использовать и без указания параметров: в этом случае закроется последнее созданное соединение.

Функция mysql_pconnect() используется в тех случаях, когда соединение с сервером базы данных нужно поддерживать постоянно. Другими словами, соединение не закроется, даже если завершится выполнение программы. Функция mysql_close() также не закрывает соединения, созданные посредством функция mysql_pconnect().

Создание и удаление базы данных

После удачного соединения с сервером базы данных можно производить различные действия, в частности создавать и удалять базы данных (листинг 12.2).

Листинг 12.2 ▾ Создание базы данных

```
<html>
<head>
   <title> Создание базы данных </title>
</head>
<body>
<?php
// соединение с сервером базы данных
if (!$link = mysql_connect("localhost", "root", ""))
{
   echo "<br>Не могу соединиться с сервером базы данных<br>";
   exit();
}
echo "<br>Соединение с сервером базы данных произошло успешно<br>";
// создание базы данных
if (!mysql_create_db("test_db", $link) )
{
```

```
    echo "<br>Не могу создать базу данных<br>";
    exit();
}
echo "<br>Создание базы данных произошло успешно<br>";
// закрытие соединения с сервером базы данных
mysql_close($link);
?>
</body>
</html>
```

Создание базы данных осуществляется с помощью функции PHP `mysql_create_db()`. Первый параметр этой функции задает имя создаваемой базы данных (`test_db`), а второй – указатель на соединение с сервером. В случае успеха функция `mysql_create_db()` возвращает TRUE, иначе – FALSE.

Для удаления базы данных имеется функция `mysql_drop_db()`, которая принимает такие же параметры, что и `mysql_create_db()`. Использовать эти функции очень просто, однако многие программисты не рекомендуют употреблять их в своих сценариях. Рассмотрим альтернативный способ создания и удаления баз данных с помощью SQL-запросов (листинг 12.3).

Листинг 12.3 ▼ Альтернативный способ создания базы данных

```
<html>
<head>
   <title> Альтернативный способ создания базы данных </title>
</head>
<body>
<?php
// соединение с сервером базы данных
if (!$link = mysql_connect("localhost", "root", ""))
```

```
{
    echo "<br>Не могу соединиться с сервером базы данных<br>";
    exit();
}
echo "<br>Соединение с сервером базы данных произошло успешно<br>";

// строка запроса
$str_sql_query = "CREATE DATABASE test_db";
// выполнение запроса
if (!mysql_query($str_sql_query, $link))
{
    echo "<br>Не могу выполнить запрос<br>";
    exit();
}
echo "<br>Создание базы данных произошло успешно<br>";
// закрытие соединения с сервером базы данных
mysql_close($link);
?>
</body>
</html>
```

Начало этой программы такое же, как в предыдущем примере. Мы создаем соединение с сервером базы данных с помощью функции `mysql_connect()`, затем составляем SQL-запрос. Если вы знакомы с языком SQL, то скорее всего приведенный запрос не вызовет у вас никаких затруднений. Ключевые слова `CREATE DATABASE` указывают на то, что нужно создать базу данных. Далее следует имя базы данных (`test_db`). Обратите внимание, что запрос представляет собой обычную строку. Чтобы послать SQL-запрос на сервер базы данных, используется функция `mysql_query()`. Ее первым параметром является строка запроса, вторым –

указатель на соединение. В данном примере эта функция возвратит TRUE, если база данных была создана, и FALSE в противном случае.

Для удаления базы данных используется SQL-запрос следующего вида:

```
DROP DATABASE test_db
```

Как вы, наверное, догадались, после ключевых слов DROP DATABASE нужно указать имя базы данных, которую требуется удалить.

Создание и удаление таблиц

Итак, теперь у нас есть база данных с именем test_db, однако она совершенно пустая. Чтобы ее наполнить, создадим таблицу с именем book, которую мы рассматривали в начале главы (листинг 12.4).

Листинг 12.4 ▼ Создание таблиц

```
<html>
<head>
   <title> Создание таблиц </title>
</head>
<?php
$sdb_name = "localhost";
$user_name = "root";
$user_password = "";
$db_name = "test_db";
// соединение с сервером базы данных
if (!$link = mysql_connect($sdb_name, $user_name, $user_password))
{
   echo "<br>Не могу соединиться с сервером базы данных<br>";
   exit();
}
// выбираем базу данных
if (!mysql_select_db($db_name, $link))
```

```
{
    echo "<br>Не могу выбрать базу данных<br>";
    exit();
}
// строка запроса
$str_sql_query = "CREATE TABLE book (id INT NOT NULL AUTO_INCREMENT,
PRIMARY KEY (id),
name VARCHAR(50),
author VARCHAR(50),
num_pages INT(10))";
// выполнение запроса
if (!mysql_query($str_sql_query, $link))
{
    echo "<br>Не могу выполнить запрос<br>";
    exit();
}
echo "<br>Таблица создана успешно<br>";
// закрытие соединения с сервером базы данных
mysql_close($link);
?>
</body>
</html>
```

Для удобства в самом начале имеет смысл ввести несколько переменных, которые будут содержать в себе имена баз данных, таблиц, пользователя и т.д. Затем соединяемся с сервером. Для работы с определенной базой данных нужно ее выбрать. Эту операцию выполняет функция `mysql_select_db()`, которая принимает в качестве входных параметров имя базы данных и указатель на соединение. В случае отсутствия указателя

используется последнее созданное соединение с сервером базы данных. Функция возвращает TRUE в случае успеха, иначе – FALSE.

В нашем примере непосредственное создание таблицы осуществляется с помощью SQL-запроса. Ключевые слова CREAT TABLE говорят серверу базы данных, что нужно создать таблицу. Затем следует ее имя (в данном случае book), после чего в скобках описываются все поля этой таблицы с указанием типов данных, а именно id (идентификационный номер), name (название), author (автор), num_pages (количество страниц).

Запрос посылается серверу таким же образом, как в предыдущем примере – с помощью функции mysql_query().

Для удаления таблицы выполняются такие же действия, но с другим запросом:

```
DROP TABLE book
```

Работа с данными

В результате выполнения программы в предыдущем примере мы создали структуру базы данных. Теперь рассмотрим добавление, удаление и изменение данных. Например, внесем несколько записей в нашу таблицу book. Для этого в строке запроса используется ключевые слова INSERT INTO (листинг 12.5).

Листинг 12.5 ▼ Добавление данных

```
<html>
<head>
   <title> Добавление данных </title>
</head>
<body>
<?php
$sdb_name = "localhost";
$user_name = "root";
$user_password = "";
```

```
$db_name = "test_db";
// соединение с сервером базы данных
if (!$link = mysql_connect($sdb_name, $user_name, $user_password))
{
   echo "<br>Не могу соединиться с сервером базы данных<br>";
   exit();
}
// выбираем базу данных
if (!mysql_select_db($db_name, $link))
{
   echo "<br>Не могу выбрать базу данных<br>";
   exit();
}
// строка запроса
$str_sql_query = "INSERT INTO book (name, author, num_pages) VALUES ('Самоучи-
тель PHP','Иванов','222')";
// выполнение запроса
if (!mysql_query($str_sql_query, $link))
{
   echo "<br>Не могу выполнить запрос<br>";
   exit();
}
echo "<br>Запись добавлена успешно<br>";
// закрытие соединения с сервером базы данных
mysql_close($link);
?>
</body>
</html>
```

После запуска этого сценария, в таблице book появится запись о книге «Самоучитель PHP», автор которой некий Иванов, а количество страниц равно 222. Обратите внимание, что поле id не указанно в запросе. Дело в том, что при создании оно было отмечено с параметром AUTO_INCREMENT (автоувеличение). Другими словами, при добавлении записи это поле автоматически получает значение на единицу больше, чем в предыдущей записи. Отсчет начинается с единицы.

Таким образом, можно добавлять другие записи. Дальше мы будем работать с таблицей, данные которой представлены на рис. 12.3.

id	name	author	num_pages
1	Самоучитель PHP	Иваиов Петр Сергеевич	222
2	Самоучитель Perl	Мелешко Виталий Николаевич	333
3	Справочник по математике	Дитятева Аниа Викторовиа	415
4	Справочник по физике	Доброхотова Татьяна Петровна	312

Рис. 12.3 ▼ Данные таблицы book

Для удаления записей используется SQL-запрос с ключевым словом DELETE. Пример:

```
DELETE FROM book WHERE author = 'Иванов'
```

Несложно заметить, что после слов DELETE FROM нужно указать имя базы данных (book), а затем условие, по которому можно найти записи для удаления.

Для изменения уже существующих данных используется команда UPDATE. Пример:

```
UPDATE book SET num_pages = 333 WHERE name = 'Самоучитель PHP'
```

После слова UPDATE нужно указать имя таблицы, где необходимо изменить данные. Затем задается новое значение (в нашем случае мы изменяем количество страниц с 222 на 333) и условие.

