Александр Мазуркевич
Дмитрий Еловой

PHP:

настольная книга программиста

МИНСК 000 «НОВОЕ ЗНАНИЕ» 2003

УДК 004.92
ББК 32.973.26-018.2
М13

Мазуркевич А.
МВ РНР: настольная книга программиста /Александр Мазуркевич, Дмитрий Еловой. — Мн.: Новое знание, 2003. — 480 с.: ил.
ISBN 985-475-014-0.

В удобной наглядной форме описаны все элементы PHP — популярного языка создания CGI-сценариев. Рассмотрены не только особенности синтаксиса языка, но и редактирование кода в программах EditPlus и UltraEdit, а также установка PHP и сервера Apache. Материал систематизирован таким образом, что читатель может использовать книгу и как учебник, и как справочник. Примеры, взятые из реальной практики Web-программирования, позволяют лучше усвоить теоретический материал.

Книга рассчитана на самую широкую аудиторию — не только на новичков, но и на опытных программистов.

УДК 004.92
ББК 32.973.26-018.2

Производственно-практическое издание

Мазуркевич Александр Михайлович
Еловой Дмитрий Сергеевич

PHP:
настольная книга программиста

Ведущий редактор *А.В. Жвалевский*
Редактор *Е.С. Каляева*
Корректор *Т.К. Хваль*
Художник обложки *С.В. Ковалевский*
Компьютерная верстка *Е.В. Криводубская*

Подписано в печать с готовых диапозитивов 10.03.2003. Формат 70x100 1/16.
Бумага газетная. Гарнитура Ньютон. Печать офсетная.
Усл. печ. л. 39. Уч.-изд. л. 24,6. Тираж 2600 экз. Заказ № 556.

Общество с ограниченной ответственностью «Новое знание».
ЛВ № 310 от 27.12.2002. Минск, ул. Академическая, д. 28, к. 112.
Почтовый адрес: 220050, Минск, а/я 79. Телефон/факс: (10-375-17) 211-10-33, 284-03-23.
Москва, ул. Маросейка, 10/1. Телефон (095) 921-67-21.
E-mail: nk@wnk.biz
http://wnk.biz

Республиканское унитарное предприятие «Издательство «Белорусский Дом печати».
220013, Минск, пр. Ф. Скорины, 79.

ISBN 985-475-014-0

Оглавление

Часть II. Программирование на PHP

Посвящение

Данная книга посвящается нашим родителям, Мазуркевичам Михаилу и Светлане, Еловым Ирине и Сергею, людям, которые вселили в нас силу, целеустремленность и желание.

На протяжении всей работы они поддерживали нас и мирились со многими трудностями. Наши родители смогли понять наши проблемы и обеспечить режим максимального благоприятствования в эти безумные месяцы.

Мазуркевич Александр,
Еловой Дмитрий.
Июнь 2002 г.

Благодарность

Мы хотели бы выразить свою признательность и благодарность тем людям, без которых эта книга не увидела бы свет:

- Научные редакторы — Андрей Золотарь и Сергей Коженов. Эти люди шаг за шагом досконально изучали первоначальную рукопись и вносили свои изменения.
- Редактор — Екатерина Каляева. Советы Екатерины Каляевой очень помогли нам при написании этой книги.
- Ведущий редактор — Жвалевский Андрей Валентинович. Этот человек изначально вселил в нас веру в успех, и благодаря ему эта книга получила такой облик. В случае каких-либо затруднений Андрей Валентинович выслушивал и давал нам подходящий совет. Он был не только ведущим редактором, но и своего рода учителем. Спасибо ему за это.

Александр М. Мазуркевич,
Дмитрий С. Еловой.

Предисловие

Итак, вы приступаете к чтению книги «PHP: настольная книга программиста». Если вы уже знакомы с такими языками, как C++, Java, Perl, то изучение книги пройдет для вас максимально быстро и без каких-либо сложностей. Но даже если вы не имеете ни малейшего представления об этих языках, данная книга поможет вам как можно глубже получить знания в области интернет-программирования, а в частности языка программирования PHP (Personal Home Page). Задача данного языка — позволить за предельно минимальное время создавать динамически сгенерированные страницы в Интернете. Разработчики Personal Home Page постарались создать такой язык, который был бы лаконичен и предельно точен в реализации той или иной поставленной задачи. С помощью PHP можно намного проще решить проблему, чем используя, допустим, Perl. Хотя, конечно же, все зависит от самого программиста и реализуемой задачи.

Следует уточнить один момент, особенно для тех, кто вообще никогда не сталкивался с программированием. Задачи, которые перед вами стоят, решаются так, как вы это видите. Как есть разные способы доехать до одного и того же места, есть несколько способов открыть панель управления в Windows, так есть и разные способы и методы программирования. С поставленной задачей можно справиться и так, как описано, и совсем по-другому. В нашей книге акцент будет делаться на простоту и логичность, а не на количество тех или иных способов реализации задуманного. Главное — понять принцип, все остальное придет с опытом.

Не исключено, что вы сможете придумать другой подход к решению тех или иных задач, которые будут описаны в данной книге. И это нормально, ведь сколько разработчиков, столько и решений.

PHP — это не только очень просто, это еще и увлекательно. К такому выводу приходишь, как только начинаешь знакомиться непосредственно с основными программами на этом языке. Только не подумайте, пожалуйста, что не успели освоиться с основными терминами, а тут сразу непонятные слова. По этому поводу хотелось бы добавить: в данной книге мы постарались применять упрощенные и понятные всем термины, чтобы не только профессионалы, но и начинающие в программировании могли прочесть эту книгу на «одном дыхании».

Рассудив логически, можно сделать вывод: в любом деле есть свои сложности. Только каждый преодолевает их по-своему. Хотелось бы посоветовать — не останавливайтесь на достигнутом. Только тогда, когда вы упорно и целеустремленно идете шаг за шагом к цели, вы ее достигнете. Поэтому придумывайте, изобретайте и не бойтесь экспериментировать, ведь лишь на практике возможно научиться профессионально программировать и достигать поставленной цели.

Мазуркевич Александр.
Июнь 2002г.

О чем эта книга

- В части I «Основные характеристики PHP» вы сможете изучить историю создания языка, а также самостоятельно освоить процесс установки модулей PHP на такие платформы, как Unix, Linux, Windows и т. д.
- В части II «Программирование на PHP» рассматриваются основные аспекты языка программирования. Эта часть необходима для того, чтобы заложить основы и направления реализации языка, так называемый фундамент для создания своих собственных программ. Также в этой части описаны классы и объекты, представляющие наибольшую сложность в изучении языка. Для начинающих программистов здесь есть параграфы о базовых типах данных и написании выражений.
- Часть III «Особенности реализации языка» рассказывает об использовании широко известных функций, подробно разъясняет и демонстрирует некоторые из передовых особенностей программирования. В этой части «разоблачается» кажущаяся сложность многих прогрессивных концепций и затрагиваются некоторые вопросы, связанные с базами данных.
- Часть IV «PHP-функции» позволит систематизировать полученные знания в области PHP-программирования. Изложенная информация охватывает наиболее часто используемые функции, также приводится практическое применение каждой из них. Постарайтесь как можно подробнее изучить данную часть. Это поможет вам понять не только принципы программирования, но и способы построения больших программ, их реализацию, а также даст возможность составлять наиболее практичные и логически последовательные коды.
- Часть V «Приложения» содержит перечень информации, необходимой при изучении PHP.

Принятые выделения

Чтобы упростить восприятие материала, в книге используются следующие условные обозначения:

Моноширинным шрифтом набраны все листинги, приведенные в тексте зарезервированные слова языка, пути к файлам и значения переменных.

Полужирным шрифтом отмечены названия элементов интерфейса (окон, пунктов, меню, опций).

Шрифтом без засечек выделены утверждения, на которые следует обратить внимание, а также советы.

Введение

Развитие современной компьютерной техники и внедрение новейших технологий положили начало нового направления жизни на Земле. За довольно короткий промежуток времени развития микроэлектроники и кибернетики произошло очень много изменений.

Прогрессивное развитие техники вызвало появление новых программных продуктов. С каждым годом внедряется все большее и большее количество языков программирования. Все они ориентированы прежде всего на целевую аудиторию.

Развиваются не только компьютеры, но и сети. Если еще несколько десятков лет назад Интернет представлял собой небольшую частную сеть, то теперь это гигантская система взаимосвязанных компьютеров, без которой, возможно, мы не сможем представить себе жизнь.

Интернет — это не только «прохладный бассейн», в котором так комфортно чувствуют себя весьма «перегревшиеся» люди, но и место, где можно «потрогать» практически любую горячую тему, не опасаясь при этом обжечься.

Люди постепенно перестают пользоваться обычной почтой, так как, посылая письма по Сети, они экономят не только время, но и средства. Электронная почта получила такое широкое применение, что совсем скоро каждый человек будет иметь свой собственный электронный ящик.

Раньше Web-страница выглядела как обычный текст, без какой-либо графики. Теперь же все совсем иначе: сайты насыщены иллюстрациями и имеют непростую структуру. Решать сложные задачи и призваны такие языки программирования, как C++, Perl, PHP и др.

Эта книга была задумана как полный справочник по PHP, руководство для профессионалов и начинающих программистов, которые хотели бы освоить этот язык и, возможно, связать с ним свою профессиональную деятельность в Сети.

На протяжении всей книги подробно описывается PHP и его функции. В конце каждой главы приведены простые примеры, чтобы на начальном этапе вы смогли без проблем понять, о чем идет речь. По мере усложнения материала примеры также усложняются, однако с учетом приобретенных знаний освоить их будет нетрудно. Мы постарались сделать так, чтобы наша книга подходила и для последовательного, и для произвольного изучения. Если вам необходима какая-либо конкретная тема, вы всегда можете обратиться к той или иной главе либо части и пополнить свои знания.

Надеемся, что наша книга будет полезна в работе.

Часть I

Основные характеристики PHP

Глава 1

Введение в PHP

1.1. Из истории PHP

Допустим, вы вышли за рамки статических Web-страниц и вам требуется обрабатывать HTML-формы. Или вы хотите создать Web-страничку, которая осуществляла бы обработку введенных вами данных, обращаясь непосредственно к базе данных. А возможно, вам надо запрограммировать электронный магазин, опрос посетителей вашего сайта, разнообразные счетчики посещений — язык PHP поможет вам реализовать все эти и многие другие задачи.

Возникает вопрос, а почему именно PHP, ведь множество программ, работающих через CGI (Common Gateway Interface — общий шлюзовой интерфейс, являющийся стандартом, описывающим HTTP-приложения), написаны на языке Perl, C/C++, Fortran, TCL, Unix Shell, Visual Basic, Apple Script и других подобных языках. Сервер передает запросы пользователя CGI-программе, которая их обрабатывает и возвращает результат своей работы на экран пользователя. В результате пользователь получает динамическую информацию, которая может изменяться под влиянием различных факторов. Разработчики всегда стремились создавать языки программирования, оптимально подходящие для всех условий работы в той или иной среде. Изучая в дальнейшем данную книгу, вы сможете понять, почему же создатель PHP — известный программист Расмус Лердорф (Rasmus Lerdorf) — начал разработку языка программирования, исполняющегося на стороне сервера.

Что касается, например, популярного языка программирования Perl, то это — очень хороший, мощный язык, но слишком велики системные издержки во время вызова программы на каждый запрос страницы, особенно в Windows. Поэтому в 1994 *г.* появился совершенно новый язык, который и получил название PHP (Personal Home Page).

Сам язык программирования был написан буквально за день в промежутки между деловыми встречами Расмуса Лердорфа. Сначала это была простая, невзрачная CGI-оболочка, написанная на языке Perl, которая служила исключительно для специфических целей.

С течением времени при эксплуатации выяснилось, что оболочка обладает маленькой производительностью, и создателю ничего не оставалось, как переписать ее заново, исправив существующие ошибки. Узел Сети, на котором находилось резюме, был чрезвычайно перегружен, и чтобы избавиться от значительных непроизвольных затрат, он переписал оболочку на языке C. Это позволило значительно увеличить скорость работы PHP. Пользователи сервера, где располагался сайт с первой версией PHP, попросили себе такой же инструмент. Затем, как неизбежно это случается, пользователи начали просить о большем количестве функций. Расмус Лердорф старался выполнить все пожелания пользователей и в результате наполовину собрал дистрибутив наряду с документацией и часто задаваемыми вопросами. И хоть он не предполагал, что кто-то другой будет пользоваться этим языком, довольно быстро PHP перерос в самостоятельный проект, и в начале 1995 г. вышла первая известная версия продукта. Имя этого первого пакета было Personal Home Page Tools (средства для персональной домашней страницы). В то время PHP обладал более чем скромными возможностями. Он имел простейший анализатор кода, который понимал несколько специальных команд, а также разные утилиты для использования на домашней странице, необходимые для построения таких полезных вещей, как гостевая книга, счетчик, чат, системы статистики и т. д. В то же время Расмус Лердорф начал заниматься базами данных, чему положило начало написание инструмента для реализации SQL-запросов в Web-страницах. Это была отдельная CGI-оболочка, которая анализировала запросы SQL и облегчала создание форм и таблиц, основанных на этих запросах. Этот инструмент был назван Form Interpreter (FI — интерпретатор форм). После того как были добавлены функции работы с базами данных, вышла вторая версия продукта.

Различие между PHP и FI незначительное. И PHP, и FI созданы из одного и того же исходного текста. Когда происходит формирование пакета данных без какой-либо регистрации доступа или поддержки ограничения доступа, вызывается выполняемый модуль FI. Когда происходит формирование документов при помощи вышеперечисленных опций, используется PHP.

Сейчас PHP — это быстро развивающееся средство программирования, работающее на очень многих серверах в Интернете (рис. 1.1). Как средство разработки Web-приложений PHP сейчас является одним из самых популярных вместе с ASP, FrontPage и mod_perl. Благодаря этому языку появляется возможность легко создавать динамические сайты. Файлы, созданные таким образом, хранятся и обрабатываются на сервере. Когда посетитель запрашивает документ с PHP, скрипт обрабатывается не браузером посетителя, как, например, JavaScript, а сервером, и посетителю передаются только результаты работы. Точно так же работает CGI-программа, написанная на C или Perl. Но в отличие от CGI код PHP-программы можно встраивать в любое место HTML-страницы, что, конечно, является основным преимуществом перед CGI. Кроме того, сам язык очень прост для изучения и не требует каких-либо специфических знаний.

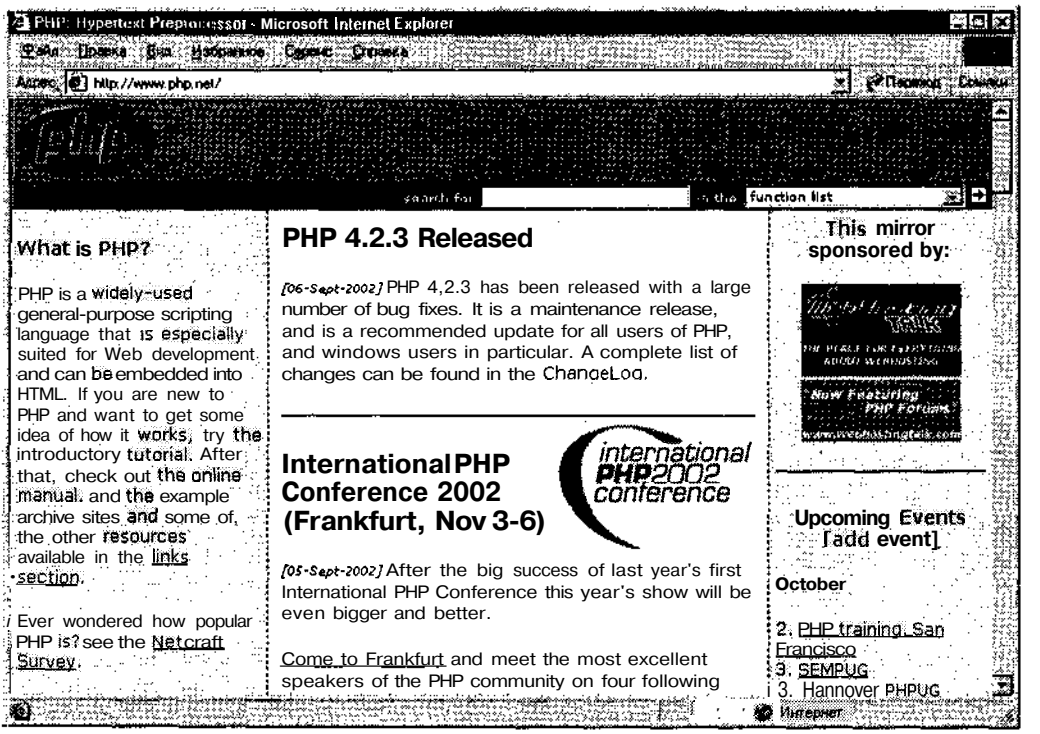


Рис. 1.1. Сайт www.php.net

PHP можно разделить на язык и библиотеку функций. Существует большое количество инструментальных средств для PHP, интерфейсы ко всем популярным СУБД (системам управления базами данных), почтовым протоколам, разделяемой памяти, графическим файлам, архивам и множество других инструментов, с которыми вы сможете познакомиться в процессе изучения данной книги.