Тем не менее наиболее часто требуется не добавлять и удалять данные, а выводить их. При этом можно задавать различные условия вывода, например с ограничением страниц или определенного автора. Вариаций на эту тему можно придумать великое множество. Однако в нашем случае главное уяснить, как можно получить данные и как с ними работать.

Рассмотрим простой пример, в котором в окне браузера будет выводиться вся информация о книгах, имеющихся в таблице book (листинг 12.6).

Листинг 12.6 ▾ Вывод данных из таблицы

```
<html>
<head>
   <title> вывод данных из таблицы </title>
</head>
<body>
<?php
$sdb_name = "localhost";
$user_name = "root";
$user_password = "";
$db_name = "test_db";
// соединение с сервером базы данных
if (!$link = mysql_connect($sdb_name, $user_name, $user_password))
{
   echo "<br>Не могу соединиться с сервером базы данных<br>";
   exit();
}
// выбираем базу данных
if (!mysql_select_db($db_name, $link))
{
   echo "<br>Не могу выбрать базу данных<br>";
   exit();
}
// строка запроса
$str_sql_query = "SELECT * FROM book";
```

```
// выполнение запроса
if (!$result = mysql_query($str_sql_query, $link))
{
   echo "<br>Не могу выполнить запрос<br>";
   exit();
}
// вывод результата запроса
while ($mas = mysql_fetch_row($result))
{
   foreach ($mas as $field)
   {
         echo $field . " ";
   }
   echo "<br>";
}
// закрытие соединения с сервером базы данных
mysql_close($link);
?>
</body>
</html>
```

Несмотря на то, что бóльшая часть кода этой программы уже встречалась, есть смысл повторно разобрать его. Как говорится, повторение – мать учения. Итак, все начинается с инициализации основных переменных. Затем нужно соединиться с сервером базы данных. Осуществляется это с помощью функции PHP `mysql_connect()`, где в качестве входных параметров задаем имя сервера базы данных (`localhost`), а также имя и пароль пользователя. В случае удачного соединения эта функция возвращает указатель типа source, с которым мы будем работать в дальнейшем, иначе осуществляется вывод сообщения об ошибке и выход из программы.

Следующим этапом будет выбор базы данных посредством функции `mysql_select_db()`. Здесь мы должны указать имя базы данных и указатель на соединение. Если база данных выбрана, то функция возвращает значение TRUE, иначе – FALSE.

После выбора базы данных можно выполнять различные действия над ней. В нашем случае требуется вывести все ее содержимое. Для этого мы создаем SQL-запрос и выполняем его с помощью функции `mysql_query()`. Возвращаемое ею значение запишем в переменную `$result`.

Работать с данными, полученными в результате выполнения запроса, можно различными способами. В нашем случае – это функция `mysql_fetch_row()`, которая обрабатывает ряд результата запроса и возвращает его в качестве массива, пронумерованного с нуля. Непосредственный вывод данных в окно браузера производится в цикле `while`. С каждым вызовом функции `mysql_fetch_row()` специальный указатель перемещается к следующему ряду результата запроса.

Заключение к главе

В этой главе были рассмотрены основные принципы работы PHP с реляционными базами данных MySQL. Как вы, наверное, убедились, PHP имеет широкий ряд функций для этого. Конечно, для эффективной работы с базами данных требуются некоторые навыки программирования на языке SQL. Тем не менее, внимательно изучив эту главу, вы сможете создавать довольно функциональные PHP-приложения с использованием баз данных MySQL.

Глава 13 Работа с изображениями

Посещая Web-сайты, вы, наверное, встречали графические счетчики посещений, графики, диаграммы и многое другое. Основная особенность этих изображений заключается в том, что они являются динамическими.

В этой главе мы познакомимся с основами создания таких изображений посредством PHP, а также разберем несколько полезных примеров их использования.

Изображение – это...

По сути, изображение – это множество пикселей (точек), каждый из которых представляет определенный цвет. Существует множество форматов изображений, отличающихся степенью сжатия, качеством и многим другим. В сети Internet наиболее распространенны GIF (Graphic Interchange Format), JPEG (Joint Photographic Experts Group) и PNG (Portable Network Graphics). Выбор того или иного формата определяется, прежде всего, характером изображения. Если это цветные фотографии, изобилующие различными оттенками цветов, то стоит применять JPEG. Форматы GIF и PNG используются для небольших

изображений, например баннеров, кнопок и т.п., где содержится малое количество цветов.

Библиотека GD

Для работы PHP с различными изображениями нужно установить специальную библиотеку GD, которая содержит множество функций для создания двумерной графики. С ее помощью можно рисовать различные геометрические фигуры, писать текст, производить заливку определенным цветом и многое другое.

Установить библиотеку GD можно следующим образом:

1. Откройте конфигурационный файл php.ini.
2. Найдите строку extension=php_gd2.dll.
3. Удалите символ точку с запятой (;).
4. Сохраните изменения и закройте файл.
5. Перезапустите Web-сервер Apache.

Теперь все функции этой библиотеки будут доступны вашим PHP-сценариям. Обратите внимание, что название библиотеки содержит цифру 2, что говорит о ее текущей версии (2.x.x).

Функциональность этой библиотеки очень велика, однако существует ряд ограничений, связанных с форматом изображений GIF. Произошло это по следующей причине. GIF использует алгоритм сжатия LZW (Lempel Ziv Weich), который принадлежит компании Unisys Corp. Когда срок действия бесплатной лицензии истек, разработчики библиотеки GD решили отказаться от поддержки данного формата. Произошло это, начиная с версии 1.6. Сейчас вышла новая версия библиотеки GD 2.0.28, в которой, как сообщают разработчики, восстанавливается работа с форматом GIF. Тем не менее в этой книге приводятся примеры с использованием формата PNG, что позволит избежать путаницы.

Надо отметить, что использование библиотеки GD является не единственным способом работы PHP с изображениями. Для этого существует множество других библиотек и программ.

Создание и вывод изображений

Прежде чем приступить к непосредственному созданию изображений, ознакомимся с информацией о библиотеке GD. Для этого воспользуемся функцией библиотеки gd_info() – листинг 13.1.

Листинг 13.1 ▼ Вывод информации о библиотеке GD

```
<html>
<head>
   <title> вывод информации о библиотеке GD </title>
</head>
<body>
<?php
print_r(gd_info());
?>
</pre>
</body>
</html>
```

Результат выполнения этой программы смотрите на рис. 13.1.

Функция gd_info() возвращает массив, в котором отражена информация о текущей версии библиотеки, о поддержке того или иного формата изображений и т.п. Заметьте, что в нашем случае создание изображений формата GIF не поддерживается.

Итак, приступим к созданию изображений. Для начала нарисуем обычный квадрат черного цвета (листинг 13.2).

Листинг 13.2 ▼ Вывод квадрата черного цвета

```
<?php
// посылает заголовок браузеру об изображении формата PNG
header("Content-type: image/png");
// создает изображение в памяти и возвращает его идентификатор
```

```
$image = imagecreatetruecolor (100, 100);
// вывод сообщение в окно браузера
imagepng($image);
// уничтожение изображения из памяти
imagedestroy($image);
?>
```

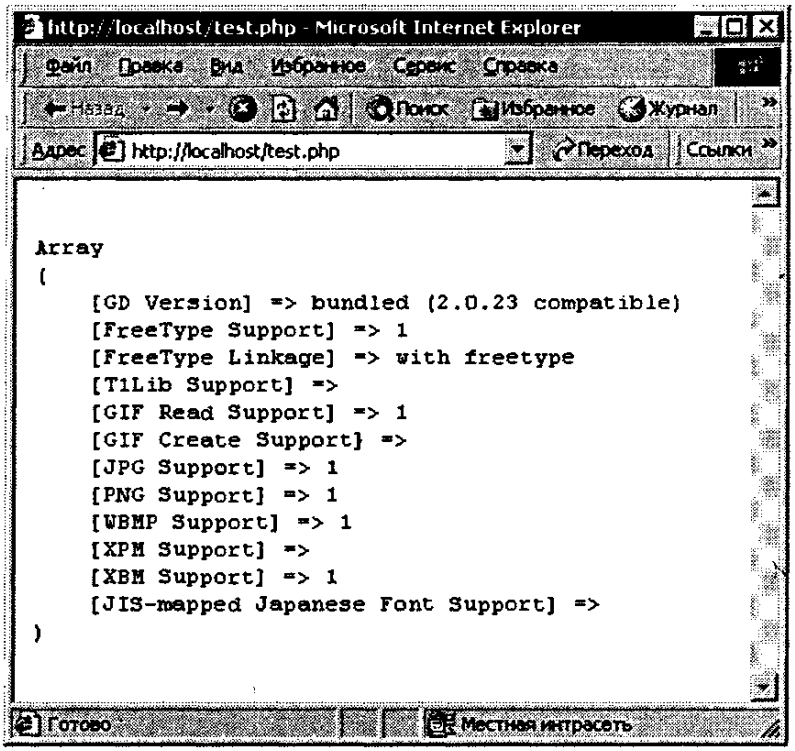

Рис. 13.1 ▾ Информация о библиотеке GD

Результат выполнения этой программы смотрите на рис. 13.2.

С помощью функции `header()` мы сообщаем браузеру, что передаваемая информация является изображением формата PNG. Следующая функция `imagecreateTRUEcolor()` создает в памяти изображение в виде прямоугольника черного цвета с размерами, указанными в качестве входных параметров, и возвращает его целочисленный идентификатор (тип resource). Этот идентификатор имеет смысл сохранить в переменную, так как последующая работа с изображением будет происходить непосредственно с ним: например, как это делается в следующей строчке при использовании

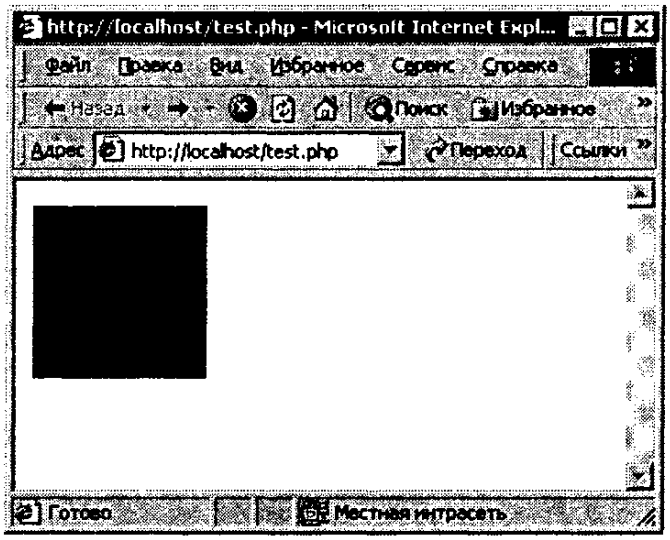

Рис. 13.2 ▼ Черный квадрат

функции `imagepng()`, которая выводит изображение в окно браузера. В качестве входного параметра этой функции используется именно сохраненный идентификатор. И наконец, с помощью функции `imagedestroy()` мы уничтожаем изображение, тем самым освобождая память.

Существует альтернативный способ создания изображения с помощью его загрузки из соответствующего файла (листинг 13.3).

Листинг 13.3 ▼ Вывод изображения из файла

```
<?php
header("Content-type: image/png");
// загрузка изображения из файла
$image = imagecreatefrompng("black_square.png");
// вывод изображения
imagepng($image);
// уничтожение изображения из памяти
imagedestroy($image);
?>
```

В данном случае мы предварительно создали изображение черного квадрата и сохранили его в файл black_square.png, который поместили в одну папку с вызываемым файлом PHP. В программе с помощью функции

imagecreatefrompng() создается соответствующее изображение. Далее все повторяется как в предыдущем примере.

Для вывода изображений также существует альтернативный способ, в котором используется HTML-тег <img>. Создадим файл black_square. php со следующим содержанием – см. листинг 13.4.

Листинг 13.4 ▾ Создание изображения

```
<?php
header("Content-type: image/png");
$image = imagecreatetruecolor (100, 100);
?>
```

Для вставки изображения в HTML-файл в его текст добавляем следующий код:

```
<img src=" black_square.php ">
```

В результате выведется черный квадрат.

Модификация изображений

После того как вы научились создавать и выводить изображение, можно приступать к рисованию. Сразу надо сказать, что приведенные примеры не претендуют на качественные дизайнерские работы, поэтому не стоит к ним относиться больше, чем к просто демонстрации возможностей функций библиотеки GD.

Рисование геометрических фигур

Разобьем наш черный квадрат двумя диагональными белыми линиями (листинг 13.5).

Листинг 13.5 ▾ Рисование квадрата с диагоналями

```
<?php
header("Content-type: image/png");
$image = imagecreatetruecolor (100, 100);
// определение цвета
$white = imagecolorallocate($image, 255, 255, 255);
```

```
// рисование линии
imageline($image, 0, 0, 99, 99, $white);
imageline($image, 0, 99, 99, 0, $white);
// вывод изображения
imagepng($image);
// удаление изображения из памяти
imagedestroy($image);
?>
```

Заметьте, что прежде чем рисовать линии, мы определили белый цвет с помощью функции `imagecolorallocate()`, в параметры которой входят не только числа, определяющие непосредственно цвет (от 0 до 255), но и идентификатор изображения. Эта функция возвращает целочисленное значение, которое будет определять цвет, используемый в изображении.

Непосредственная прорисовка линии осуществляется с помощью функции `imageline()`, где мы указываем изображение, координаты начала и конца линии и ее цвет. В итоге получается такая же картинка как на рис. 13.3.

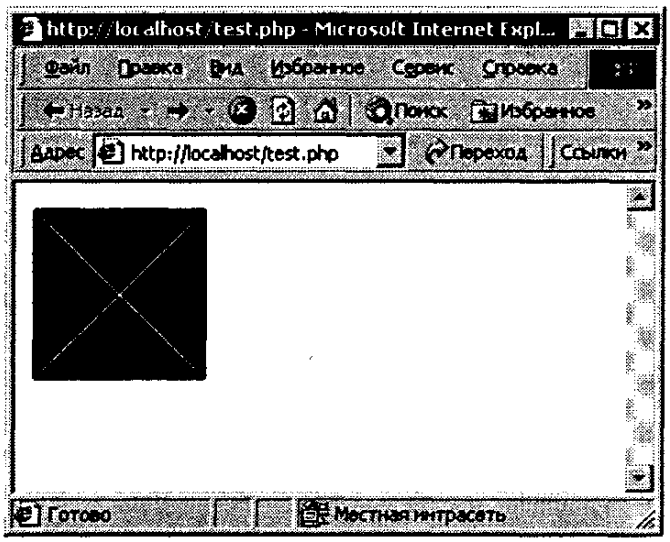

Рис. 13.3 ▼ Черный квадрат с белыми диагоналями

Итак, прямые линии рисовать оказалось весьма просто, поэтому дальнейшие эксперименты можете проделывать самостоятельно. Перейдем к рисованию дуг, причем изобразим их на зеленом фоне (листинг 13.6.).

Листинг 13.6 ▼ Рисование дуг

```
<?php
header("Content-type: image/png");
// создание пустого изображения
$image = imagecreate (100, 100);
// цвет фона зеленый
$green = imagecolorallocate($image, 0, 128, 0);
// цвет белый
$white = imagecolorallocate($image, 255, 255, 255);
// рисование кривых линий
imagearc($image, 49, 49, 90, 90, 0, 360, $white);
imagearc($image, 49, 70, 50, 20, 0, 180, $white);
imagearc($image, 49, 49, 5, 20, 0, 360, $white);
imagearc($image, 29, 30, 20, 5, 0, 360, $white);
imagearc($image, 69, 30, 20, 5, 0, 360, $white);
// вывод изображения
imagepng($image);
// уничтожение изображения из памяти
imagedestroy($image);
?>
```

Заметьте, что в этом случае мы использовали функцию `imagecreate()`, которая создает пустое изображение. Первый определенный цвет при этом будет являться фоновым.

Функция `imagearc()` рисует часть эллипса. Сначала надо задать идентификатор изображения, затем координаты центра эллипса, его ширину

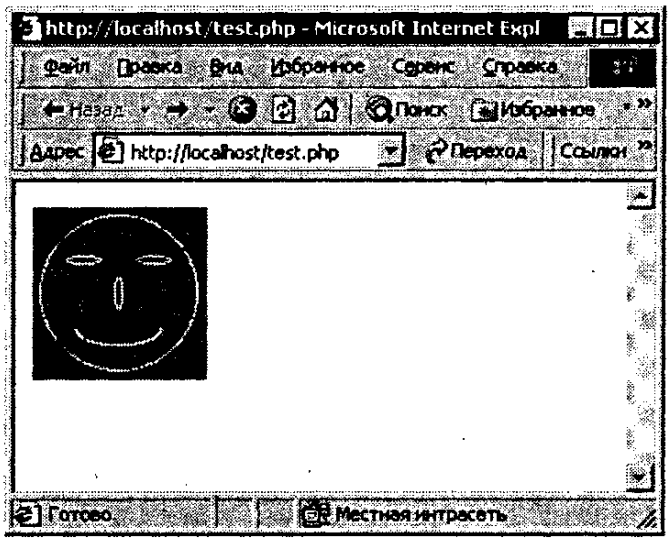

Рис. 13.4 ▼ Картинка, нарисованная дугами

и длину, начальный и конечный угол в градусах и, наконец, цвет линии. Результат выполнения этой программы представлен на рис. 13.4.