Язык настолько прост в использовании, что совсем скоро каждый программист в области Интернета сможет овладеть и применять полученные знания для решения все большего числа поставленных задач. Уже в наши дни PHP используется более чем на 2,5 млн сайтов.

1.2. Преимущества и недостатки PHP

Рассмотрим все особенности PHP и выясним, почему многие считают, что PHP — именно то, что нужно.

Преимущества PHP

- PHP — скрипт-язык, который непосредственно встраивается в HTML-код и выполняется сервером. Приведем пример, демонстрирующий саму работу и способ встраивания PHP-скрипта в документ.

Пример 1.1. Встраивание PHP в документ

```
<html>
<head>
<title>Listing 1.1</title>
</head>
<body>
<?php echo "Hello! My script works good!"; ?>
</body>
</html>
```

Опишем каждую строку данного документа, чтобы у вас не возникало никаких вопросов. Открывающий тег <html> показывает браузеру, что данный файл содержит HTML-страницу. Весь документ заключен в контейнер HTML. Текст, помещенный вне этого документа, может игнорироваться браузером. <title>Listing 1.1</title> задает название документа, которое может быть любым, по желанию программиста. Эти теги располагаются между тегами <head> </head>, которые содержат информацию о текущем документе, такую как заголовок, ключевое слово и т. п. Браузеры не отображают информацию, помещенную в контейнер <head>, однако могут ее использовать для каких-либо иных целей. Контейнер <body> охватывает все содержимое документа, которое должно быть представлено пользователю. Именно в этой части вставлен PHP-скрипт. Функция echo"Hello! My script works good!"; выведет в окно браузера фразу, заключенную между двойными кавычками. После выполнения скрипта получим страницу с надписью (рис. 1.2).

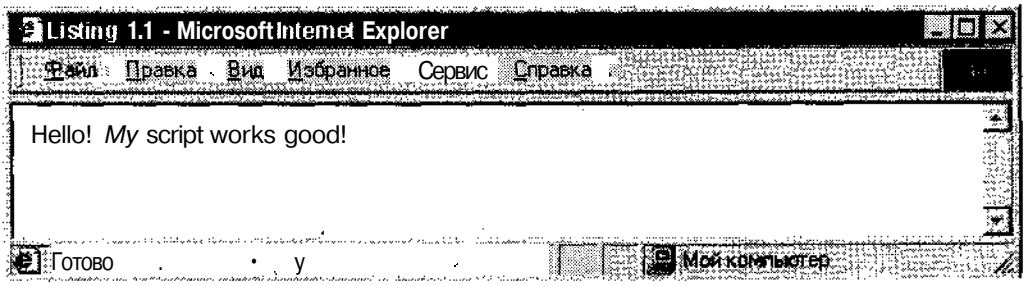


Рис. 1.2. Результат выполнения программы

ВНИМАНИЕ

Каждый оператор в PHP отделяется точкой с запятой. Это очень важно, так как в противном случае браузер выдаст сообщение об ошибке, указывающее номер строки, в которой она произошла.

- Основное отличие PHP от CGI-скриптов, написанных на других языках, типа Perl или C++, — это то, что в CGI-программах вы сами пишете выводимый HTML-код, а используя PHP, встраиваете свою программу в готовую HTML-страницу при помощи открывающего и закрывающего тегов (см. пример 1.1 — <?php и ?>).
- Отличие PHP от JavaScript состоит в том, что PHP-скрипт выполняется на сервере, а клиенту передается результат работы, тогда как JavaScript-код полностью передается на клиентскую машину и только там выполняется.
- Любители Internet Information Server найдут, что PHP очень похож на Active Server Pages (ASP), а энтузиасты Java скажут, что PHP похож на Java Server Pages (JSP). Все три языка позволяют размещать код, выполняемый на Web-сервере, внутри HTML-страниц.
- В PHP включена поддержка многих баз данных, что делает написание Web-приложений с использованием БД до невозможности простым.

Вот неполный перечень поддерживаемых БД:

Adabas D, InterBase, Solid, dBase, mSQL, Sybase, PostgreSQL, Empress, MySQL, Velocis, FilePro, Oracle, Unix dbm, Informix.

СОВЕТ

Приведенный список далеко не полный. Это говорит о том, что сам язык ориентирован на работу с базами данных. Он имеет достаточное количество функций для реализации поставленных задач. Поэтому, если у вас возникнет вопрос, что выбрать — Perl, PHP или C при реализации той или иной задачи, связанной с базами данных, лучший выбор — PHP.

- В PHP есть функции для работы с протоколами IMAP, SNMP, NNTP, POP3 и даже HTTP, а также имеется возможность работать с сокетами (sockets) и общаться по другим протоколам.

Недостатки PHP 3

Как правило, медаль имеет две стороны. В данной книге мы бы хотели подойти к PHP не только с положительной стороны, но и с критической. Даже самые лучшие системы имеют свои недостатки, которые со временем устраняются. И это нормально, нет необходимости считать ту или иную операционную систему несостоявшейся, если только в какой-либо функции при написании программы не было предусмотрено той или иной ситуации. Поэтому хотелось бы поговорить и об отрицательных качествах PHP.

- Основным недостатком PHP 3, является то, что по своей идеологии PHP изначально был ориентирован на написание небольших скриптов. Несмотря на то что ядро несколько раз переписывалось, PHP 3 не пригоден для использования в сложных проектах — при исполнении больших скриптов производительность системы резко падает. Однако этот недостаток ликвидирован в PHP 4.
- PHP является псевдокомпилируемым языком и вследствие этого не может сравниться по скорости с компилируемым C. Однако при написании небольших

программ, когда весь проект состоит из многих небольших страниц с кодом, вступают в силу накладные расходы на загрузку в память и вызов CGI-программы, написанной на C.

- Не такая большая база готовых модулей, как, например, CPAN у Perl. С этим ничего нельзя поделать — это дело времени. В PHP 4 разработчики предусмотрели специальный репозиторий, аналогичный CPAN, и очень скоро будет написано достаточное количество модулей для его наполнения.

Заключение

В этой главе было подробно описано, почему PHP является наиболее перспективным языком и в скором времени займет лидирующую позицию среди подобных языков программирования. Когда посетитель обращается к вашей страничке, PHP обрабатывает встроенные в нее команды, выдает результат браузеру пользователя — точно так же, как это делает CGI-программа, написанная на C или Perl. Только в отличие от последней PHP имеет ряд преимуществ:

- создание скриптов PHP значительно проще, чем создание скриптов на других языках;
- для решения разных специфических задач не нужно писать и отлаживать многочисленные маленькие CGI-программы, что сводит к минимуму время доступа к вашим страницам, а также продолжительность разработки страниц и сайта в целом.

Вместе с тем PHP обладает огромным набором функций и большой гибкостью, которые могут быть значительно расширены с помощью дополнительных внешних библиотек. Вы можете управлять доступом к вашим страницам, создавать и обрабатывать базы данных любой сложности, генерировать изображения или PDF-документы и т. д.

Глава 2
Установка PHP

В первой главе вы получили начальное представление о PHP, изучив историю его возникновения и некоторые отличия от других языков программирования. Теперь рассмотрим наиболее распространенные способы установки PHP.

Невозможно начать изучать программирование, не изучив и не поняв принципы установки модулей. Установка PHP является весьма легким занятием, особенно, если вы когда-нибудь настраивали сервер. В данной главе дается подробное описание всех принципов установки PHP на самые распространенные платформы.

PHP можно установить в двух вариантах: как отдельный интерпретатор, работающий через интерфейс CGI, или как модуль Web-сервера, встроенный в сам сервер.

Последний случай является наиболее эффективным, так как становятся актуальными все преимущества PHP.

Если пакет устанавливается в виде модуля Web-сервера, то это означает, что функциональные возможности PHP объединены с функциональными возможностями Web-сервера в одной программе. Также можно привести ряд преимуществ, которые могли бы повлиять на выбор способа установки. Выполнение происходит намного быстрее, чем при помощи обычных программ CGI, так как CGI-программа не запускается, а скрипт-код в HTML-файлах выполняется непосредственно процессом Web-сервера.

Из аналогичных, встроенных в сервер, программных средств хорошо известны SSI, `mod_perl`, ASP. Ближайшим аналогом PHP является ASP, но технология ASP не прижилась в мире Unix/Apache, где простой, удобный и быстрый язык PHP постепенно выходит в лидеры.

Из этой главы вы узнаете:

- как происходит установка PHP на системы Unix;
- как происходит установка PHP на платформы Unix/Linux;
- как происходит установка на системы Windows9x/Me/NT/2000;
- как происходит установка расширений функциональных возможностей Windows;
- как происходит проверка работоспособности PHP после установки.

2.1. Установка на системы Unix

Данный раздел поможет вам полностью изучить принципы установки PHP на операционную систему Unix.

Если у вас нет никакого опыта работы с Unix, вы можете попросить кого-либо, имеющего хотя бы немного знаний по Unix, помочь вам с установкой модуля. Процесс инсталляции настолько прост, насколько это возможно, но так как программное обеспечение сильно различается и зависит от ряда компонентов системы, установка не всегда проходит гладко на разных системах.

Имеется несколько способов установки PHP для платформы Unix — включающий компиляцию и конфигурирование или с использованием пакетных методов. В этой книге основной упор делается на процесс компиляции и конфигурирования PHP.

Исходный процесс установки и конфигурирования PHP управляется с помощью параметров командной строки скрипта `configure`. На этой странице описано использование распространенных параметров, но, кроме них, имеются и другие параметры, с которыми вы можете поэкспериментировать. Полное описание параметров конфигурации — в приложении А данной книги. Имеются следующие способы установки:

- установка как модуля Apache;
- установка как модуля fhttpd;

- для использования AOLServer, NSAPI, phttpd, Pi3Web, Roxen, thttpd, or Zeus;
- как исполняемый модуль CGI.

Как было сказано ранее, PHP может быть установлен несколькими способами, но в нашей книге мы не будем рассматривать все способы, а постараемся дать подробное описание наиболее популярных способов, одним из которых и является установка PHP как модуля Apache.

При установке можно выбрать аргументы для добавления в `configure` на строке 8 из полного списка опций конфигурации (см. приложение А). Обратите внимание, что вместо соответсвующей версии стоят `'xxx'`. При установке необходимо просто заменить `'xxx'` на корректное значение версии Apache или PHP.

```
1. gunzip apache_xxx.tar.gz
2. tar -xvf apache_xxx.tar
3. gunzip php-xxx.tar.gz
4. tar -xvf php-xxx.tar
5. cd apache_xxx
6. ./configure-prefix=/www -enable-module=so
7. make
8. make install
9. cd . ./php-xxx
10. ./configure-with-mysql - with-apxs=/www/bin/apxs
11. make
12. make install
```

Если вы решите изменить опции конфигурации после инсталляции, вам нужно будет только повторить последние три шага. Необходимо перезагрузить Apache, чтобы новый модуль заработал. Переустановка Apache не нужна.

```
13. cp php.ini-dist /usr/local/lib/php.ini
```

Вы можете редактировать ваш INI-файл для установки опций PHP. Если вы предпочитаете иметь этот файл в другом месте, используйте `-with-config-file-path=/path` начиная со строки 8.

14. Отредактируйте `httpd.conf` или `srm.conf` и проверьте, что эти строки имеются и не закомментированы:

```
AddType application/x-httpd-php .php
LoadModule php4_module libexec/libphp4.so
```

Вы можете выбрать здесь любое расширение файлов. Мы рекомендуем .php. Можете включить даже .html, а .php3 можно добавить для обеспечения обратной совместимости.

Путь с правой стороны оператора `LoadModule` обязан указывать на PHP-модуль на вашей системе. Выше указанный оператор корректен для предыдущих шагов.

15. Используйте вашу обычную процедуру старта сервера Apache. Если он уже запущен, то необходимо остановить его и загрузить сервер заново.

В зависимости от варианта установки Apache и вида Unix есть много вариантов остановки и перезагрузки сервера. Ниже даны типичные строки, используемые для перезагрузки сервера, для различных установок `apache/unix`. Вы должны заменить /path/to/ на путь к этим приложениям на вашей системе.

1. Разные варианты Linux и SysV:

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

2. **Использование скриптов** `apachectl`:

```
/path/to/apachectl stop
/path/to/apachectl start
```

3. `httpdctl` и `httpsdctl` **(с использованием OpenSSL) аналогично** `apachectl`:

```
/path/to/httpsdctl stop
/path/to/httpsdctl start
```

4. Используя `mod_ssl` или иной SSL-сервер, вы можете вручную остановить и перезагрузить:

```
/path/to/apachectl stop
/path/to/apachectl startssl
```

Расположение двоичных `apachectl` и `http(s)dctl` часто варьируется. Если ваша система имеет команды locate, `whereis` или which, они могут вам помочь найти программы управления вашим сервером.

Далее идут примеры компиляции PHP для Apache.

```
./configure –with-apxs –with-pgsql
```

Это создаст библиотеку `libphp4.so`, которая загружается в Apache с использованием строки LoadModule в файле конфигурации Apache `httpd.conf`. Поддержка PostgreSQL встроена в эту библиотеку `libphp4.so`.

```
./configure –with-apxs –with-pgsql=shared
```

Это создаст библиотеку `libphp4.so` для Apache, а также библиотеку `pgsql.so`, которая загружается в PHP путем использования директивы расширения /extension в файле `php.ini` либо загрузкой ее в скрипт явным образом с использованием функции `dl()`.

```
./configure –with-apache=/path/to/apache_source –with-pgsql
```

Это создаст библиотеку `libphp4.so`, `mod_php4.c` и некоторые сопутствующие файлы и скопирует их в директорию src/modules/php4 вдереве ресурсов Apache.

Затем вы компилируете Apache с использованием –activate-module=src/modules/php4/libphp4.a, и система построения Apache создаст libphp4.a и свяжет ее статически с двоичным httpd. Поддержка PostgreSQL включена непосредственно в этот двоичный httpd, так что окончательным результатом здесь является единственный двоичный httpd, который содержит все из Apache и все из PHP.

```
./configure -with-apache=/path/to/apache_source -with-pgsql=shared
```

То же самое, только вместо включения поддержки PostgreSQL непосредственно в окончательный httpd вы получите библиотеку pgsql.so, которую можете загрузить в PHP из php.ini либо напрямую через использование dl().

При выборе варианта установки PHP вы должны рассмотреть преимущества и недостатки каждого метода. При установке PHP как CGI вы сможете компилировать Apache независимо и не должны ничего рекомпилировать, если добавите или измените PHP. Встраивание PHP в Apache (статический метод) означает, что PHP будет загружаться и работать быстрее.

Для более наглядного представления процесса установки PHP как модуля Apache, мы приведем вам лаконичную последовательность действий для установки. Материал мы использовали с популярного сервера документации www.php.net с полного разрешения автора. В дальнейшем мы будем пользоваться такой же методикой, для того чтобы более точно можно было представить последовательность действий при установке и не совершить какой-либо ошибки:

```
1) gunzip apache_1.3.x.tar.gz
2) tar xvf apache_1.3.x.tar
3) gunzip php-x.x.x.tar.gz
4) tar xvf php-x.x.x.tar
5) cd apache_1.3.x
6) ./configure –prefix=/www
7) cd ../php-x.x.x
8) ./configure –with-mysql –with-apache=../apache_1.3.x –enable-
track-vars
9) make
10) make install
11) cd ../apache_1.3.x
12) ./configure–activate-module=src/modules/php4/libphp4.a
13) make
14) make install
15) cd ../php-x.x.x
```

16) `cp php.ini-dist /usr/local/lib/php.ini`

17) Отредактируйте ваш `httpd` Apache `conf/srm.conf` файл и добавьте строку

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

18) Воспользуйтесь обычной процедурой для перезагрузки сервера Apache.

После того как PHP-модуль установлен и вся конфигурация полностью настроена, можно приступить к выполнению программ.

ВНИМАНИЕ

Когда PHP установлен как CGI, может возникнуть следующая проблема: PHP не понимает некоторые глобальные переменные ($PHP_AUTH_USER $PHP_AUTH_PW и т. д.). Поэтому, если вдруг у вас возникла необходимость работы с глобальными переменными, просто устанавливайте PHP в качестве модуля Apache.

2.2. Установка на системы Linux

Операционная система Linux с годами приобретает все большую и большую популярность. Прежде всего это связано со стоимостью системы, и не на последнем месте стоит возможность простой взаимосвязи с другими программами. Система Linux является одной из самых удобных операционных систем для работы в Сети. Это позволило ей получить широкую популярность у пользователей. Процесс установки PHP на такую платформу не составит никакого труда вообще.

Многие дистрибутивы Linux имеют некоторую разновидность инсталляции пакета, такую как RPM. Это может помочь в установке режимов стандартной конфигурации, но если вам необходимо иметь некоторые особенности (такие как безопасность сервера, различные драйверы баз данных и др.), то вам, возможно, будет нужна компоновка PHP или Web-сервера. Если вы не знакомы с построением и компиляцией вашего программного обеспечения, то, возможно, на сайте `www.php.net`. вы найдете уже созданную версию PHP со всеми особенностями, в которых вы нуждаетесь.