В библиотеке GD вы также найдете функции `imagerectangle()` и `imagepolygon()`, которые рисуют соответственно прямоугольник и многоугольник. Если вам потребуется не только начертить контуры этих фигур, но и закрасить их определенным цветом, то можно применить функции `imagefilledrectangle()` и `imagefilledpolygon()`.

Разберем случай, когда требуется закрасить определенную область изображения. Для этого применяют функцию `imagefill`. Например, изменим цвет лица на предыдущей картинке (листинг 13.7).

Листинг 13.7 ▼ Заливка цветом

```
<?php
header("Content-type: image/png");
$image = imagecreate (100, 100);
$green = imagecolorallocate($image, 0, 128, 0);
$yellow = imagecolorallocate($image, 255, 255, 0);
$black = imagecolorallocate($image, 0, 0, 0);
imagearc($image, 49, 49, 90, 90, 0, 360, $black);
```

```
imagearc($image, 49, 70, 50, 20, 0, 180, $black);
// закрашиваем лицо в желтый цвет
imagefill ($image, 49, 49, $yellow);
imagearc($image, 49, 49, 5, 20, 0, 360, $black);
imagearc($image, 29, 30, 20, 5, 0, 360, $black);
imagearc($image, 69, 30, 20, 5, 0, 360, $black);
// вывод изображения
imagepng($image);
// уничтожение изображения из памяти
imagedestroy($image);
?>
```

Здесь мы не только изменили цвет лица, но и поменяли цвет линий (см. рис. 13.5).

Принцип работы функции `imagefill()` заключается в следующем. Ставится точка на изображении с координатами и цветом, указанными в качестве входного параметра, и закрашивается все вокруг нее до определенных границ.

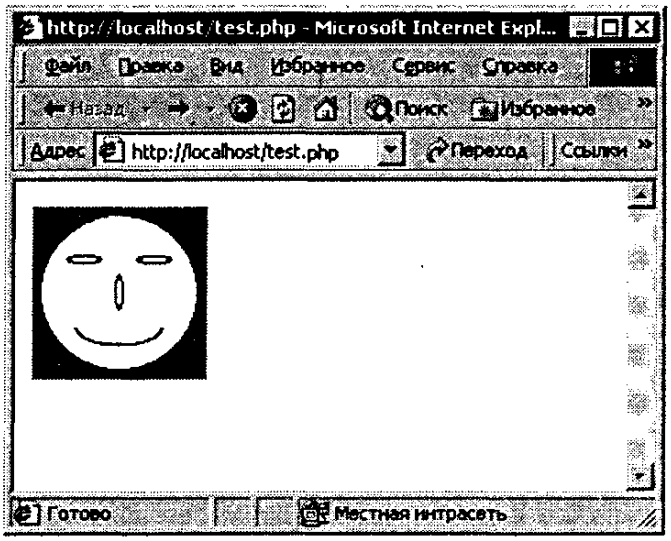

Рис. 13.5 ▾ Заливка цветом

Также существует функция imagefillborder(), которая не только осуществляет заливку области, но и позволяет провести ее границу определенным цветом.

Работа с текстом

Рисование линий, прямоугольников и других фигур является важной частью при работе с изображениями. Однако зачастую все это было бы не столь информативно, если бы не возможность писать текст на них. Для этого в библиотеке GD имеется множество функций, одна из которых imagestring() — листинг 13.8.

Листинг 13.8 ▼ Вывод строки на изображении

```
<?php
header("Content-type: image/png");
$image = imagecreatetruecolor (100, 100);
$white = imagecolorallocate($image, 255, 255, 255);
// создание строки
imagestring($image, 3, 5, 49, "Hello, World!", $white);
// вывод изображения
imagepng($image);
// уничтожение изображения из памяти
imagedestroy($image);
?>
```

В результате выполнения этой программы выводится текст «Hello, World!» на черном фоне квадрата (см. рис. 13.6).

В качестве входных параметров этой функции выступают соответственно идентификатор изображения, номер встроенного шрифта (от 1 до 5), координаты первой буквы, строка для вывода и, наконец, цвет текста.

Помимо разобранной функции в библиотеке GD имеется еще несколько альтернативных способов вывести строку. Например, функция

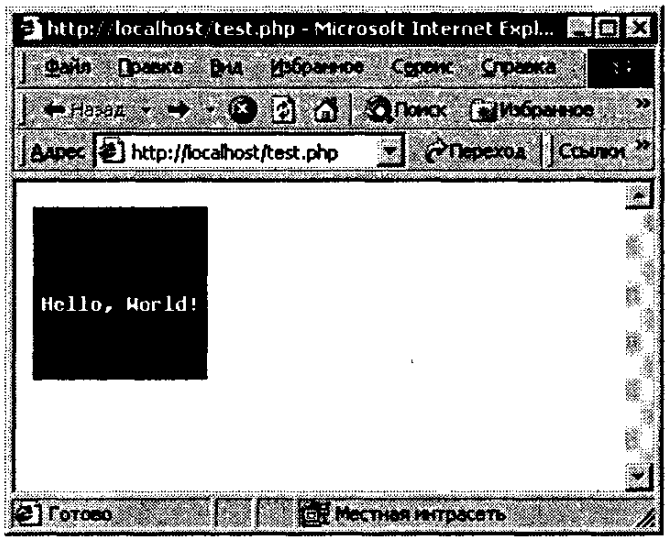

Рис. 13.6 ▾ Вывод строки на изображение

imagettftext(), которая позволяет использовать шрифты TrueType (листинг 13.9).

Листинг 13.9 ▾ Вывод строки на изображении с помощью функции imagettftext()

```
<?php
header("Content-type: image/png");
$image = imagecreatetruecolor (100, 100);
$white = imagecolorallocate($image, 255, 255, 255);
// создание строки
imagettftext($image, 9, 0, 5, 49, $white, "Snap.ttf", "Hello, World!");
// вывод изображения
imagepng($image);
// удаление изображения из памяти
imagedestroy($image);
?>
```

В этом примере мы использовали файл Snap.ttf, который содержит шрифт Snap ITC. Входные параметры этой функции следующие: идентификатор изображения, размер шрифта, угол поворота строки против часовой стрелки, координаты начала строки, ее цвет, путь к файлу со

шрифтом (здесь он находится в одной папке с вызываемым файлом PHP) и строка для вывода.

Иногда бывает очень полезным знать длину и высоту выводимой строки, так как зачастую она постоянно пытается выйти за пределы изображения. Для решения этой проблемы применяют функции `imagefontheight()`, `imagefontwidth()` и `imagettfbbox()`.

Принцип работа первой и второй функции очень прост. Мы задаем номер внутреннего шрифта в качестве входного параметра, а нам возвращается соответственно высота и длина символа (листинг 13.10).

Листинг 13.10 ▼ Использование длины и высоты строки

```
<?php
header("Content-type: image/png");
$image = imagecreatetruecolor (200, 200);
$white = imagecolorallocate($image, 255, 255, 255);
// строка для вывода
$str = "Hello, World!";
// расчет координат начала строки
$x = (200 - strlen($str) * imagefontwidth(4)) / 2;
$y = (200 - imagefontheight(4)) / 2;
// создание строки
imagestring($image, 4, $x, $y, $str, $white);
// вывод изображения
imagepng($image);
// удаление изображения из памяти
imagedestroy($image);
?>
```

В этом примере мы добиваемся вывода строки приблизительно посередине квадрата. Заметьте, что для вычисления длины строки мы умножили длину символа на количество букв в строке.

Однако дело осложняется, если текст находится под определенным углом. Для этого применяют функцию imagettfbbox() – листинг 13.11).

Листинг 13.11 ▼ Использование функции imagettfbbox()

```
<?php
header("Content-type: image/png");
$image = imagecreatetruecolor (100, 100);
$white = imagecolorallocate($image, 255, 255, 255);
// записываем массив координат
$mas_loc = imagettfbbox(15, 23, "Snap.ttf", "Hello, World!");
// расчет координат начала строки
$x = 100 - ($mas_loc[0] + $mas_loc[2] + $mas_loc[4] + $mas_loc[6]) / 4;
$y = 100 - ($mas_loc[1] + $mas_loc[3] + $mas_loc[5] + $mas_loc[7]) / 4;
// создание строки
imagettftext($image,15, 23, $x, $y, $white, "Snap.ttf", "Hello, World!");
// вывод изображения
imagepng($image);
// уничтожение изображения из памяти
imagedestroy($image);
?>
```

Входными параметрами функции imagettfbbox() являются соответственно размер шрифта, угол поворота строки, путь к файлу со шрифтом и строка для вывода. Функция возвращает массив с координатами углов прямоугольника, описывающего строку в порядке, представленном в табл. 13.1.

Таблица 13.1 ▼ Порядок описания строки

Ключ	Значение Координаты
0	Левый нижний угол (ось OX)
1	Левый нижний угол (ось OY)

Таблица 13.1 ▼ Порядок описания строки (окончание)

Ключ	Значение Координаты
2	Правый нижний угол (ось OX)
3	Правый нижний угол (ось OY)
4	Правый верхний угол (ось OX)
5	Правый верхний угол (ось OY)
6	Левый верхний угол (ось OX)
7	Левый верхний угол (ось OY)

Заключение к главе

На этом заканчивается обзор основных графических возможностей PHP и библиотеки GD. Надо отметить, что разобранные функции являются всего лишь малой частью от всех остальных, поэтому у вас будет над чем поработать.

Глава 14 Работа с датой и временем

Крылатая фраза «время – деньги» стала в последние годы особенно актуальна. Ритм жизни в мире настолько высок, что порой не успеваешь за ним угнаться. Все это не могло не отразиться на специфике решаемых в сети Internet задач. Самая обыденная из них – это потребность человека в точном времени. Однако иногда интересно узнать, сколько времени прошло с последнего посещения страницы, какая посещаемость Web-сайта за определенный промежуток времени. Все это и многое другое реализуется с помощью функций времени PHP.

Время – это...

На данный момент единицей времени считается секунда. Раньше ее эталоном считалась 1/86400 часть от средних солнечных суток. Однако такое положение вещей не могло устроить людей, так как погрешность была очень высока. Поэтому в 1967 году на XIII Международной конференции по мерам и весам был введен другой эталон, равняющийся периоду времени, за которое атом цезия-133 производит 9 192 631 700 колебаний. Шестьдесят секунд – это минута, шестьдесят минут – это час и т.д. Все эти и другие элементы составляют время. Однако мы отвлеклись от основной темы.

Особенности времени в PHP

«Спросить» PHP сколько сейчас времени можно с помощью функции `time()` – листинг 14.1.

Листинг 14.1 ▼ Вывод текущего времени

```
<html>
<head>
   <title> вывод текущего времени </title>
</head>
<body>
<?php
echo time(); // выводит 1092486666
?>
</body>
</html>
```

Заметьте, что в результате выводится многозначное целое число, которое соответствует количеству секунд, прошедших с начала эпохи Unix (The Unix Epoch, 1 января 1970, 00:00:00 GMT) до текущего времени, которое установлено на сервере (абсолютное время). На первый взгляд такой способ представления времени выглядит весьма странным. Однако как показала практика, он позволяет избежать многих трудностей при работе с датами.

Помимо функции `time()` в PHP имеется еще несколько возможностей выяснить текущее время, причем в более наглядном формате. Например, можно использовать функцию `date()`. Ее отличительная особенность заключается в том, что программист может самостоятельно задавать формат вывода даты и времени (листинг 14.2).

Листинг 14.2 ▼ Форматный вывод текущего времени

```
<html>
<head>
```

```
    <title> Форматный вывод текущего времени </title>
</head>
<body>
<?php
echo date("d.m.y");
?>
</body>
</html>
```

В результате в окне браузера будет выведено следующее: «14.08.04». Список символов форматирования можно посмотреть в таблице 14.1.

Таблица 14.1 ▼ Символы форматирования даты и времени

Символ значения	Описание	Возможные
a	Ante meridiem или Post meridiem в нижнем регистре	am или pm
A	Ante meridiem или Post meridiem в верхнем регистре	AM или PM
B	Время в стадарте Swatch Internet	От 000 до 999
c	Дата в формате ISO 8601 2004-08-14T19: (добавлено в PHP 5)	46:43+04:00
d	День месяца, 2 цифры с ведущими нулями	От 01 до 31
D	Сокращенное наименование дня недели	От Mon до Sun
F	Полное наименование месяца	От January до December
g	Часы в 12-часовом формате без ведущих нулей	От 1 до 12
G	Часы в 24-часовом формате без ведущих нулей	От 0 до 23
h	Часы в 12-часовом формате с ведущими нулями	От 01 до 12
H	Часы в 24-часовом формате с ведущими нулями	От 00 до 23

Таблица 14.1 ▼ Символы форматирования даты и времени (окончание)

Символ значения	Описание	Возможные
i	Минуты с ведущими нулями	От 00 до 59
j	День месяца без ведущих нулей	От 1 до 31
l	Полное наименование дня недели	От Sunday до Saturday
L	Признак високосного года	1, если год високосный, иначе 0
m	Порядковый номер месяца с ведущими нулями	От 01 до 12
M	Сокращенное наименование месяца, 3 символа	От Jan до Dec
n	Порядковый номер месяца без ведущих нулей	От 1 до 12
O	Разница со временем по Гринвичу в часах	Например: +0200
s	Секунды с ведущими нулями	От 00 до 59
t	Количество дней в месяце	От 28 до 31
w	Порядковый номер дня недели	От 0 (воскресенье) до 6 (суббота)
Y	Порядковый номер года	4 разряда
y	Номер года	2 разряда
z	Порядковый номер дня в году	От 0 до 365

В том случае если вам потребуется вывести помимо времени какие-либо другие символы, совпадающие с символами форматирования, то их нужно экранировать с помощью знака \ (листинг 14.3).

Листинг 14.3 ▼ Особенности форматного вывода текущего времени

```
<html>
<head>
   <title> Особенности форматного вывода текущего времени </title>
```

```
</head>
<body>
<?php
echo date("\\t\he \\t\i\me \i\s H:i:s");// выводит "the time is 18:53:10"
?>
</body>
</html>
```

Обратите внимание, что буква t экранирована дважды, так как \t образует специальный символ горизонтальной табуляции.

Помимо текущего времени, функция date() может преобразовывать время, заданное в качестве входного параметра (листинг 14.4).

Листинг 14.4 ▼ Особенности форматного вывода заданного времени

```
<html>
<head>
   <title> Особенности форматного вывода заданного времени </title>
</head>
<body>
<?php
echo date("d.m.y H:i:s", 0);// выведет "01.01.70 03:00:00"
?>
</body>
</html>
```

Как вы, наверное, догадались, в качестве параметра задается количество секунд, прошедших с начала эпохи Unix. Однако вы можете задать резонный вопрос о появлении дополнительных трех часов. Произошло это по следующей причине. Сервер, на котором запускалась данная программа, находился на платформе, где время настроено по третьему часовому поясу.

Надо отметить, что функция date() очень удобна для моментального вывода даты и времени. Однако если требуется работать с отдельными элементами (часы, минуты и т. д.), то для этого лучше применять функцию getdate() – листинг 14.5.

Листинг 14.5 ▼ Вывод текущего времени с помощью функции getdate()

```
<html>
<head>
   <title> вывод текущего времени с помощью функции getdate() </title>
</head>
<body>
<pre>
<?php
print_r (getdate());
?>
</pre>
</body>
</html>
```

Функции getdate() возвращает массив. Значения ключей и элементов можно посмотреть на рис. 14.1.

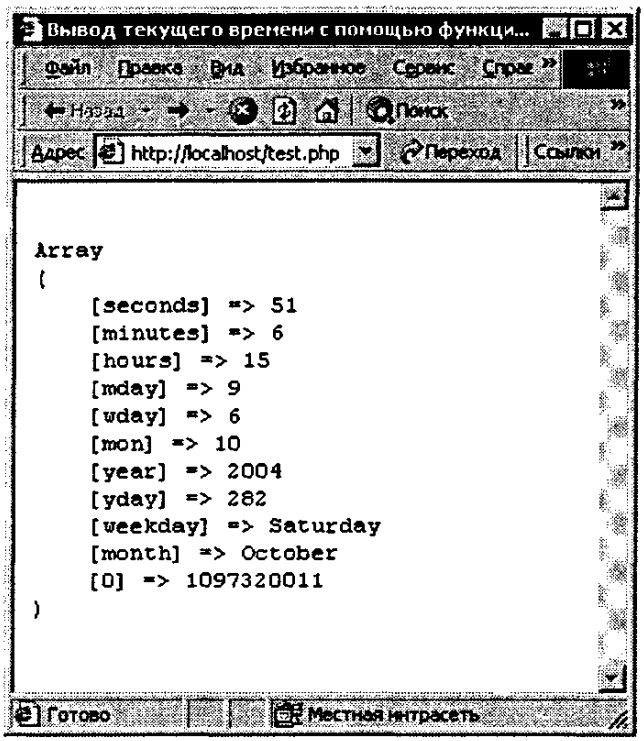

Рис. 14.1 ▼ Вывод массива, возвращаемого функцией getdate()

Описание элементов массива смотрите в табл. 14.2.

Таблица 14.2 ▼ Описание элементов массива

Ключ	Описание	Возможные значения	Тип данных
Second	Секунды	От 0 до 59	Integer
Minutes	Минуты	От 0 до 59	Integer
Hours	Часы	От 0 до 23	Integer
Mday	Номер дня месяца	От 1 до 31	Integer
Wday	Номер дня недели	От 0 (воскресенье) до 6 (суббота)	Integer
Mon	Месяц	От 1 до 12	Integer
Year	Год	4 разряда	Integer
Yday	Номер дня в году	От 0 до 366	Integer
Weekday	День недели	От Sunday до Saturday	String
Month	Название месяца	От January до December	String
0	Количество секунд	Зависит от платформы	Integer

Помимо преобразования абсолютного времени к привычному для нас, часто требуется выполнить обратное действие. Для этого обычно используют функцию `mktime()` – листинг 14.6.