2.3. Установка на системы Windows 9x/Me/NT/2000

Существует два способа установки PHP для операционных систем Windows 9x/Me/NT/2000. Один — при помощи инсталлятора InstallShield, другой заключается в ручной установке. Если вы профессионал или очень хорошо владеете самим пакетом установки, то советуем вам воспользоваться вторым способом. Для пользователей, которые только начинают изучать PHP, второй способ будет не совсем полезен. Но со временем вы сможете освоить все способы установки без каких-либо проблем.

Windows InstallShield

Для установки PHP вам потребуется обратиться на сервер `www.php.net`. Там вы получите полную версию инсталлятора, который установит вам CGI-версию PHP, а также конфигурацию для IIS, PWS и Xitami-серверов.

При запуске Install установите ваш HTTP-сервер и убедитесь, что он работает.

Начните процесс установки. Внимательно следите за мастером и командами, которые он будет предлагать выполнить. Существует два типа инсталляции — стандартный, который обеспечивает установку необходимых компонентов по умолчанию, без выбора пользователя, и расширенный, выбрав который в ходе процесса установки вам придется отвечать на поставленные мастером установки вопросы. Если вы устанавливаете PHP первый раз, советуем вам выбрать стандартную установку.

Мастер установки собирает информацию, чтобы сформировать файл `php.ini`.

Что касается IIS и PWS на NT Workstation, то там отражается список всех узлов на сервере с картой установок сценария. Вы можете выбрать те узлы, к которым желаете применить картографирование сценария PHP.

Как только инсталляция завершится, мастер установки сообщит, нужно ли перезагрузить компьютер, сам сервер или же можно приступить к использованию PHP без перезагрузки.

Ручная установка PHP

Мы выражаем большую благодарность нашему другу и автору данного руководства Бобу Сильва (Bob Silva) за то, что он позволил нам воспользоваться его методами установки PHP. Полную версию его документации на английском языке можно найти по адресу: `http://www.umesd.k12.or.us/php/`.

Это руководство по ручной установке PHP обеспечивает поддержку инсталляции на следующие серверы:

- Personal Web Server 3 или старше;
- Internet Information Server 3 или старше;
- Apache 1.3.x;
- OmniHTTPd 2.0b 1 или старше;
- Oreilly Website Pro;
- Xitami.

PHP 4 для Windows — CGI выполнимая программа (`php.exe`) и несколько модулей SAPI (например, `php4isapi.dll`). Последняя форма является новой в PHP 4 и обеспечивает существенно улучшенную деятельность и некоторые новые функциональные возможности. Однако модули SAPI еще не признаны продуктом высокого качества. Причина этого в том, что PHP SAPI-модули используют безопасную версию PHP-кода, что является новым в PHP и не достаточно проверенным, чтобы считаться полностью устойчивым. С другой стороны, некото-

рые пользователи сообщали об очень хороших результатах в использовании SAPI-модулей. В любом случае, совсем скоро будет возможность пользоваться ими, имея стопроцентную гарантию того, что они работают без каких-либо проблем.

Если вы выберете один из SAPI-модулей и воспользуетесь Windows 95, подготовьтесь к тому, что придется загрузить модификацию DCOM update со страницы Microsoft DCOM. Для модуля ISAPI, требуется Web-сервер ISAPI 4.0 (проверенный IIS 4.0, PWS 4.0 и IIS 5.0), IIS 3.0. Если нужна поддержка PHP, то необходимо загрузить и установить модуль опций Windows NT 4.0 вместе с IIS 4.O.

Прежде чем использовать определенные команды сервера, выполните следующее:

- извлеките из дистрибутива файлы в каталог `C:\PHP\`;
- скопируйте файл `php.ini-dist` в директорию WINDOWS для Windows 95/98 или для Windows NT/2000 в директорию SYSTEMROOT и переименуйте его в `php.ini`. Ваши WINDOWS и SYSTEMROOT могут быть расположены:

  `C:\WINDOWS` для Windows 95/98

  `C:\WINNT` или `C:\WINNT40` для NT/2000 серверов;

- отредактируйте ваш файл php. ini:
  - измените установки строки `extension_dir`, чтобы указать на вашу `php_install_dir`, или, другими словами, установите путь, где вы разместили ваши `php_*.dll` файлы. Например, `C:/PHP`;
  - выберите, какие расширения вы хотели бы загрузить, когда PHP начнет свою работу. Вы можете не компилировать строку `extension=php_*.dll` в файле `php.ini`, для того чтобы загрузить эти расширения. Некоторые из них требуют, чтобы вы установили дополнительные библиотеки на вашу систему, тогда эти модули будут отлично работать. Вы можете загружать модуль динамически в ваш сценарий, используя функцию `dl()`. Более подробно о работе этой функции вы узнаете немного позже;
  - на PWS и IIS вы можете в файле `browscap. ini` записать строку, которая указывала бы На: C: \WINDOWS\SYSTEM\INETSRV\`browscap. ini` в Windows 9x/Me и в NT/2000 сервере.

2.4. Установка расширений функциональных возможностей Windows

После того как произошла установка PHP и Web-сервера на операционные системы Windows, вы, вероятно, пожелаете установить некоторые функциональные расширения, при помощи которых можно воспользоваться необходимыми вам функциональными возможностями. В табл. 2.1 приведены некоторые из доступных добавлений функциональных возможностей. Как было описано ранее, вы можете выбрать по собственному желанию, какое расширение хотели бы запускать сразу после загрузки PHP, конечно, при условии, что у вас установлены необходимые

библиотеки для этого, хотя существуют такие расширения, для которых установка библиотек не нужна. Эти расширения являются встроенными, но о них немного ниже. Также вы можете загрузить динамические библиотеки при помощи функции `dl()`.

Динамические библиотеки DLL для PHP-расширений фиксируются в строке `php_`. Это предотвращает беспорядок между PHP-расширениями и библиотеками, поддерживающими их.

Приведем перечень функциональных возможностей, которые имеют встроенную поддержку в PHP 4.0.4pl1: MySQL, oDBC, FTP, Calendar, BCMath, COM, PCRE, Session, WDDX и XML. Чтобы воспользоваться этими дополнениями, вам даже не потребуется загружать какие-либо дополнительные библиотеки (табл. 2.1). Для знакомства со списком встроенных модулей необходимо взглянуть в файл `readme.txt` или `install.txt`.

Таблица 2.1. Динамические библиотеки PHP

Библиотека	Описание
`php_calendar.dll`	Календарные функции
`php_crypt.dll`	Функции Crypt
`php_dbase.dll`	DBase-функции
`php_dbm.dll`	Berkeley DB2 библиотека
`php_filepro.dll`	Доступ только для чтения к базам данным Filepro
`php_gd.dll`	GD библиотечные функции
`php_hyperwave.dll`	Функции HyperWave
`php_imap4r2.dll`	Функции IMAP 4
`php_ldap.dll`	LDAP-функции
`php_msql1.dll`	MSQL 1 клиент
`php_msql2.dll`	MSQL 2 клиент
`php_mssql.dll`	MSSQL клиент (требует библиотеки MSSQL DB)
`php3_mysql.dll`	MySQL-функции
`php_nsmail.dll`	Netscape mail-функции
`php_oci73.dll`	Oracle-функции
`php_snmp.dll`	SNMP-функции
`phpzlib.dll`	Функция сжатия Zlib

2.5. Тестирование PHP

Давайте теперь убедимся, что PHP-скрипты работают. Для этого обратимся к директории, которую вы создали для хранения документов и программ PHP, обычно это `/www`, там же создадим файл `test.php` со следующим содержанием:

```
<?
echo "Test PHP! <br>\n";
phpinfo();
?>
```

Теперь наберите в браузере путь к этому файлу, например `http://localhost/test.php`. Должна отобразиться страница с разнообразной информацией о PHP, которая генерируется функцией `phpinfo()` (рис. 2.1).

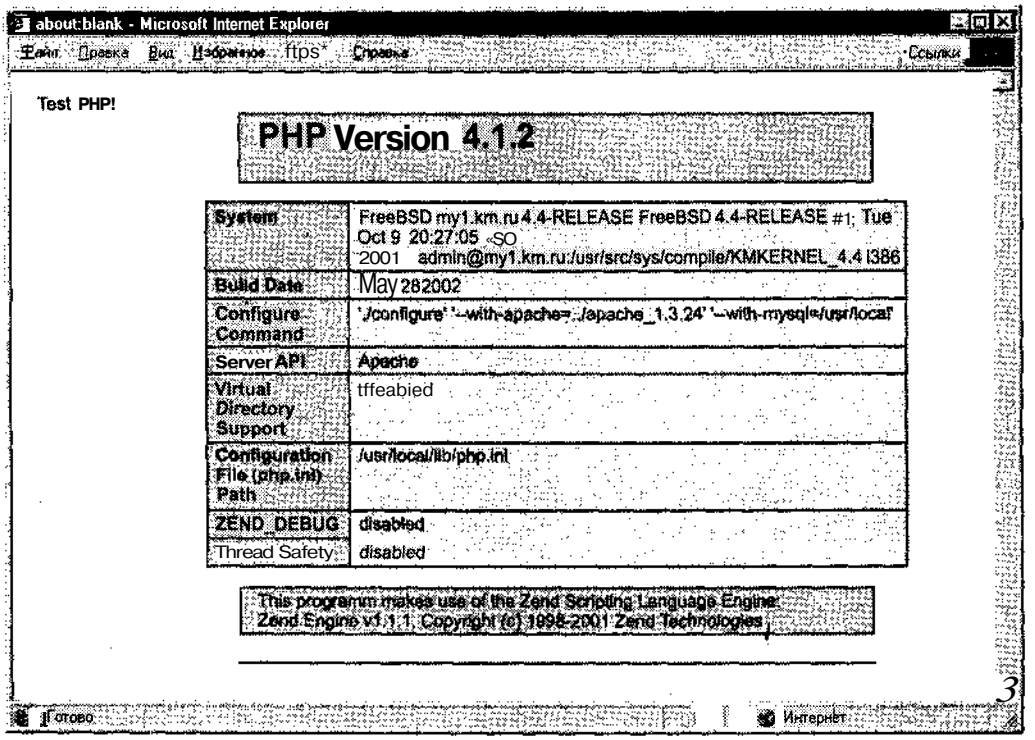


Рис. 2.1. Информация о PHP, сгенерированная функцией phpinfo()

ВНИМАНИЕ

Напоминаем, что PHP-скрипты — не то же самое, что CGI-скрипты. В частности, если CGI-скрипты обычно располагаются в /cgi-bin/ или /cgi/, то PHP-скрипт должен находиться в директории с документами.

Если страница не отображается, значит, вы допустили ошибку в файле `httpd.conf`. Откройте его снова и исправьте ошибку, затем не забудьте перезапустить Apache.

Заключение

Изучение данной главы позволило вам освоить основные принципы установки PHP на различные платформы. Если вы — профессиональный Web-программист, то после внимательного ознакомления с этой главой вы сможете на порядок упростить себе жизнь, точнее, ее часть, касающуюся написания и отладки скриптов.

Глава 3
Быстрая установка Apache-сервера

Как известно, для того чтобы создавать какие-либо программы на PHP, иметь обычный интерпретатор не достаточно. Помимо интерпретатора и прилагаемых библиотек вам понадобится сам сервер. Некоторые особенности по установке PHP на разнообразные серверы были приведены ранее (см. гл. 2). Теперь же мы поговорим о максимально быстром способе установки сервера Apache. Сначала нужно скачать сам Apache. Мы рекомендуем сделать это с официального сайта разработчиков сервера Apache — `http://www.apache.org/` (рис. 3.1).

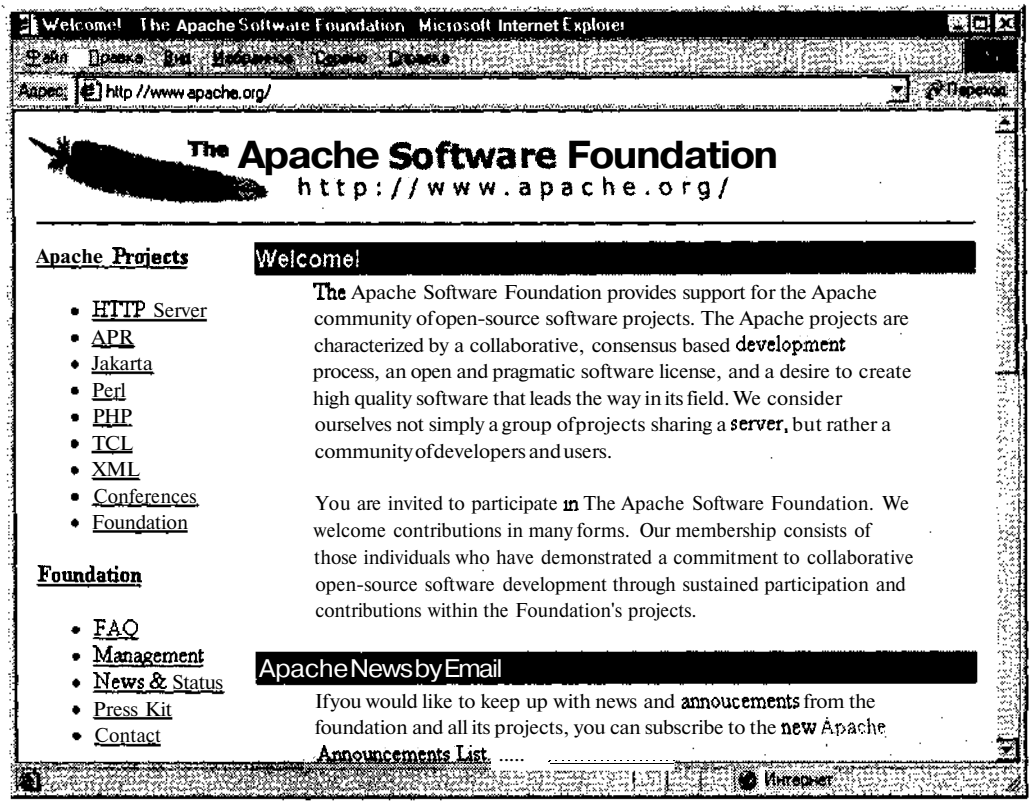


Рис. 3.1. Сайт www.apache.org

Данная глава рассказывает о следующем:

- потребность в виртуальном сервере;
- установка Apache;
- виртуальные хосты Apache.

3.1. Потребность в виртуальном сервере

Прочитав эту главу, вы сможете установить один из лучших на сегодняшний день серверов — Apache. Если вы уже скачали необходимые дистрибутивы, о которых упоминалось выше, смело приступайте к делу.

А для чего, собственно, нужен сервер на домашнем компьютере? Дело в том, что часто возникает необходимость проверить полноценный вид вашей HTML-страницы или работоспособность скрипта на PHP, Perl и т. д.

Но, как правило, проверить дома это невозможно, так как SSI, CGI и, естественно, PHP, обязательно требуют сервера. Вот тут на помощь приходят специальные программы (Web-серверы), которые позволят установить сервер даже на компьютер, не подключенный к Интернету. Мы рассмотрим один из самых удобных и быстрых серверов — Apache. Ведь как ни крути, а это практически единственный сервер, который позволяет работать в Windows 95/98 с CGI, SSI и PHP Perl-скриптами одновременно! Причем так же непринужденно, как будто у вас установлен Unix.

3.2. Установка Apache

Итак, мы скачали нужный дистрибутив PHP, запаслись терпением и полностью готовы установить его на свой компьютер под управлением Windows 95/98.

ВНИМАНИЕ

Точно выполняйте все приведенные ниже шаги установки, не пропуская ни одного. Только в этом случае сервер заработает — это проверено. Вам нужно просто выполнять те или иные действия, иначе у вас есть шанс провести пару не самых приятных часов (дней) за изучением документации Apache.

Запустите мастер установки Apache. Первым вопросом будет предложение согласиться с условиями лицензии либо же отказаться от них. Нажмите «Yes», чтобы продолжить установку.

Далее нажимайте кнопку «Next» в появляющихся окнах до тех пор, пока не увидите вопрос о выборе директории для установки Apache. Мы рекомендуем оставить директорию по умолчанию, впрочем, вы можете выбрать более удобный для вас вариант. А путь, предлагаемый по умолчанию, может выглядеть примерно так:

```
C:\Program Files\Apache Group\Apache
```

В появившемся окне установите флажок Typical (стандартный — этот вариант установки подходит наибольшему числу пользователей) и нажмите кнопку «Next».

Далее Apache предложит вам создать папку в меню «Пуск» в папке «Программы». Если вы согласны, то нажмите кнопку «Next». Сразу после этого начнется копирование программного обеспечения.

После окончания копирования нажмите кнопку «Finish» и тем самым завершите установку сервера. Теперь приступим к его настройке.