Листинг 14.6 ▼ Преобразования к абсолютному времени

```
<html>
<head>
   <title> Преобразования к абсолютному времени </title>
</head>
<body>
<?php
$hour = 0;
$minute = 0;
$second = 0;
```

```
$month = 5;
$day = 5;
$year = 1984;
// выводит 452548800
echo mktime($hour, $minute, $second, $month, $day, $year);
?>
</body>
</html>
```

Как ни странно, но все параметры функции `mktime()` являются необязательными и поэтому могут быть опущены в порядке справа налево. Если параметр не указан, то функция работает со значением текущего времени. Другими словами, функция `mktime()` без параметров будет эквивалентна функции `time()`.

Значение года может быть записано как с помощью четырехзначного числа, так и двухзначного или однозначного. При этом числа от 0 до 69 соответствуют годам XXI века (например, 2054), а 70–99 – XX (например, 1984). Тем не менее во избежание путаницы лучше записывать год четырьмя цифрами.

При использовании функции `mktime()` на платформе Windows нужно также учитывать то обстоятельство, что здесь не может быть отрицательных временных меток (листинг 14.7).

Листинг 14.7 ▼ Особенности преобразования к абсолютному времени

```
<html>
<head>
   <title> Особенности преобразования к абсолютному времени </title>
</head>
<body>
<?php
$hour = 0;
$minute = 0;
```

```
$second = 0;
$month = 5;
$day = 5;
$year = 1954;
// Если у вас платформа Windows, то будет ошибка
echo mktime($hour, $minute, $second, $month, $day, $year);
?>
</body>
</html>
```

В этом примере у нас возникнет ошибка, так как 1954 год соответствует отрицательному значению абсолютного времени. Однако это не единственное ограничение на значение года. Так как обычно значение типа Integer не может превышать 2 147 483 647, то 2 038 год будет границей использования функции `mktime()`.

Заключение к главе

Несложно заметить, что при работе со временем и датой нужно быть предельно внимательным. Поэтому проверяйте свои сценарии по несколько раз перед размещением в сети Internet.

Глава 15

Работа с регулярными выражениями

При решении Web-задач часто сталкиваешься с ситуацией, когда нужно проверить корректность вводимой пользователем информации. Реализовать данную проверку с помощью обычных строковых функций, разобранных в главе 9, довольно сложно, так как они специально не предназначены для этого. Самый эффективный способ решения подобных задач – это регулярные выражения.

На первый взгляд регулярные выражения покажутся вам сродни китайскому языку, так как они изобилуют непонятными сочетаниями символов и знаков препинания. Тем не менее даже самые малые знания в этой области помогут вам быстро решить многие трудные задачи.

Регулярные выражения – это...

Регулярное выражение – это строка. Данное определение менее пугает, чем механизм, позволяющий указать шаблон для строки и осуществить поиск данных, соответствующих этому шаблону в заданном тексте. Регулярное выражение действительно задает шаблон в виде строки. Делается это с помощью специальных символов, о которых пойдет речь в этой главе.

PHP позволяет работать с двумя типами регулярных выражений: POSIX и PCRE, каждый из которых обладает собственным синтаксисом. Регулярные выражения типа PCRE с небольшими изменениями перешли из языка Perl, они считаются более мощными и выполняются гораздо быстрее, чем регулярные выражения типа POSIX. В этой главе мы познакомимся с регулярными выражениями типа POSIX.

Шаблоны

Прежде чем приступать к непосредственному изучению синтаксиса регулярных выражений, нужно уяснить принцип соответствия строки шаблону. На самом деле шаблоны встречаются не только в программировании. Например, зайдите в папку, куда мы устанавливали программное обеспечение, описанное в главе 2, и нажмите клавишу F3. Появится окно поиска файлов и папок. Наберите в строке ввода слово `php` и нажмите **Enter**. В результате выведется большой список файлов и папок, где встречается это слово. Вы только что написали шаблон.

Рассмотрим более сложный пример с использованием подстановочных символов. Наберите в строке поиска файлов и папок следующее:

```
php?
```

В нашем случае выведется папка с названием PHP5. Символ ? означает, что название (включая расширение) искомых файлов и папок должно начинаться со слова `php`, за которыми следует любой символ.

Приблизительно по такому же принципу происходит работа с регулярными выражениями в PHP.

Регулярные выражения POSIX

Литералы

Итак, самый простой способ задать шаблон – это использовать литералы. Рассмотрим пример: `is`.

В данном случае любая строка, где встречается сочетание символов `is` , будет удовлетворять этому шаблону. Пример:

```
Love is
List
Is worth nothing
```

Проверка соответствия строки определенному POSIX шаблону в PHP осуществляется с помощью функций `ereg()` и `eregi()` – листинг 15.1.

Листинг 15.1 ▼ Проверка соответствия строки POSIX шаблону

```
<html>
<head>
   <title> Проверка соответствия строки POSIX шаблону </title>
</head>
<body>
<?php
if ( ereg("is", "Love is") )
{
   echo "Строка соответствует шаблону";
}
else
{
   echo "Строка не соответствует шаблону";
}
?>
</body>
</html>
```

Если строка соответствует шаблону, то функция возвращает TRUE, иначе – FALSE. В данном случае выведется `Строка соответствует шаблону`, так как в строке `Love is` присутствует `is`.

Функция ereg() может содержать еще один необязательный параметр в виде массива передаваемого по ссылке (смотрите главу 7). В него записываются совпавшие комбинации (листинг 15.2).

Листинг 15.2 ▼ Особенности проверки соответствия строки POSIX шаблону

```
<html>
<head>
   <title> Особенности проверки соответствия строки POSIX шаблону </title>
</head>
<body>
<?php
if ( ereg("is", "List", $regs) )
{
   echo "Строка соответствует шаблону";
}
else
{
   echo "Строка не соответствует шаблону";
}
echo "<br>";
print_r ($regs);
?>
</body>
</html>
```

В результате значением элемента массива с индексом 0 будет строка is. На первый взгляд может показаться, что использование массива бессмысленно, так как мы сами задаем значение шаблона. Однако последующие примеры расставят все по своим местам.

Надо отметить, что рассмотренная функция учитывает регистры, в которых записываются символы (листинг 15.3).

Листинг 15.3 ▼ Учет регистра строки

```
<html>
<head>
   <title> Учет регистра строки </title>
</head>
<body>
<?php
if ( ereg("is", "Is worth nothing") )
{
   echo "Строка соответствует шаблону";
}
else
{
   echo "Строка не соответствует шаблону";
}
?>
</body>
</html>
```

В этом случае строка `Is worth nothing` не будет соответствовать шаблону. Если вас не интересует регистр символов, то следует применять функцию `eregi()` – листинг 15.4.

Листинг 15.4 ▼ Проверка соответствия строки POSIX шаблону без учета регистра

```
<html>
<head>
   <title> Проверка соответствия строки POSIX шаблону без учета регистра < title>
</head>
<body>
<?php
if ( eregi("is", "Is worth nothing") )
{
```

```
    echo "Строка соответствует шаблону";
}
else
{
    echo "Строка не соответствует шаблону";
}
?>
</body>
</html>
```

Теперь строка соответствует шаблону.

Метасимволы

Подобно подстановочным символам при поиске файлов и папок операционной системы Windows в регулярных выражениях POSIX используются метасимволы (листинг 15.5).

Листинг 15.5 ▼ Использование метасимволов

```
<html>
<head>
    <title> Использование метасимволов </title>
</head>
<body>
<?php
if ( eregi("php.", "php5 is the best", $regs) )
{
    echo "Строка соответствует шаблону";
}
else
{
    echo "Строка не соответствует шаблону";
```

```
}
echo "<br>";
print_r($regs); // выведет Array ( [0] => php5 )
?>
</body>
</html>
```

Приведенный шаблон содержит специальный символ (точку), означающий любой символ, кроме символа перевода строки. Другими словами, любая строка, содержащая символы php и сразу идущий за ним любой символ, будет удовлетворять этому регулярному выражению. В данном случае php5 is the best является такой строкой. Попробуйте вместо нее написать просто php и посмотрите результат.

Обратите внимание на массив $regs. В отличие от примера с литералами мы заранее не знали, какой символ будет найден вместо точки. Именно в этих случаях применение необязательного параметра имеет смысл.

Метасимволы ^ и $ часто называют якорями, так как они привязывают слова к началу или концу строки. Например, шаблону ^php будет соответствовать любая строка, начинающаяся с символов php (php5, php is the best и т.д.), а шаблону php$ – заканчивающаяся на php (I like php, tophp и т.д.). Заметьте, что строка php будет удовлетворять обоим шаблонам.