На первых этапах настройки нужно определиться с директорией, в которой будут храниться ваши сайты. Вариант, предлагаемый Apache по умолчанию, не слишком удобен — `C:\Program Files\Apache Group\Apache\htdocs`. Там же можно найти и документацию по серверу. Но, как сами видите, этот путь слишком длинный, и поэтому будет гораздо удобнее сделать для всех сайтов отдельный диск (например, с именем `Z:`) при помощи утилиты subst, которая входит в поставку Windows.

Директорий, в которых будут храниться ваши сайты, может быть несколько. Допустим, это будет `C:\INTERNET`. Наша директория будет содержать корневой каталог нового диска `Z:`.

В начале файла autoexec.bat (но после команды @echo off, если она у вас там есть) напишите такую строку:

```
subst Z: C:\INTERNET
```

Теперь все, что записано в директории `C:\INTERNET`, будет отображаться на диске `Z:`, как будто это обычный жесткий диск.

ВНИМАНИЕ

В Windows 95/98 есть ошибка, в результате которой при использовании subst, пути иногда преобразуются в абсолютные. В нашем случае Z: преобразуется в C:\INTERNET, причем в процессе работы какой-нибудь программы и совершенно неожиданно для нее. Однако если вы настроите все так, как описано выше, ошибки возникнуть не должно.

СОВЕТ

Вы можете также создать диск Z: с помощью какой-нибудь программы для виртуальных разделов (например, с помощью встроенной в Windows 95/98 программы DriveSpace). Это решение, пожалуй, даже лучше, чем использование subst, как с точки зрения экономии памяти, так и с точки зрения быстродействия. Ведь что такое Web-сайт, как не набор небольших файлов? A DriveSpace как раз оптимизирует работу с такими файлами. Описание того, как использовать DriveSpace, смотрите во встроенной в Windows документации.

Создайте на диске `Z:` директорию home, а в ней — каталог `localhost`. В нем будет храниться содержимое главного хоста Apache — того, который доступен по адресу

`http://localhost`. Перейдите в каталог Localhost. Создайте в нем директории cgi и `www`. В первой будут храниться CGI-скрипты, а во второй — ваши документы и программы на PHP. Подобную операцию вам нужно будет проделывать каждый раз при создании нового виртуального хоста (см. п. 3.3).

Откройте в «Блокноте» файл конфигурации `httpd.conf`, который расположен в поддиректории conf директории Apache (в нашем примере это `C:\Program Files\Apache Group\Apache`). Впрочем, вы можете и не искать этот файл вручную, а воспользоваться ярлыком **Edit configuration** в меню **Пуск ▸ Программы ▸ Apache Web Server ▸ Management** (если в процессе установки вы разрешили программе инсталляции создать ярлыки в меню **Пуск ▸ Программы**). `Httpd.conf` — единственный файл, который нужно подредактировать. Вам предстоит найти и изменить в нем некоторые строки. Чтобы избежать ошибок в настройках, не трогайте все остальное. Хотя нужно отметить, что в файле каждый параметр сопровождается несколькими строками комментариев и с первого раза разобраться в них тяжело.

Для начала настроим параметры для главного хоста Apache — localhost, а также параметры по умолчанию, которые будут унаследованы всеми остальными виртуальными хостами, если возникнет необходимость их создать.

Установите поле ServerName, как изображено на рис. 3.2.

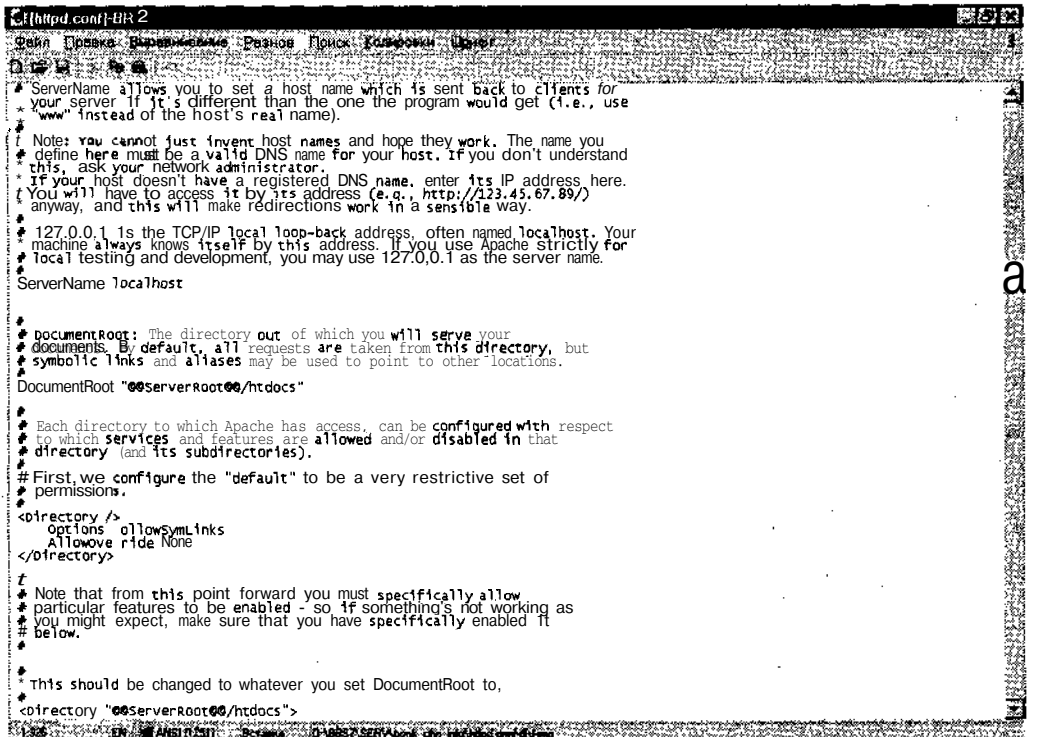


Рис. 3.2. Установка поля ServerName

ВНИМАНИЕ

Не забудьте раскомментировать поле ServerName, т. е. убрать символ «#» перед этим параметром (по умолчанию он закомментирован). Все, что идет после этого символа и до конца строки, Apache игнорирует.

В поле DocumentRoot укажите ту директорию, в которой будут храниться ваши HTML-файлы. Мы ранее условились, что это будет z:\home\localhost\www:

```
DocumentRoot "z:/home/localhost/www"
```

Найдите блок, начинающийся строкой <Directory/> и заканчивающийся </Directory> (вообще, такие блоки обозначают установки для заданной директории и всех ее поддиректорий). Этот блок может содержать множество комментариев — не обращайте на них внимания. Его нужно изменить на такой блок:

```
<Directory z:/>
Options Indexes Includes
AllowOverride All
Allow from all
</Directory>
```

Таким образом, в этом блоке будут храниться установки для всех директорий по умолчанию (так как это — корневая директория). Для всех директорий по умолчанию устанавливается возможность автоматической генерации индекса — списка содержимого директории при просмотре ее в браузере, а также поддержка SSI и разрешение использовать файлы .htaccess для индивидуальных настроек каталогов.

Найдите аналогичный блок, начинающийся <Directory "C:/Program Files/Apache Group/Apache/htdocs"> и заканчивающийся </Directory>. На комментарии не обращайте внимание. Этот блок нужно удалить, так как все настройки для директории со страницами должны наследоваться от настроек по умолчанию, которые мы только что установили.

Настройте DirectoryIndex так:

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

Это так называемые файлы индекса, которые автоматически выдаются сервером при обращении к какой-либо директории, если не указано имя HTML-документа. Можно добавить сюда и другие имена, например index.php и т. д., однако дополнительные настройки все же лучше делать в файлах .htaccess для каждого сайта в отдельности (рис. 3.3).

Найдите такой параметр:

```
ScriptAlias /cgi-bin/ "z:/home/localhost/cgi/"
```

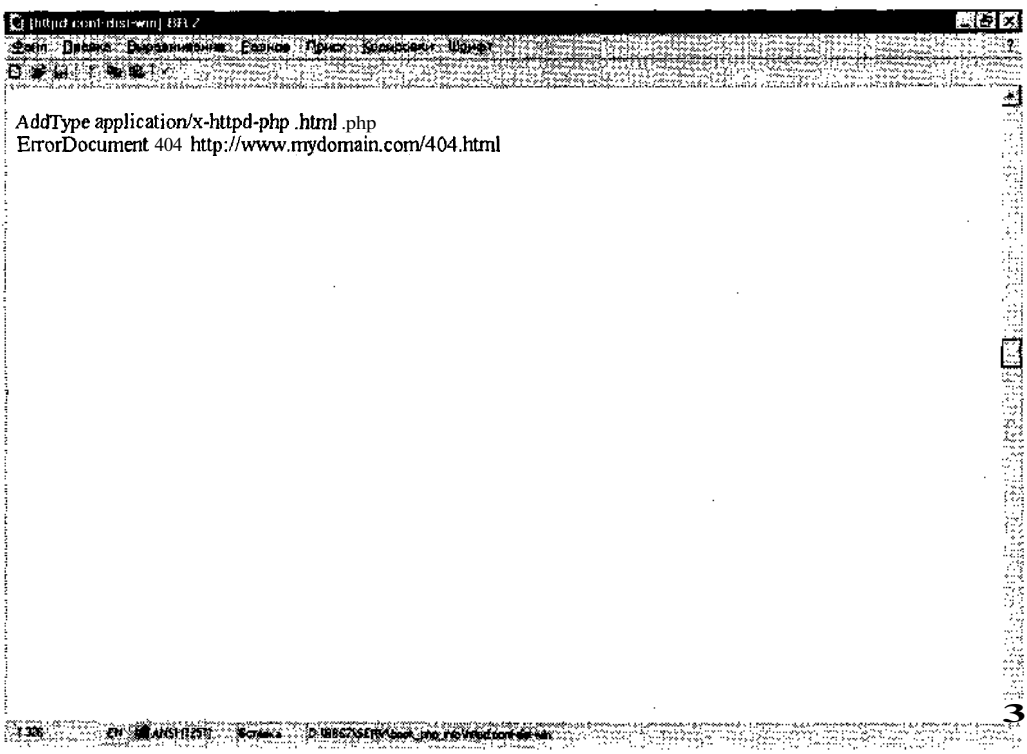


Рис. 3.3. Редактирование файла .htaccess

Добавьте после него еще такую строку:

```
ScriptAlias /cgi/ "z:/home/localhost/cgi/"
```

Именно так, с двумя слешами. Это будет та директория, в которой должны храниться ваши CGI-скрипты. Подобный параметр говорит Apache о том, что если будет указан путь вида `http://localhost/cgi-bin`, то на самом деле следует обратиться к директории `z:` /home/localhost/cgi. Два синонима для CGI-директории используются потому, что /cgi-bin/ будет доступна не только главному хосту `localhost`, но и всем остальным виртуальным хостам. В то же время у каждого из них будет дополнительно своя CGI-директория `/cgi/`.

Теперь следует найти блок параметров, начинающийся с <Directory "C: /Program E'iles/Apache Group/Apache/cgi-bin">и заканчивающийся </Directory>. Это установки для CGI-директории. Мы не собираемся указывать никаких дополнительных параметров вместо тех, которые уже установлены по умолчанию, поэтому этот блок нужно удалить.

Найдите и настройте (не забудьте раскомментировать!) следующий параметр:

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

Это говорит Apache о том, что файлы с расширениями exe, bat и cgi нужно рассматривать как CGI-скрипты.

И последнее — установите следующие параметры:

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

При этом Apache будет обрабатывать файлы с указанными расширениями процессором SSI (рис. 3.4).

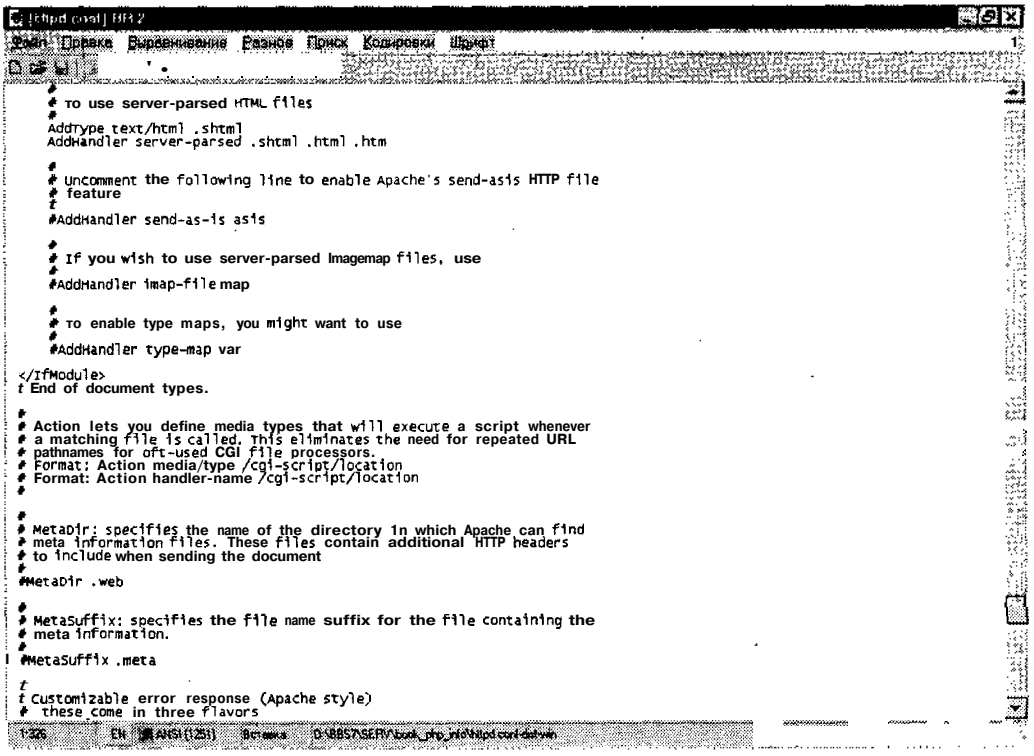


Рис. 3.4. Установка расширений файлов

Теперь не забудьте сохранить изменения и закройте блокнот.

Вот вы и настроили свой домашний сервер Apache. Если вы соблюдали все пункты настройки и правильно выполняли все действия, то сервер должен быть работоспособен. Сейчас мы это проверим. Сначала нужно запустить сервер, для этого нажмите **Пуск ▸ Программы ▸ Apache Web Sever ▸ Management ▸ Start Apache.** Появится окно, похожее на «Сеанс MS-DOS», и ничего больше не произойдет. Не закрывайте это окно до конца работы с Apache.

ВНИМАНИЕ

Если окно открывается и тут же закрывается, это означает, что вы допустили какую-то ошибку в файле httpd.conf. В этом случае придется искать неточность. Проще всего это сделать следующим образом.

Запустите «Сеанс MS-DOS». Для этого нажмите **Пуск > Выполнить.** Наберите в появившемся диалоге command и нажмите «Enter». Появится подсказка командной строки. Если у вас нет возможности запустить «Сеанс MS-DOS», воспользуйтесь каким-нибудь другим средством. Например, запустите Far Manager и работайте через него. Просмотр результатов работы программы — Ctrl+O.

Наберите такие команды DOS:

```
c:
cd "\Program Files\Apache Group\Apache"
apache.exe
```

Если до этого Apache не запускался, то вы получите сообщение об ошибке и номер строки в httpd.conf, где она произошла. Исправьте httpd.conf и повторите описанный процесс сначала, до тех пор, пока в окне не отобразится что-то вроде Apache/1.3.14 (Win32) running...

Несколько слов о том, как можно упростить запуск и завершение работы сервера. В Windows можно назначить любому ярлыку функциональную комбинацию клавиш, нажав которые вы запустите этот ярлык. Щелкните правой кнопкой на панели задач, в контекстном меню выберите **Свойства К Настройка меню ▸ Дополнительно. В** открывшемся «Проводнике» назначьте ярлыку Start Apache комбинацию Ctrl + Alt + A, а ярлыку Stop Apache — Ctrl + Alt + S. Теперь вы сможете запускать сервер нажатием Ctrl + Alt + A и останавливать его нажатием Ctrl + Alt + S.

Теперь проверим, правильно ли мы настроили директории для документов.

Проверка HTML

В директории `z:/home/localhost/www` с HTML-документами Apache создайте файл `index.html` с любым текстовым содержанием.

Теперь запустите браузер и наберите:

```
http://localhost/index.html
```

или просто

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

После этого должен загрузиться ваш файл.

Проверка SSI

В директории `z:/home/localhost/www` с HTML-документами Apache создайте файл `test.shtml` с таким содержанием (внимательно следите за соблюдением пробелов в директиве include):

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

Теперь наберите в браузере:

```
http://localhost/test.shtml
```

Должен загрузиться файл, состоящий из текста «SSI Test!», за которым следует содержимое файла index.html между двумя горизонтальными чертами. Если этого не произошло, значит, вы неправильно настроили работу SSI.

Проверка CGI

В директории z:/home/localhost/cgi для CGI-скриптов создайте файл test.bat с таким содержанием:

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

Теперь в браузере наберите:

```
http://localhost/cgi/test.bat
```

В окне отобразится результат команды DOS dir.

ВНИМАНИЕ

Нужно отметить, что указанный тест работает не на всех версиях Windows: иногда, вместо того чтобы выполнить файл test.bat, Apache выводит в браузер его содержимое (а иногда Windows запускает этот файл в отдельном окне). С чем это связано — не совсем ясно, однако можно избавиться от указанной ошибки путем манипулирования с реестром Windows. Да и не стоит расстраиваться, если у вас не запускается test.bat: вряд ли вы когда-нибудь будете писать скрипты в виде bat-файлов, тем более, что этот способ несовместим с Unix.

Если что-то пошло не так либо окно Apache открывается и тут же закрывается, значит, где-то произошла ошибка — скорее всего, в httpd.conf. За детальным разъяснением ее причин можно обратиться к лог-файлам, расположенным в директории C:\Program Files\Apache Group\Apache\logs.

3.3. Виртуальные хосты Apache

Итак, вы установили Apache и создали директорию z:/home/localhost/www для хранения документов и z:/home/localhost/cgi для CGI-скриптов. Однако в Интернете вы, скорее всего, будете поддерживать несколько серверов, а Apache создал для вас только один. Конечно, структуру этих нескольких серверов можно хранить на одном сервере, однако проще и удобнее было бы создать несколько виртуальных хостов с помощью Apache. В нашем распоряжении есть два вида виртуальных хостов: отдельные для каждого IP-адреса и использующие один общий IP-адрес (так называемые name-based-хосты — хосты, определяемые по имени). В тренировочных целях рассмотрим оба варианта, а именно, создадим хост my_host, использующий тот же адрес, что и localhost, а также хост my_host_new с адресом 127.0.0.2.

ВНИМАНИЕ

Конечно, вместо «my_host» и «my_host_new» вам нужно будет указать желаемые имена ваших виртуальных хостов. Советуем назвать их так же, как и на вашем настоящем Web-сервере, но только без суффикса .ru или .com - это может многое упростить при программировании скриптов.

Как это принято в Unix, каждый сервер будет представлен своим каталогом в директории z:/home с именем, совпадающим с именем сервера (мы уже проделывали нечто подобное с хостом localhost). Например, сервер my_host будет храниться в директории z:/home/my_host, которую вам необходимо создать прямо сейчас (конечно, вместе с ее поддиректориями cgi и www, как мы делали это ранее), а хост my_host_new — в директории z:/home/my_host_new. В этих директориях будут находиться:

- файлы access.log с журналом доступа к виртуальному серверу;
- файлы errors.log с журналом ошибок сервера;
- директория www, где, как обычно, будут храниться HTML-документы;
- директория cgi для хранения CGI-программ.

Для установки виртуальных хостов необходимо сделать некоторые изменения в файле конфигурации Apache httpd.conf, а также в некоторых файлах Windows.

Откройте файл httpd.conf (можете для этого воспользоваться ярлыком **Edit configuration в меню Пуск ▸ Программы ▸ Apache Web Server ▸ Management). Добавьте** следующие строки в конце файла после всех комментариев:

```
NameVirtualHost 127.0.0.1
#—localhost
<VirtualHost localhost>
ServerAdmin webmaster@localhost
ServerName localhost
DocumentRoot "z:/home/localhost/www"
```

```
ScriptAlias /cgi/ "z:/home/localhost/cgi/"
ErrorLog z:/home/localhost/error.log
CustomLog z:/home/localhost/access.log common
</VirtualHost>
#—my_host
<VirtualHost my_host>
ServerAdmin webmaster@my_host.ru
ServerName my_host
DocumentRoot "z:/home/my_host/www"
ScriptAlias /cgi/ "z:/home/my_host/cgi/"
ErrorLog z:/home/my_host/error.log
CustomLog z:/home/my_host/access.log common
</VirtualHost>
#—my_host_new
<VirtualHost my_host_new>
ServerAdmin webmaster@my_host_new.ru
ServerName my_host_new
DocumentRoot "z:/home/my_host_new/www"
ScriptAlias /cgi/ "z:/home/my_host_new/cgi/"
ErrorLog z:/home/my_host_new/error.log
CustomLog z:/home/my_host_new/access.log common
</VirtualHost>
```

ВНИМАНИЕ

Мы добавили дополнительную секцию <VirtualHost> для хоста localhost. Если этого не сделать, то все запросы к хосту (т. е. по адресу 127.0.0.1) будут обработаны name-based хостом my_host. Происходит это из-за того, что хосты в секции <VirtualHost> имеют больший приоритет при обработке, чем главный хост, который мы создали до этого.

Директива NamevirtualHost говорит серверу, что указанный IP-адрес может использоваться несколькими виртуальными хостами, поэтому для обработки запросов, поступающих на этот адрес, нужно использовать протокол HTTP 1.1 (который, собственно, и поддерживает технику работы с хостами name-based).

При желании можно добавить и другие параметры в блоки <virtualHost> (например, `DirectoryIndex` и т. д.) Непереопределенные параметры наследуются виртуальным хостом от главного. Не злоупотребляйте настройками в этих секциях — лучше сделать их в файле `.htaccess` в директории нужного хоста, потому что компания, которая предоставляет (будет предоставлять) вам «настоящие» виртуальные хосты в Интернете, вряд ли позволит менять эти блоки.

Но как же система узнает, что хост `my_host_new` сопоставлен с адресом 127.0.0.2, а `my_host` — вообще name-based-хост? Для решения проблемы надо немного подправить системный файл hosts, который находится в директории `C:\WINDOWS` для операционных систем Windows 95/98/Millenium и `C:\WINNT\SYSTEM32\DRIVERS\...` для Windows NT/2000/XP.

ВНИМАНИЕ

Не путайте файл hosts (без расширения) с файлом hosts.sam, который, скорее всего, также расположен в той же директории. Последний файл является просто демонстрационным примером Microsoft и никак не используется системой. Если файла hosts не существует, его необходимо создать.

Файл `hosts` — обычный текстовый файл, и в нем может быть заранее записана только одна строка:

```
127.0.0.1 localhost
```

Именно эта строка и задает соответствие имени localhost адресу 127.0.0.1. Для нашего виртуального хоста надо добавить соответствующую строку, чтобы файл выглядел так:

```
127.0.0.1  localhost  my_host
127.0.0.2 my_host_new
```

Обратите внимание на то, что хост `my_host` описан на той же строке, что и localhost. Дело в том, что в файле hosts должны указываться только уникальные IP-адреса. Если же одному адресу сопоставляется сразу несколько хостов, то один из них (тот, который идет первым) объявляется главным, а остальные — его синонимами. В нашем случае localhost — главный, а `my_host` — его синоним. Apache при получении запроса на адрес 127.0.0.1 узнает, что он пришел хосту с именем my_host, и активизирует соответствующий блок `<VirtualHost>`.

Итак, мы создали виртуальные хосты со следующими свойствами.

Хост `my_host`:

имя — `my_host`;

доступен по адресу `http://my_host`;

расположен в директории `z:/home/my_host`;

директория для хранения документов — `z:/home/my_host/www`, доступная по адресу `http://my_host/`;

директория для CGI-скриптов — z : /home/my_host/cgi, доступная по адресу http://my_host/cgi/;

файлы журналов хранятся в z:/home/my_host.

Хост my_host_new:

имя — my_host_new;

доступен по адресу http://my_host_new или http://127.0.0.2;

расположен в директории z: /home/my_host_new;

директория для хранения документов — z:/home/my_host_new/www, доступная по адресу http://my_host_new/;

директория для CGI-скриптов — z : /home/my_host_new/cgi, доступная по адресу http://my_host_new/cgi/;

файлы журналов хранятся в z:/home/my_host_new.

Заметьте, что главный хост (невиртуальный, тот, который мы создали ранее) по-прежнему доступен по адресу http://127.0.0.1 или http://localhost. Более того, его директория /cgi-bin/ «видна» всем созданным виртуальным хостам, так что вы можете ее использовать.

После всех изменений не забывайте перезапускать Apache.

ВНИМАНИЕ

Просто закрыть окно сервера, нажав на кнопку «Закрыть» в его правом верхнем углу, недостаточно. Нужно воспользоваться ярлыком Stop Apache в меню **Пуск ▸ Программы ▸ Apache Web Server > Management.** В противном случае закроется только окно Apache, а сам сервер останется работать в фоновом режиме, так что изменения, внесенные в httpd.conf, не будут активизированы.

Заключение

Данная глава описывает процесс установки сервера Apache. Если вам надо более подробно ознакомиться с принципами профессиональной установки сервера, изучите полную документацию по установке.

Глава 4
Текстовый редактор, используемый для редактирования PHP-скриптов

Во все времена люди стремились к совершенству. В результате появлялись все новые и новые средства коммуникации, возможности ведения рабочей деятельности на предприятии. Например, если раньше каждый завод имел определенное число портных, которые шили вещи, заказанные покупателем, то теперь это делают машины. Улучшились условия труда, появились новые способы обработки информации, на заводах используются станки, которые позволяют автоматически, без необходимого числа рабочих, выполнять те же действия за очень короткие сроки. Все это создано непосредственно для того, чтобы ускорить производство и улучшить качество выпускаемого товара с наименьшими затратами.

Это относится и к написанию программ. По сути дела, это отдельная быстро развивающаяся отрасль, имеющая свои положительные и отрицательные стороны. Интеллектуально мыслящие машины придумать пока не смогли, поэтому для улучшения всех условий работы программистов и были созданы так называемые среды разработки — Integrated Development Environment (IDE — Интегрированная среда разработки). Данная среда позволяет программисту не только быстро создавать ту или иную программу, но и выполнять ее, комментировать и сразу встраивать в необходимый файл. Для языка программирования PHP выделим две наиболее удобные в использовании среды разработки:

- EditPlus;
- UltraEdit.

Конечно, существуют и другие редакторы. Мы хотели бы рассмотреть именно эти, так как простота их настройки и использования позволит вам без проблем освоить остальные. В данной главе освещены следующие вопросы:

- сравнение редакторов EditPlus и UltraEdit;
- возможности при использовании EditPlus;
- меню **File;**
- меню **Edit;**
- меню **View;**
- меню **Search;**
- меню **Document;**
- меню **Project;**
- меню **Tools;**
- меню **Window;**
- **меню Help;**
- панель инструментов.

4.1. Сравнение редакторов EditPlus и UltraEdit

Как только вам предлагают выбрать тот или иной программный продукт, вы сразу задумываетесь, какой же из них лучше, быстрее в освоении, легок в использовании и т. д. Не испытав их всех в действии, вы так и не сможете ответить на эти вопросы. Приведенные выводы относительно EditPlus и UltraEdit просто являются объяснением того, что имеет тот или иной редактор. Мы не навязываем вам свою точку зрения, а стараемся показать основные отличия при использовании данных редакторов. После этого приведем детальное описание редактора EditPlus. Не потому, что он лучший, а только потому, что является быстро осваиваемым и легким в понимании.

При установке EditPlus и начале работы не возникает никаких проблем. После того как вы произвели установку и загрузку редактора, на экране вашего монитора появится следующая картинка (рис. 4.1).

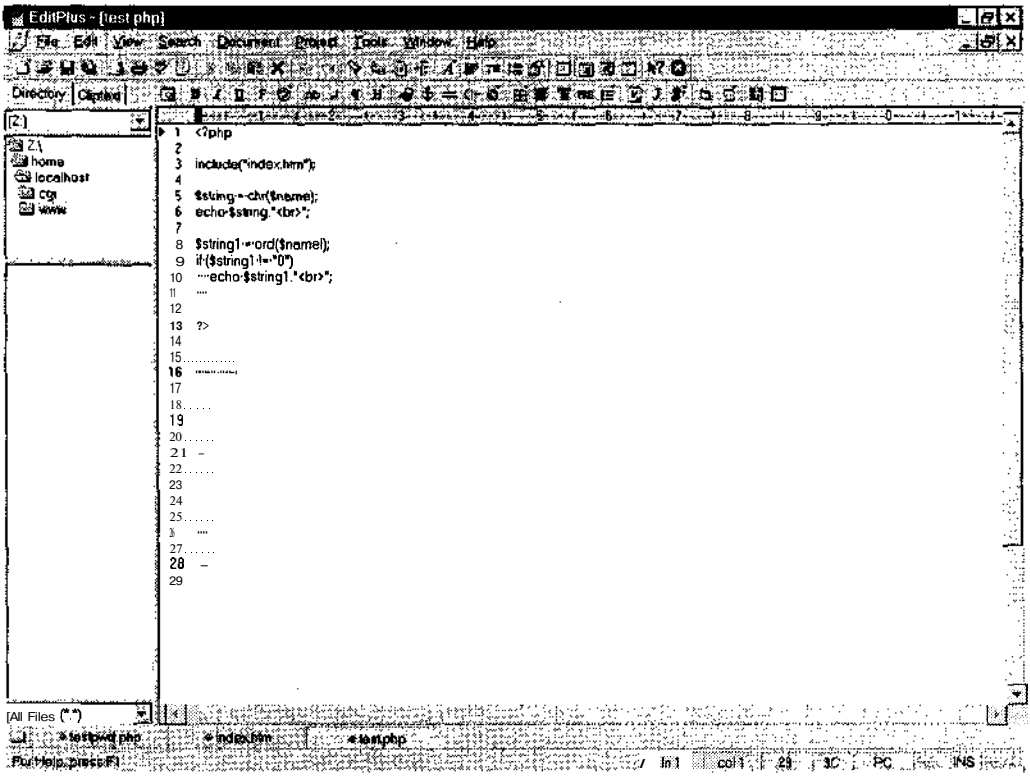


Рис. 4.1. Окно редактора EditPlus

Это стартовый вид редактора. На первый взгляд он не представляет из себя ничего особенного, хотя при дальнейшем изучении вы поймете, что это далеко не так.

На наш взгляд, это самый оптимальный редактор для создания и отладки скриптов. Причем не важно, каких скриптов, поскольку в этом редакторе можно настро-

ить, кажется, абсолютно все. Очень удобное выделение цветом текста с поддержкой синтаксиса, великолепная система проектов, пользовательские шаблоны, автозавершение (полностью регулируемое), закачка по FTP, поиск и замена во всех открытых файлах, огромный архив пользовательских файлов с синтаксисом «для-всего-на-свете» и другими полезными вещами.

Основные минусы данного редактора:

- неудобное выделение CSS по умолчанию;
- нельзя выделять блоки, можно только линии;
- shareware на 30 дней (после установки бесплатно работает только 30 дней).

Сам редактор EditPlus является 32-битным текстовым редактором для создания HTML-страниц и всевозможных программных реализаций. По простоте использования он чем-то напоминает блокнот Windows Notepad.

Редактор выделяет цветом синтаксические конструкции следующих языков:

- HTML,
- CSS,
- PHP,
- ASP,
- Perl,
- C/C++,
- Java,
- JavaScript,
- VBScript.

Помимо этого можно задать цветовое выделение и для языков программирования, не указанных выше.

Редактор также способен просматривать HTML-страницы во встроенном браузере, при этом имеются команды для обновления местных файлов при помощи FTP-сервера.

ВНИМАНИЕ

Основные характеристики, которыми должен обладать ваш компьютер при установке EditPLus, должны быть следующие:

- Windows 95/98/Me/NT4/2000 или выше,
- Internet Explorer 3 или выше для просмотра реализуемого вами кода.

Дистрибутив для установки данного редактора вы сможете найти по адресу: `http://www.editplus.com/`. Тут также содержатся ответы на вопросы, касающиеся установки и любых проблем, связанных с ней.

Теперь перейдем к рассмотрению второго редактора — UltraEdit (рис. 4. 2).

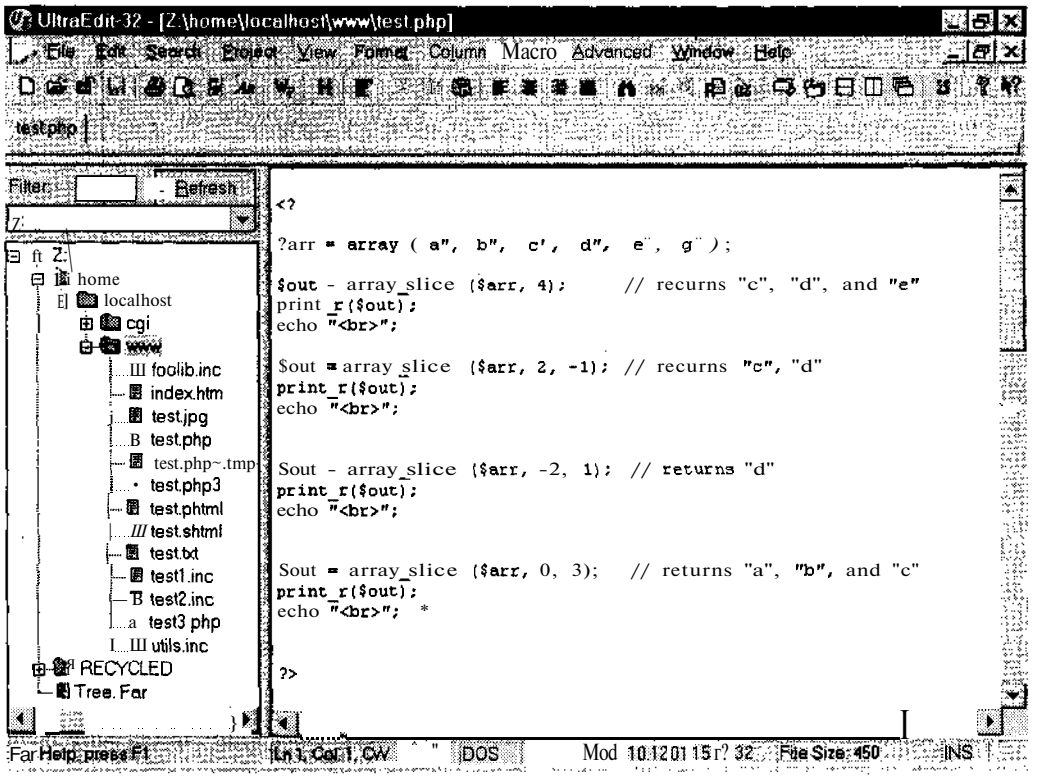


Рис. 4.2. Окноредактора UltraEdit

Данный редактор можно найти по адресу: http: //www.ultraedit.com.