Символ вертикальной черты | выполняет функцию оператора или. Например, шаблону php|perl удовлетворяют строки, содержащие либо символы php, либо perl (perl and php, php-приложение, perlovka и др.).

Для выделения отдельных частей шаблона в группы используют круглые скобки (). Например, шаблону php|perl$, будут соответствовать строки, содержащие символы php или оканчивающиеся на perl. Однако если поставить круглые скобки (php|perl)$, знак доллара будет относиться к обоим словам, то есть строки должны оканчиваться на php или perl.

Стоит обратить внимание на то, что помимо обычных символов, в шаблонах можно указывать специальные символы, такие как табуляция

(\t), перевод строки (\n), возврат каретки (\r) и др. В случае если необходимый символ совпадает с метасимволом, то его надо экранировать с помощью обратной косой черты. Пример:

- \.$ – строка должна завершаться точкой;
- ^\^ – строка должна начинаться со знака ^;
- \\ – строка должна содержать символ обратной косой черты.

Классы символов

При рассмотрении метасимвола точки мы говорили о любом символе, идущем в строке. Однако часто требуется выделить определенное множество значений для него. Например, после слова php должна следовать цифра от 3 до 5. В таких случаях применяют классы символов.

Чтобы создать класс символов, нужно записать в квадратные скобки все доступные значения или их диапазон. Пример:

- [0-9] – символ является цифрой;
- [a-z] – символ является строчной буквой английского алфавита;
- [A-Z] – символ является заглавной буквой английского алфавита;
- [a-zA-Z] – символ является любой буквой английского алфавита;
- [abc] – символ является любой из букв a, b или c.

Рассмотрим несколько примеров:

- php[3-5] – строка должна содержать php3, php4 или php5;
- [0-9][a-z] – строка содержит сочетание цифры и символа;
- ^[0-5][0-9]$ – строка должна состоять из двух цифр (от 00 до 59).

Обратите внимание на последний пример. Символ ^ указывает на то, что строка должна начинаться с цифры из диапазона от 0 до 5, за которой следует еще одна любая цифра. При этом знак доллара $ указывает, что эта цифра должна завершать строку.

Ранее говорилось о том, что символ ^ указывает на начало строки. Помимо этого, его можно применять в классах символов, но он уже будет нести другую смысловую нагрузку.

Рассмотрим пример:

- [^0–9] – строка должна содержать хотя бы один символ отличный от цифры;
- ^[^a-zA–Z] – строка должна начинаться с любого символа, кроме буквы английского алфавита;
- [^aAeEiIoOuU] – строка не должна состоять целиком из гласных букв.

В регулярных выражениях типа POSIX для удобства были введены встроенные классы символов (табл. 17.1).

Таблица 17.1 ▼ Встроенные классы символов

Обозначение	Описание
`[[:alnum:]]`	Любая буква английского алфавита или цифра
`[[:alpha:]]`	Любая буква ([a–zA–Z])
`[[:blank:]]`	Пробельный символ или символ с кодом 0 и 255
`[[:digit:]]`	Любая цифра ([0–9])
`[[:lower:]]`	Любая строчная буква английского алфавита ([a–z])
`[[:upper:]]`	Любая заглавная буква английского алфавита ([A-Z])
`[[:punct:]]`	Любой знак пунктуации
`[[:space:]]`	Любой пробельный символ
`[[:xdigit:]]`	Любая шестнадцатеричная цифра ([0–9a–fA–F])

Квантификаторы

Представьте себе случай, когда нужно отобрать строки, в которых имеется комбинация символов `ccc`. Первое, что приходит на ум, – это применить шаблон, состоящий из одних литерал (`ccc`). Однако для решения подобных задач в регулярных выражениях типа POSIX применяют специальные символы, которые называются квантификаторы. Например, в нашем случае шаблон будет выглядеть следующим образом: `c{3}`. В фигурных скобках можно указывать не только конкретное количество

повторений, но и диапазон. Например, шаблон c{2,3} будет соответствовать строкам, где содержатся сочетания символов cc или ccc . Если второй параметр опустить (c{2,}), то строка может содержать два и более символа c, идущих подряд.

На первый взгляд может показаться, что шаблоны c{2}, c{2,3} и c{2,} эквивалентны, так как если строка содержит два или более символа c, идущих подряд, то она будет соответствовать всем этим шаблонам. Однако не стоит спешить с выводами, различие все-таки есть. Рассмотрим листинг 15.6 в качестве примера.

Листинг 15.6 ▾ Использование квантификаторов

```
<html>
<head>
   <title> Использование квантификаторов </title>
</head>
<body>
<?php
eregi("c{2}" , "qwccccccqw", $regs);
eregi("c{2,3}" , "qwccccccqw", $regs);
eregi("c{2,}" , "qwccccccqw", $regs);
// выведет "cc"
echo "<br>".$regs[0];
// выведет "ccc"
echo "<br>".$regs[1];
// выведет "ccccccc"
echo "<br>".$regs[2];
?>
</body>
</html>
```

Здесь мы проверяем соответствие шаблонам строки qwccccccqw и записываем совпавшие комбинации в массив $regs. В результате значения

элементов массива оказались cc, ccc и ccccccc. В этом и состоит отличие этих шаблонов.

Помимо фигурных скобок существует еще три квантификатора:

- c+ – строка должна содержать один или более символов c , идущих подряд (c{1,});
- c* – строка должна содержать ноль или более символов c , идущих подряд (c{0,});
- c? – строка должна содержать ноль или один символ c (c{0,1}).

Замена по шаблону

В PHP имеется возможность не только проверять соответствие строки определенному шаблону, но и заменять совпавшие с ним части строки. Для этого применяют функции ereg_replace() и eregi_replace() – листинг 15.7.

Листинг 15.7 ▼ Замена по шаблону

```
<html>
<head>
   <title> Замена по шаблону </title>
</head>
<body>
<?php
$str = "His name is Bob. Bob is my friend.";
$patt = "Bob";
$str_repl = "Bill";
// выводит "His name is Bill. Bill is my friend."
echo ereg_replace($patt, $str_repl, $str);
?>
</body>
</html>
```

В данном примере в строке $str ищутся все подстроки, удовлетворяющие шаблону $patt, и заменяются строкой $str_replace. В результате функция ereg_replace() возвращает строку со всеми заменами. В нашем случае это будет His name is Bill. Bill is my friend. Если функция не находит подстроки, удовлетворяющие шаблону, то она возвращает исходную строку.

Стоит обратить внимание на особенность работы функции ereg_replace() в том случае, если в задаваемом шаблоне используются скобки (листинг 15.8).

Листинг 15.8 ▾ Особенности замены по шаблону

```
<html>
<head>
   <title> Особенности замены по шаблону </title>
</head>
<body>
<?php
$str = "Время 14:07:23";
$patt = "([0-1][0-9]|[0-2][0-4]):([0-5][0-9]):([0-5][0-9])";
$str_repl = "\\1 часов \\2 минут \\3 секунд";
// выводит Время 14 часов 07 минут 23 секунд
echo eregi_replace($patt, $str_repl, $str);
?>
</body>
</html>
```

В строке $str_repl используются специальные сочетания символов (\\1, \\2 и \\3). В результате выполнения функции ereg_replace() вместо них вставляется подстрока, которая удовлетворяет шаблону, находящемуся в скобках с соответствующим номером. Всего возможно использовать цифры от 0 до 9. При этом \\0 будет соответствовать всей строке шаблона – листинг 15.9.

Листинг 15.9 ▼ Замены по шаблону без учета регистра

```
<html>
<head>
   <title> Замены по шаблону без учета регистра </title>
</head>
<body>
<?php
$str = "His name is Bon.";
$patt = "Bon";
$str_repl = "\\0-\\0";
// выводит "His name is Bon-Bon"
echo ereg_replace($patt, $str_repl, $str);
?>
</body>
</html>
```

Как вы, наверное, догадались, функция `eregi_replace()` идентична `ereg_replace()`, но не чувствительна к регистру.

Примеры регулярных выражений

Теперь вы знакомы с основными составляющими синтаксиса регулярных выражений типа POSIX. Однако научиться использовать их вместе – это самая важная задача.

Обычно начинающие программисты пытаются создать некий универсальный шаблон, который будет удовлетворять всем заданным условиям. Однако это может очень осложнить простую, на первый взгляд, задачу. При проектировании регулярных выражений иногда лучше получить приблизительную структуру, чем добиваться абсолютной точности (например, шаблон для проверки e-mail адреса в идеале должен занимать около трех страниц текста). Приведем несколько примеров шаблонов, которые можно применять на практике. Внимательно разберите каждый из них, уяснив принцип их построения.

Идентификатор

В главе 4 мы описывали предъявляемые условия к названиям переменных. Теперь рассмотрим шаблон, задающий все эти условия:

```
^\\$[_a-zA-Z][_0-9a-zA-Z]*
```

Гиперссылка

Для поиска в тексте гиперссылок можно использовать следующий шаблон:

```
<a href=[^>]+>[^<]+</a>
```

E-mail адрес

Представьте, что на Web-сайте имеется форма, где пользователь вводит свой e-mail адрес. Вашему вниманию предлагается два примера регулярных выражений:

- ^.+@.+\\..+$ – этот шаблон предъявляет к строке очень мягкие условия. Его упрощенная схема имеет вид: что угодно@что угодно.что угодно;
- ^[a-z0-9\._-]+@[a-z0-9\._-]+\.[a-z]{2,4}\$ – если вам нужна более строгая проверка, то можно применить этот шаблон.

Заключение к главе

В этой главе мы познакомились с одной из самых сложных тем – регулярные выражения. Умение использовать их в различных ситуациях поможет вам решить многие Web-задачи. На первом этапе лучше всего разобрать уже имеющиеся шаблоны, а затем на их основе строить собственные. Экспериментируйте, и быть может именно ваше решение будет самым лучшим.

Работа с Cookies

Обычно начинающие программисты очень редко используют понятие Cookies, считая его очень сложным. Однако, прочитав эту небольшую главу, вы научитесь создавать, удалять, редактировать Cookies, а самое главное – их применять. Итак, приступим.

Cookies – это...

Cookies – это «печенья». Несложно догадаться, почему в любой русской литературе по программированию это слово не переводят. Говоря серьезно, под этим понятием подразумевают небольшие фрагменты данных, которые можно сохранить в настройках браузера клиента.

В основном Cookies применяются для запоминания информации о пользователе, например, его логина и пароля. На самом деле это очень удобно: не нужно заводить отдельную базу данных, информация поступает намного быстрее и именно та, которая нам требуется. Однако не стоит надеяться только на Cookies, так как есть браузеры, которые их просто не поддерживают, к тому же сам пользователь может отключить их использование. Поэтому не стоит ставить глобальных задач перед Cookies, они просто могут облегчить вам жизнь.

По умолчанию браузер Internet Explorer разрешает использование Cookies. Убедиться в этом можно следующем образом. Выберете пункт меню **Свойства обозревателя** (**Сервис ➤ Свойства обозревателя**). В открывшемся окне найдите закладку **Безопасность**. Нажмите кнопку **Другой...** – откроется еще одно окно **Правила безопасности**. Среди настроек этого окна вы найдете разрешения для Cookie (смотрите рис. 16.1).

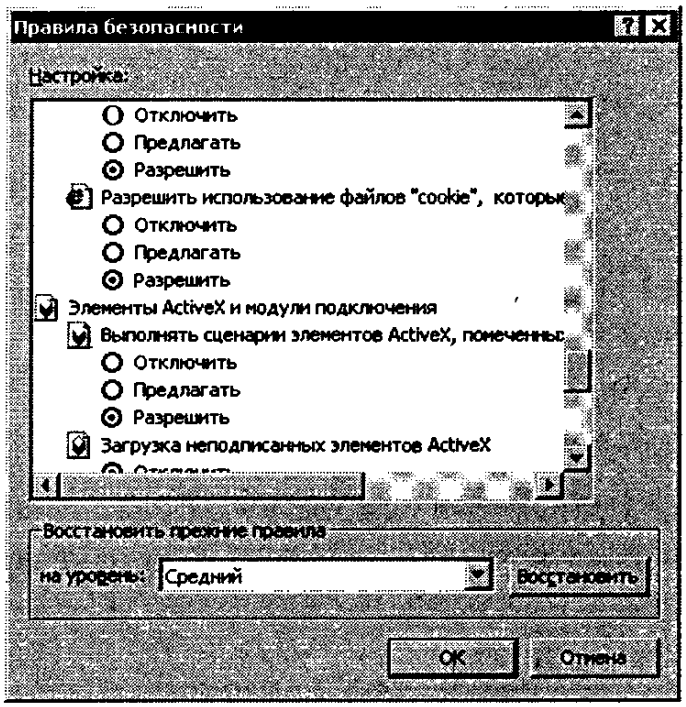

Рис. 16.1 ▼ Разрешение Cookies в браузере Internet Explorer

Создание Cookies

Cookie создается с помощью стандартной функции `setcookie()`. Она содержит всего один обязательный параметр в виде строки, которая задает имя Cookie (листинг 16.1).

Листинг 16.1 ▼ Создание Cookie

```
<?php

// создание Cookie
```

```
setcookie("name");
?>
</body>
</html>
```

Однако использование приведенной функции без дополнительных параметров вряд ли может быть эффективным, так как значением, созданным Cookie (с названием name), будет пустая строка. Другое значение можно задать так, как показано в листинге 16.2.

Листинг 16.2 ▼ Создание Cookie со значением

```
<?php
// создание Cookie
setcookie("name", "value");
?>
```

В этой программе создается Cookie с именем name и значением value. Но и в таком виде функция setcookie() редко используется, так как Cookie уничтожается сразу после закрытия браузера пользователем. Задать время жизни Cookies можно с помощью еще одного дополнительного параметра (листинг 16.3).

Листинг 16.3 ▼ Создание Cookie со значением и временем жизни

```
<?php
// создание Cookie
setcookie("name", "value", time() + 86400);
?>
```

Как вы, наверное, догадались, время задается в секундах. В данном случае Cookie исчезнет через сутки после создания.

Чтение из Cookies

Итак, вы создали Cookie и хотите получить при следующих сеансах ее значение. Для этого существует специальный массив $_COOKIE, элементы

которого содержат имя и значение Cookie. Однако стоит помнить, что запись в этот массив не происходит при первом сеансе (листинг 16.4).

Листинг 16.4 ▼ Вывод Cookie

```
<?php
// создание Cookie
setcookie("Number", "123", time() + 86400);
echo $_COOKIE["Number"]; // ничего не выводит
?>
```

В этом случае сначала происходит запись в массив $_COOKIE, а затем выполняется функция setcookie(). Этим объясняется отсутствие элемента в массиве $_COOKIE при первом вызове, который соответствует Cookie с именем Number. При повторном вызове такой элемент появляется.

Помимо массива $_COOKIE, вы можете использовать также массив $HTTP_COOKIE_VARS, который разработчики сохранили PHP для совместимости с новыми версиями.

Мы уже говорили, что в PHP 5 по умолчанию параметр конфигурационного файла php.ini register_globals имеет значение off. В этом случае PHP автоматически не создает переменную с именем, соответствующим Cookie.

Удаление Cookies

Напоследок рассмотрим способы удаления Cookies. Как мы уже говорили в этой главе, если время жизни Cookie не задано в качестве входного параметра функции setcookie(), она исчезнет сразу после закрытия браузера. Например, это можно использовать в случае регистрации пользователя посредством ввода логина и пароля. Они сохраняются в самом начале в настройках браузера в виде Cookies, и пользователь может без проблем загружать страницы, которые соответствуют им(например, при проверке электронной почты). Но когда пользователь зак-

рывает браузер, Cookies с логином и паролем удаляются, и при следующей загрузке их надо вводить снова.

Еще один способ удалить Cookie – это указать ее значение в виде пустой строки (листинг 16.5).

Листинг 16.5 ▼ Удаление Cookie

```
<?php
// удаление Cookie с именем "name"
setcookie("name", "");
?>
```

Однако этот способ иногда не работает, поэтому обычно указывают время жизни в прошлом (листинг 16.6).

Листинг 16.6 ▼ Удаление Cookie

```
<?php
// удаление Cookie с именем "name"
setcookie("name", "", time() - 3600);
?>
```

Заключение к главе

Этой главой мы завершаем рассмотрение основных возможностей PHP при работе с внешними структурами данных. Тема Cookies оказалась последней, так как их грамотное использование требует от Web-программиста достаточного опыта. С другой стороны работать с Cookies, как вы поняли, достаточно легко.

Предметный указатель

Зольников Дмитрий Станиславович

PHP 5

Как самостоятельно создать сайт любой сложности

2-е издание, стереотипное

Главный редактор *Захаров И. М.*
editor-in-chief@ntpress.ru
Научный редактор *Чумаченко И. Н.*
Ответственный редактор *Мирошникова Е. Г.*
Верстка *Черникова О. В.*
Графика *Салимонов Р. В.*
Дизайн обложки *Клубничкин Д. Е.*

Издательство «НТ Пресс», 129085, Москва,
Звездный б-р, д. 21, стр. 1.

Издание осуществлено при техническом участии
ООО «Издательство АСТ»

Отпечатано в ОАО ордена Трудового Красного Знамени
«Чеховский полиграфический комбинат».
142300, г. Чехов Московской области,
тел./факс (501) 443-92-17, (272) 6-25-36.
E-mail:marketing@chpk.ru