Это прекрасный редактор для программиста. Он обладает множеством необходимых функций:

- практически любое синтаксическое цветовое выделение (файлы можно скачать на сервере),
- возможность работы по FTP,
- поддержка проектов,
- работа в режиме колонок иблоков,
- HEX-редактирование,
- полностью настраиваемая панельинструментов,
- макросы,

а также многое другое, перечисление которого может занять не одну страницу нашей книги.

Единственный недостаток, который хотелось бы выделить, — вставка HTML-тегов, на наш взгляд, организована не очень удачно.

Но это только с точки зрения авторов, и существенным недостатком данного редактора это считать не стоит.

Приведенное описание поможет вам понять структуру редакторов, а также остановить свой выбор на том, который действительно будет достойным помощником в вашей работе.

4.2. Новые возможности редактора EditPlus 2.10

Как вы думаете, чем отличаются новые версии программ от старых? Да, вы правы, прежде чем выпустить ту или иную версию очередной программы, создатели задумываются над такой задачей: а чем же она будет оригинальнее предыдущей, что именно в ней будет такого, чего не было ранее. Так можно сказать практически о любой программе, например, Windows 95, Windows 98, Windows 2000 и Windows XP. Думаем, вам не стоит рассказывать о преимуществах и основных отличиях данных операционных систем, но факт состоит в том, что в каждой из них они есть и каждая новая система обязательно отличается от предыдущей. Точно также и редактор EditPlus 2.10 имеет много преимуществ по отношению к предыдущей версии. Хотелось бы более детально вас с ними ознакомить, чтобы при выборе того или иного редактора также могли обращать внимание на такие тонкости.

Когда загружен сам редактор без каких-либо файлов (рис. 4.3), он выглядит немного по-другому, нежели с обрабатываемым файлом (см. рис. 4.2).

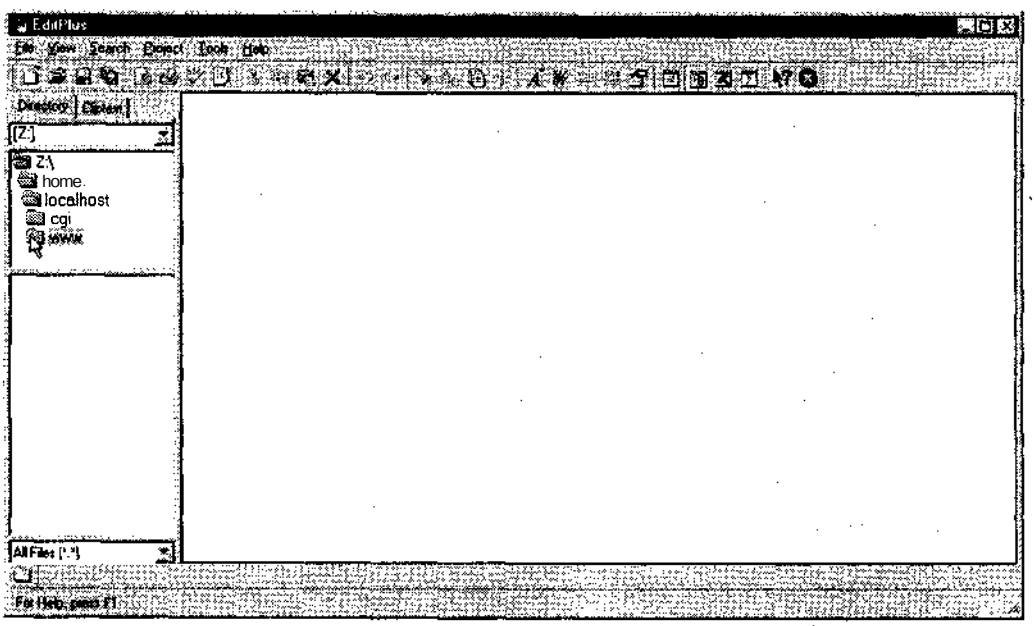


Рис. 4.3. Вид редактора EditPlus в отсутствие открытых файлов

Пока не открыт никакой файл, количество опций, производящих работу данного редактора, минимально. После того как файл открыт, в главном меню появятся еще три пункта: **Edit, Document, Windows.** Именно в этой строке представлено все богатство возможностей данного редактора. Пусть вас не пугает изобилие пунктов меню — вы будете изучать их постепенно, Основные составляющие интерфейса описаны в следующих параграфах.

Приведем основные нововведения редактора EditPlus 2.10:

- окно каталогов (меню **View**);
- улучшенная команда «автоматический отступ» (Settings & Syntax page);
- поддержка справочных файлов формата HTML (*.chm) в качестве инструмента пользователя;
- FTP не требует Internet Explorer;
- **расширенные параметры FTP (server** type, transfer type, firewall);
- настраиваемый образец выхода для инструментальных средств пользователя;
- поддержка файлов Unicode;
- команда **Page Break (Edit▸ Insert Menu);**
- функциональный блок Диалога **List** (меню Search);
- опция пропуска HTML-тегов (Spell Checker page);
- **next/Prev** команда **Column Marker** (меню **Document);**
- команда Keyboard Map (меню **Help**);
- команда **LineComment** (меню **Edit▸ Format**);
- позволяет управлять до 50 проектами;
- команда **Goto Output Window** (меню **View**);
- команда Word Count (меню **Edit**);
- значение HEX в строке состояния;
- настраиваемые разделители файла (General page);
- позволяет назначать сочетания клавиш операциям (Alt + 0 — Alt + 9);
- виртуальное использование памяти.

Не стоит бояться такого объема нововведений. Со всеми ними вы познакомитесь при дальнейшем изучении этой книги.

4.3. Меню File

Меню **File** позволяет производить операции с файлами. Имеет такие же опции, как и любое приложение Windows.

- New — создает новый документ. Документы могут быть с различным расширением. Например, можно сразу создавать файлы PHP, CSS, JAVA и т. д. Данная опция имеет свои настройки и особенности (рис. 4.4):

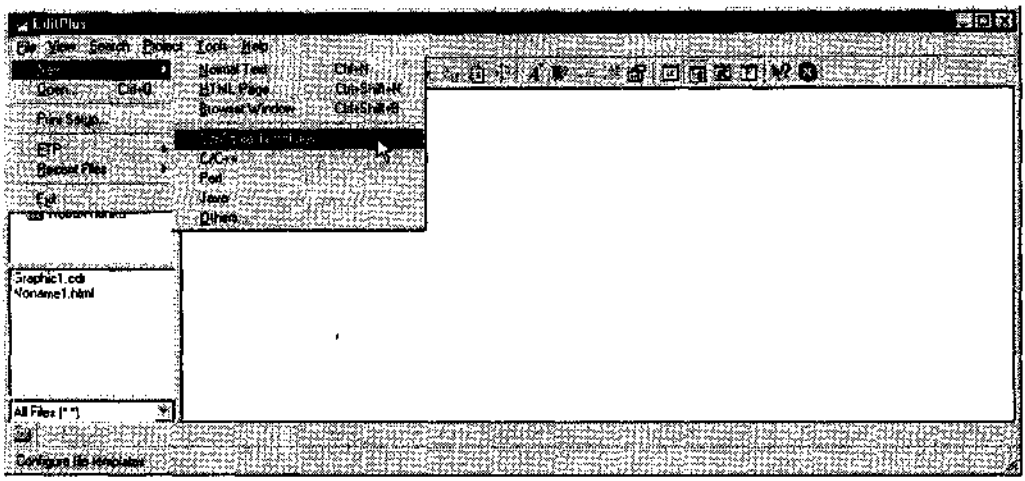


Рис. 4.4. Меню File

- **Normal Text** — создает обычный документ, после работы с которым вы сами будете задавать расширение данного документа;
- **HTML Page** — создает новый HTML-документ;
- **Browser Window** — создает (вызывает) окно браузера;
- **Configure Templates** — конфигурирует (удаляет и добавляет) наименования документов в опцию **File** ▸ New. Это делается элементарно (рис. 4.5);

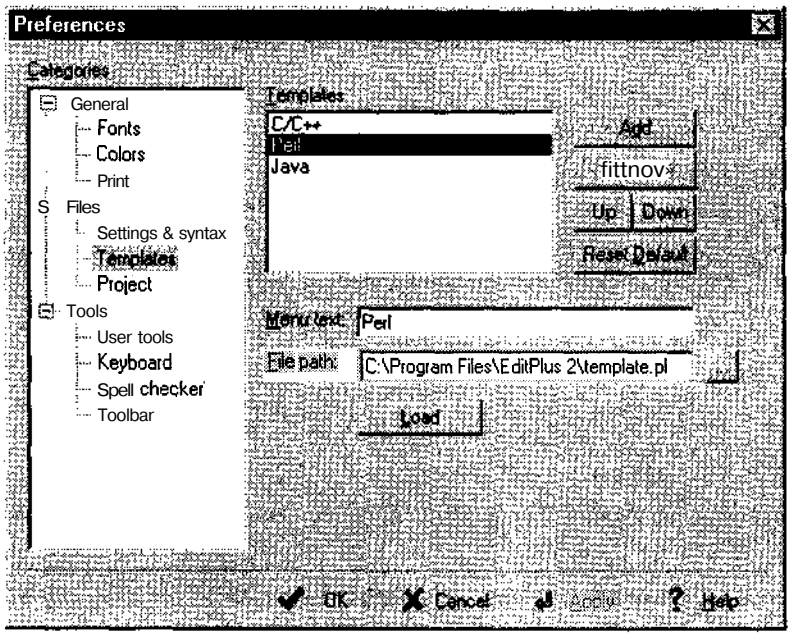


Рис. 4.5. Окно опции Configure Templates

- Open — открывает существующий документ;
- Close — закрывает ранее открытый документ;
- Close All — закрывает все открытые документы;
- Save — сохраняет документ, при этом используется то имя файла, которое было задано ранее;
- Save All — сохраняет все открытые документы;
- Save As — сохраняет файл под новым именем;
- Print — печатает документ;
- Print Preview — предварительный просмотр документа на экране;
- Print Setup — настройки принтера для печати;
- Print All — печатает все открытые файлы;
- FTP — работа с протоколом FTP (рис. 4.6):
  - FTP Upload — загружает файл по FTP на сервер. Данная опция является весьма полезной для работы с Интернетом;
  - Open Remote — открывает существующий документ, размещенный на удаленном сервере. Вам не понадобится даже производить запись документа,

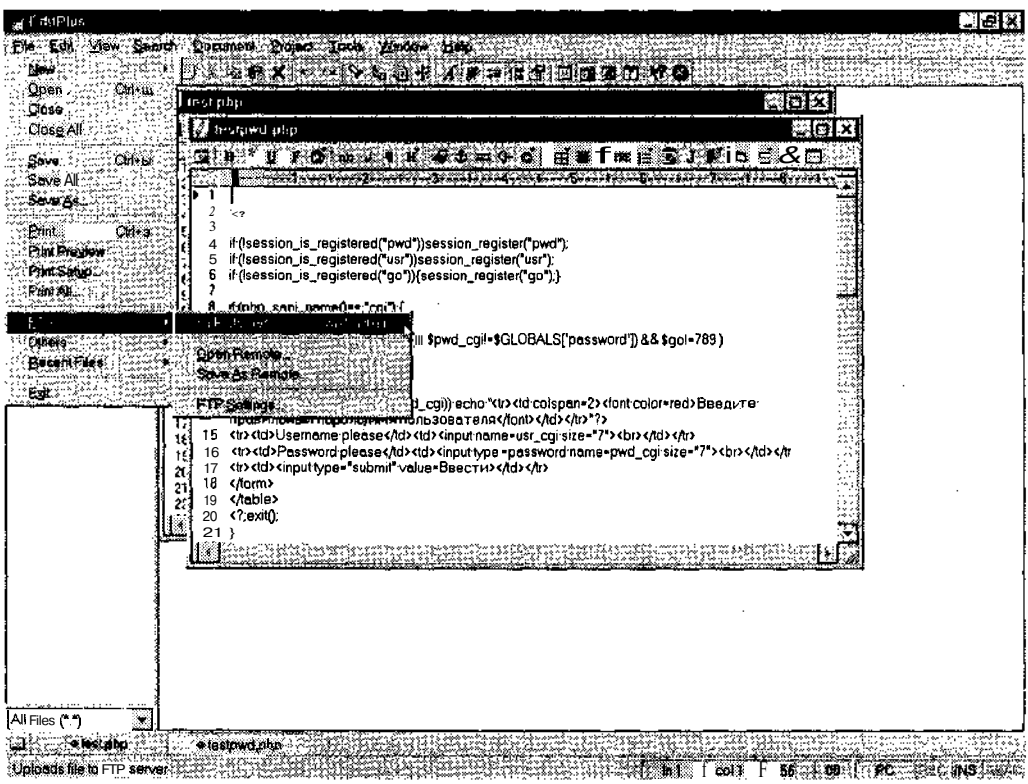


Рис. 4.6. Подменю опции FTP

так как при помощи этой опции можно прямо на месте исправить его, сохранить и сэкономить много времени;

- Save As Remote — сохраняет активный документ как удаленный FTP-файл;
- FTP Settings — настройки FTP: ваш пароль подключения, адрес узла и т. д. (рис. 4.7);

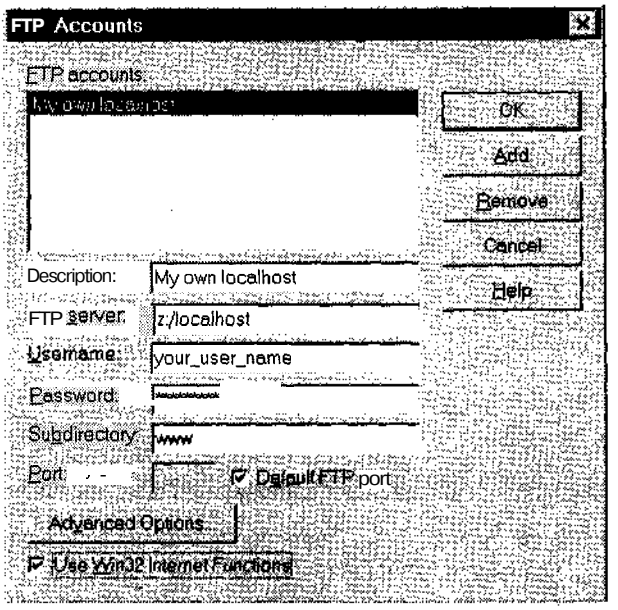


Рис. 4.7. Окно опции FTP Settings

- **Send** — отправляет документ при помощи электронной почты;
- **Recent Files** — список последних открытых файлов;
- **Exit** — выход из редактора.

4.4. Меню Edit

Меню **Edit** позволяет производить редактирование документа, с которым вы работаете в данный момент. Выделим те опции, которые, на наш взгляд, могут быть новыми для вас:

- **Duplicate Line** — дублирует текущую строку;
- **Format** — манипуляция с расположением выделенного текстового блока (рис. 4.8);
- **Change Case** — изменяет регистр выбранного блока текста (рис. 4.9). Существуют следующие варианты:
  - **Upper Case** — верхний регистр,
  - Lower Case — нижний регистр,

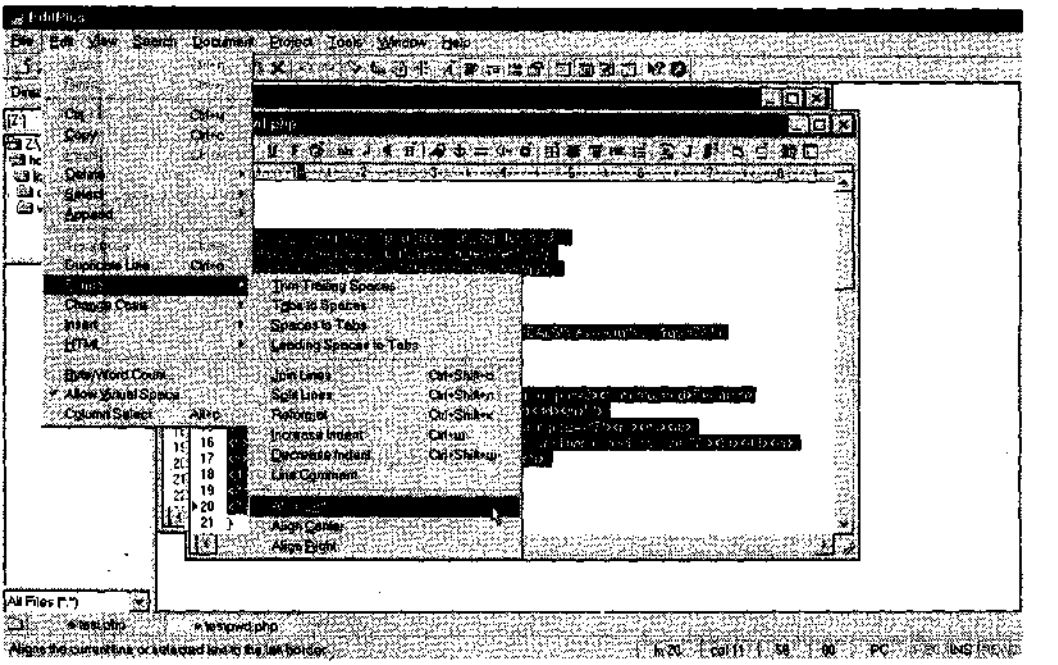


Рис. 4.8. Подменю опции Format

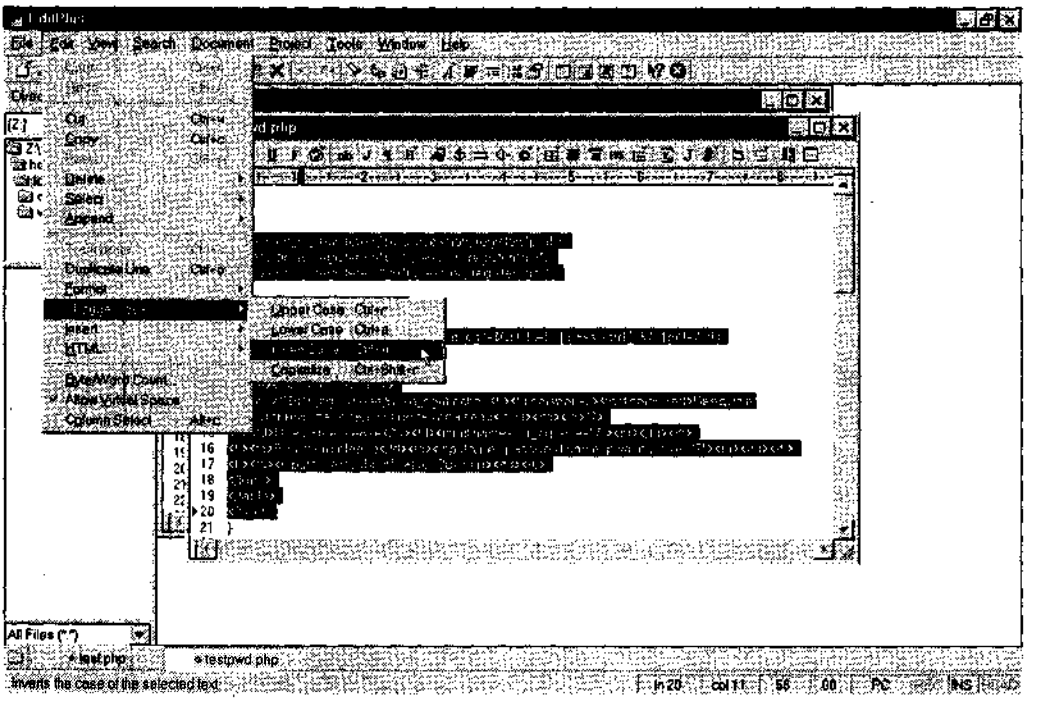


Рис. 4.9. Подменю опции Change Case

- Invert Case — изменяет регистр каждого символа на противоположный,
- Capitalize — верхний регистр выбранного текста;
- **Byte/Word** Count — показывает количество байтов и слов в выбранном блоке документа (рис. 4.10).

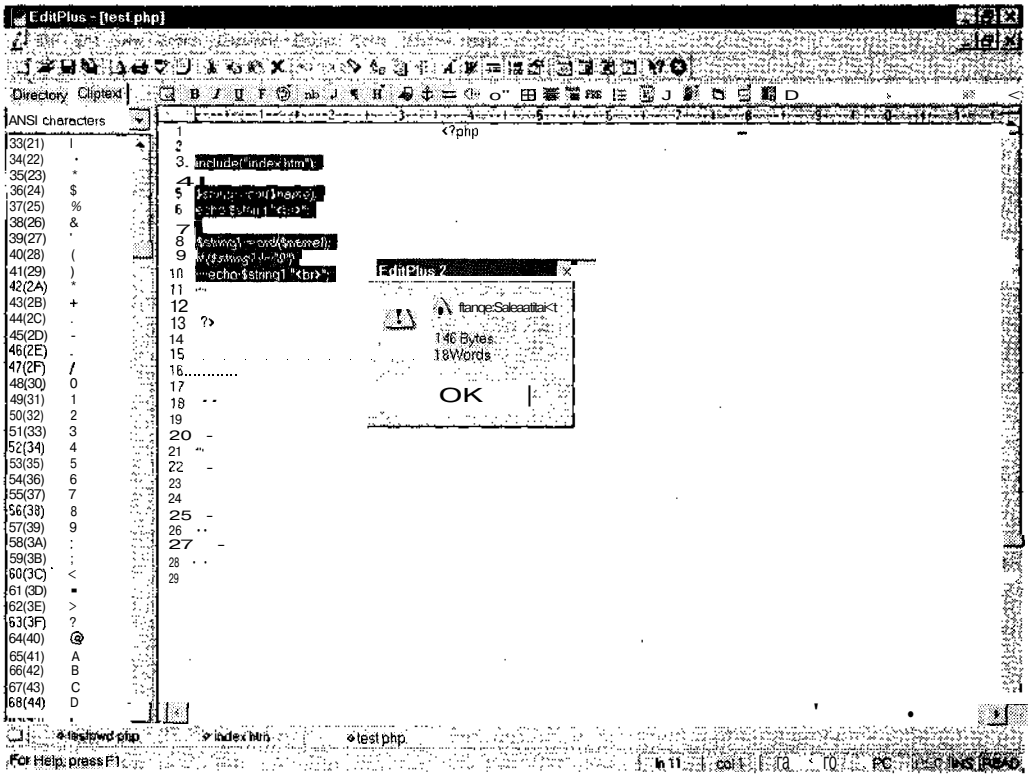


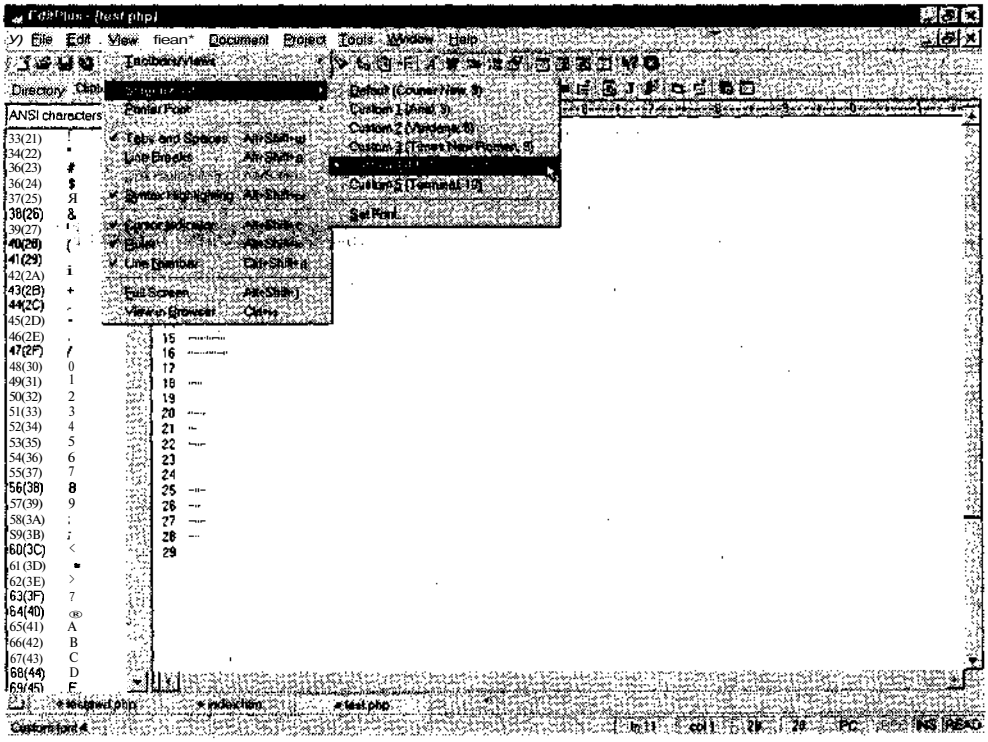
Рис. 4.10. Результат выполнения команды Byte/Word Count

4.5. Меню View

С помощью меню View можно просматривать и изменять различные параметры программы, например шрифт (рис. 4.11).

Данное меню содержит следующие опции:

- Toolbars/Views — показывает или скрывает различные панели инструментов (рис. 4.12);
- Screen Font — выбор шрифта для отображения на экране;
- Printer Font — выбор шрифта для печати;
- **Tabs** and **Spaces** — показывает или скрывает индикаторы и метки;
- Line Break — устанавливает или скрывает метку конца строки;
- URL Highlighting — устанавливает цветовое выделение URL и e-mail адресов;



Рис, 4.11. Меню View

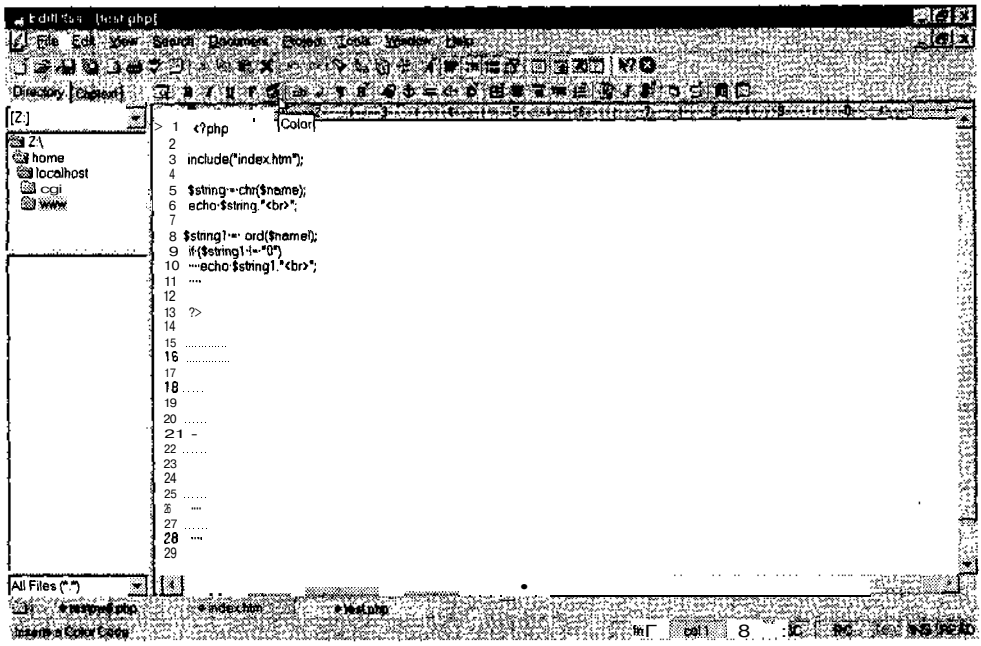


Рис. 4.12 Включенные панели инструментов

- Syntax Highlighting — устанавливает цветовое выделение синтаксиса;
- **Cursor Indicator** — устанавливает или скрывает индикатор курсора;
- **Line Number** — нумерация строк;
- **Full Screen** — полноэкранный режим работы;
- **View inBrowser** — просмотр активного документа во встроенном браузере (рис.4.13).

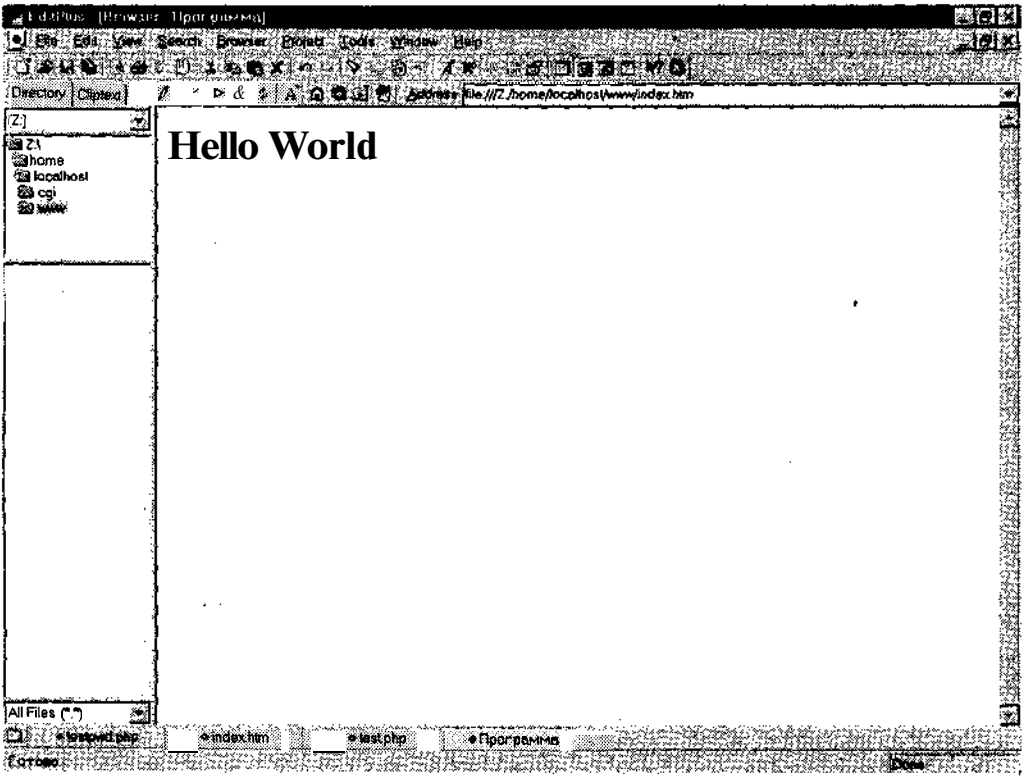


Рис. 4.13. Просмотр активного документа во встроенном браузере

4.6. Меню Search

При помощи меню **Search** можно производить поиск и замену словосочетаний, необходимых абзацев и слов в тексте (рис. 4.14).

Основные опции в меню **Search:**

- **Find** — поиск указанного текста. Указание задается в специальном окне (рис.4.15);
- **Replace** — заменяет местами текст;
- **Find** Next — поиск следующего ранее указанного текста;
- **Find Previous** — поиск предшествующего текста;
- **Find** Next **Word** — поиск следующего слова;

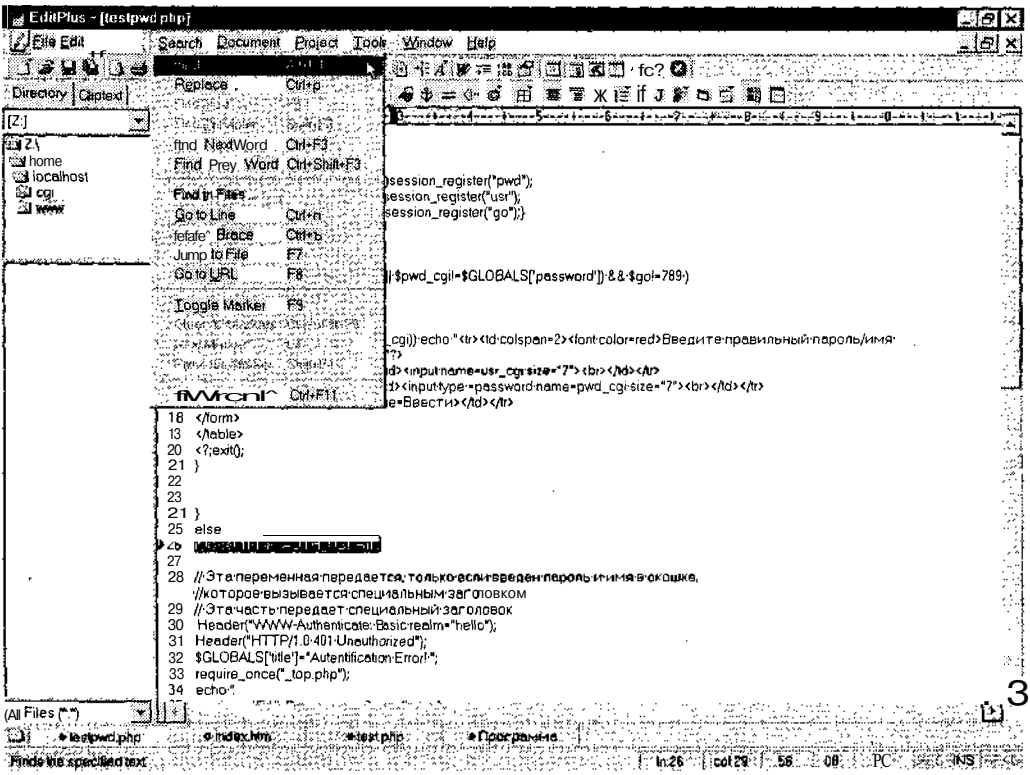


Рис. 4.14. Меню Search

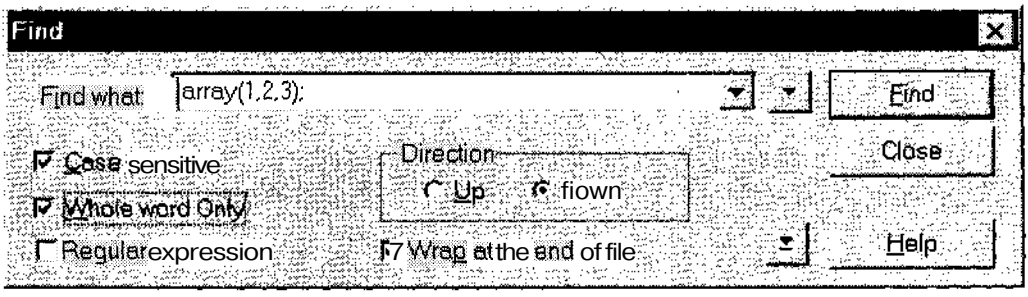


Рис. 4.15. Окно опции Find

- **Find Prev Word** — поиск предыдущего слова;
- **Find in Files** — поиск необходимого текста в указанных файлах;
- **Go to Line** — перемещение на необходимую строку в активном документе;
- **Toggle Marker** — установка и удаление метки активной строки. На рис.4.16 продемонстрировано, как устанавливаются метки и как они выглядят;
- **Function List** — показывает лист функций.

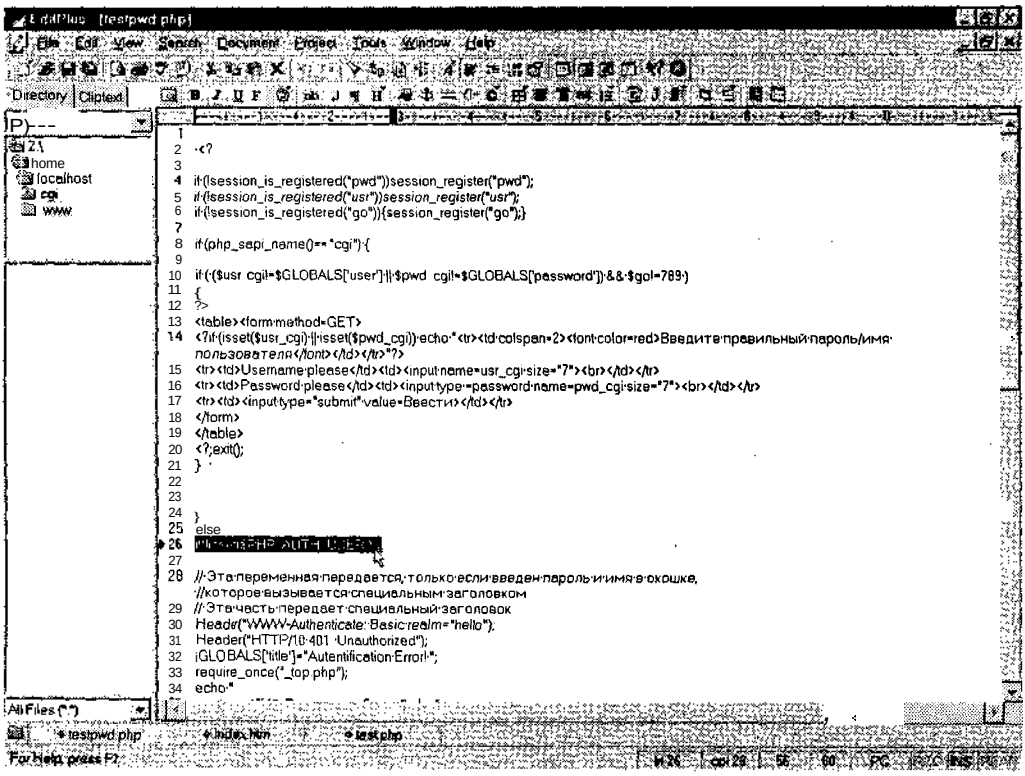


Рис. 4.16. Установка метки активной строки

4.7. Меню Document

Меню **Document** позволяет задавать различные настройки работы с документом (рис. 4.17). Это необходимо, так как есть параметры, которые не указываются в INI-файле, т. е. они являются периодически изменяемыми.

Меню **Document** состоит из следующих опций:

- **Word-wrap** — установка word-wrap особенностей активного документа. На рисунках приведены примеры с включенной (рис. 4.18) и выключенной (рис. 4.19) опцией word-wrap. Обратите внимание на расположение выделенного абзаца текста;
- **Auto Indent** — автоматическая установка отступов активного документа;
- **Auto Complete** — установка автозавершения активного документа;
- **Word-warp Options** — установка опций word-wrap;
- **Tab/Indent** — установка шага табуляции (рис. 4.20);
- Column **Marker** — разбивает на колонки:
  - **Set Column Marker** — установка маркера колонки. После задания параметров вы увидите вертикальные прозрачные линии;

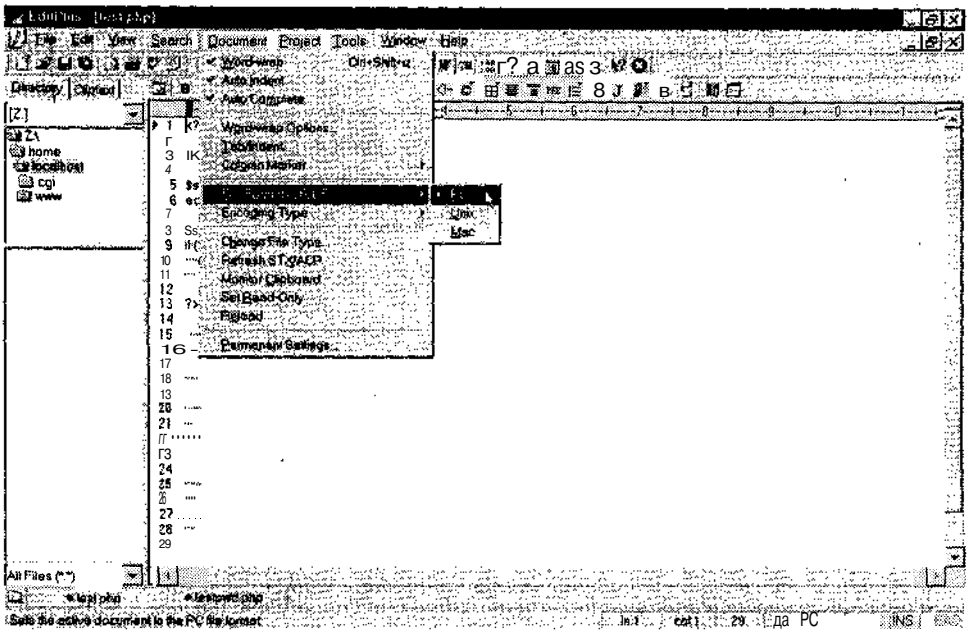


Рис. 4.17. Меню Document

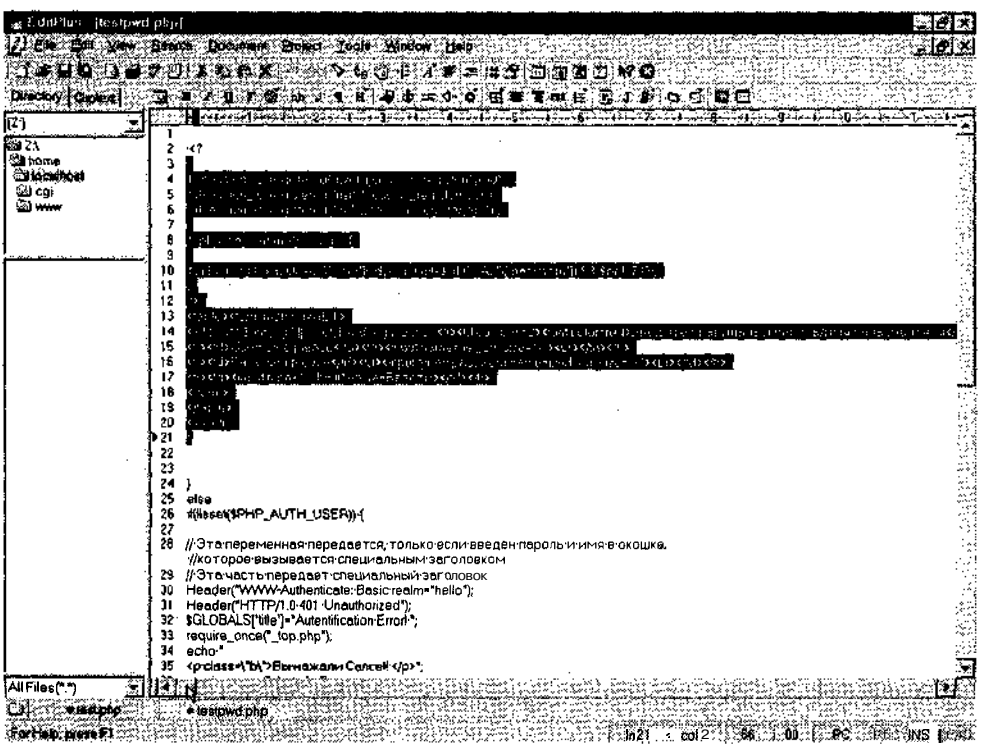


Рис. 4.18. Код со включенной опцией word-wrap

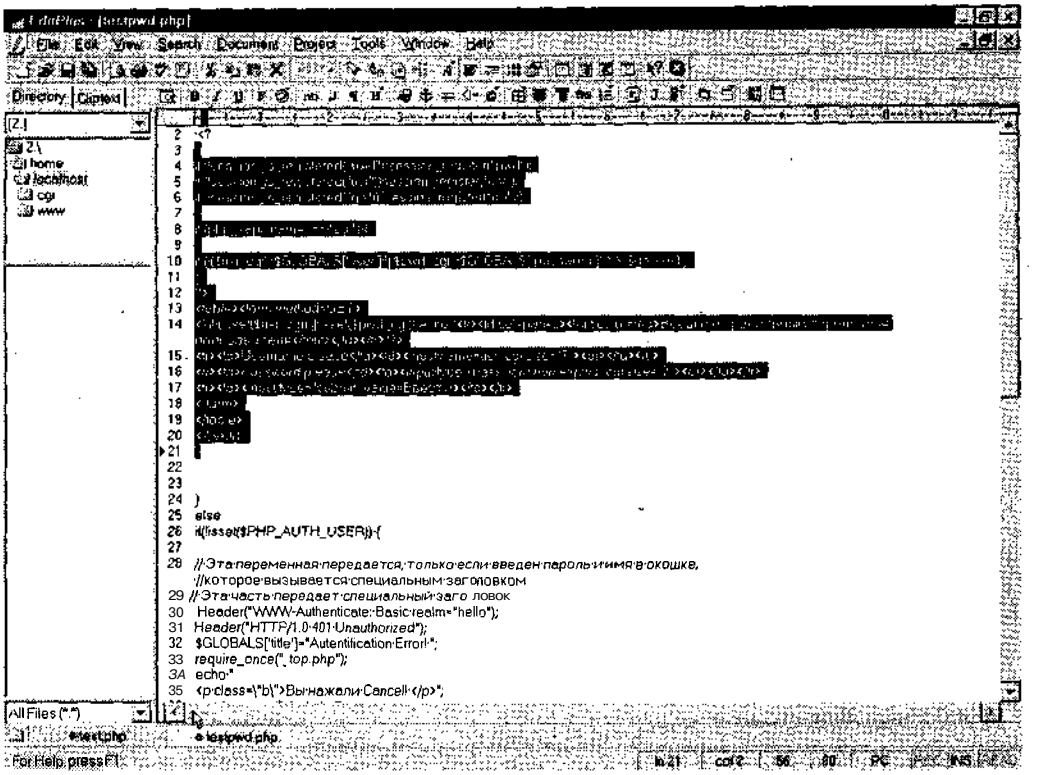


Рис. 4.19. Код с выключенной опцией word-wrap

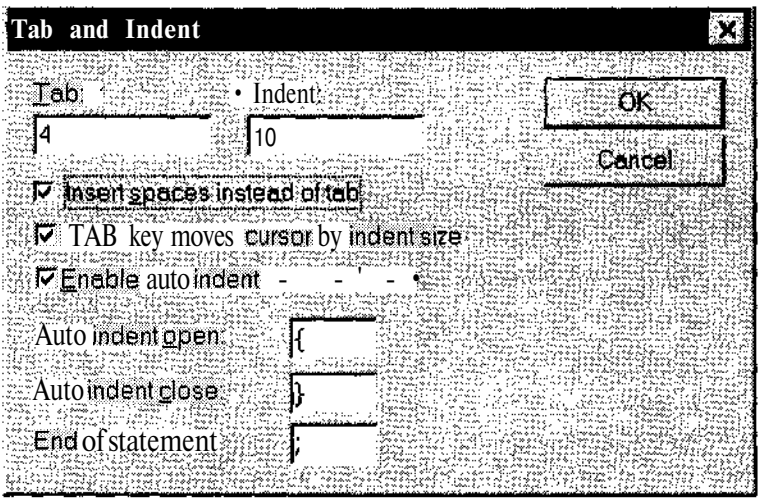


Рис. 4.20. Окно опции Tab/Indent

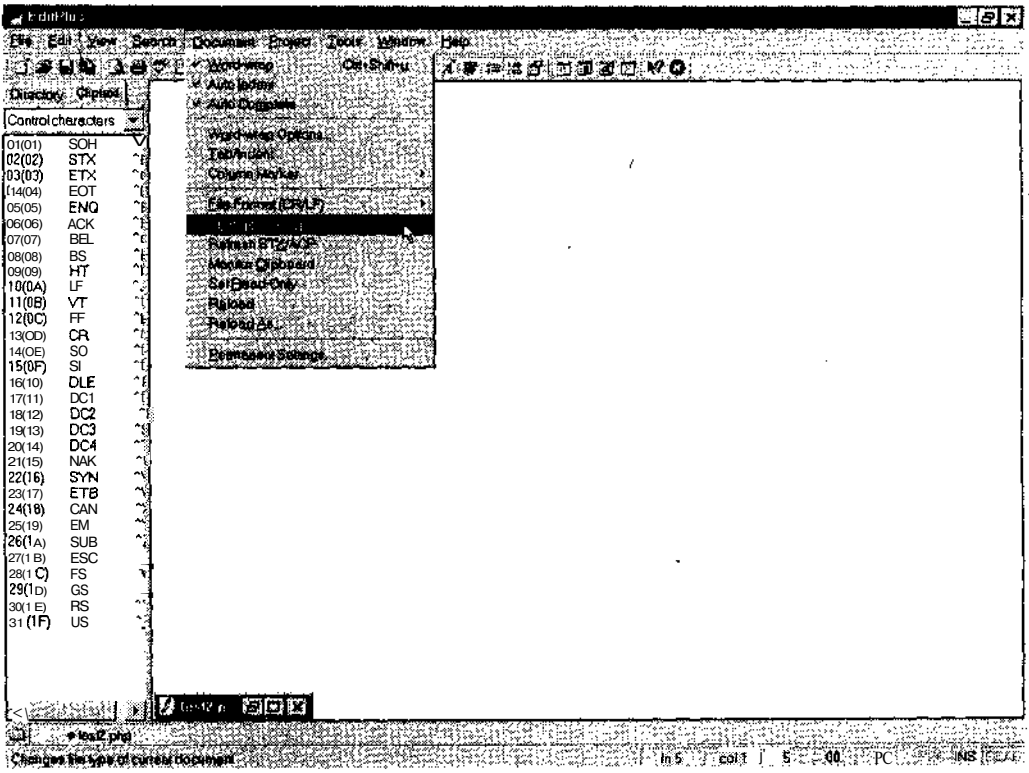


Рис. 4.21. Окно опции Change File Type

- **Show Column Marker** — показывает и скрывает метки колонки;
- **Next Column Marker** — переход на следующую позицию колонки;
- **File Format** — запись активного документа в указанный формат файла;
- Encoding**Type — установка** кодировки текста;
- **Change File Type** — изменение типа файла (рис. 4.21);
- **Refresh STX/ACP** — обновление синтаксиса файла;
- **Set Read-Only** — установка текущему документу опции «только чтение»;
- **Reload** — перезагрузка текущего документа с диска;
- **Permanent Settings** — установка параметров (рис. 4.22). Данная опция позволяет изменять настройки редактора так, как необходимо именно вам: задавать цвет или же параметры шрифта программы и т. д. Разобраться в ней не составит для вас труда.

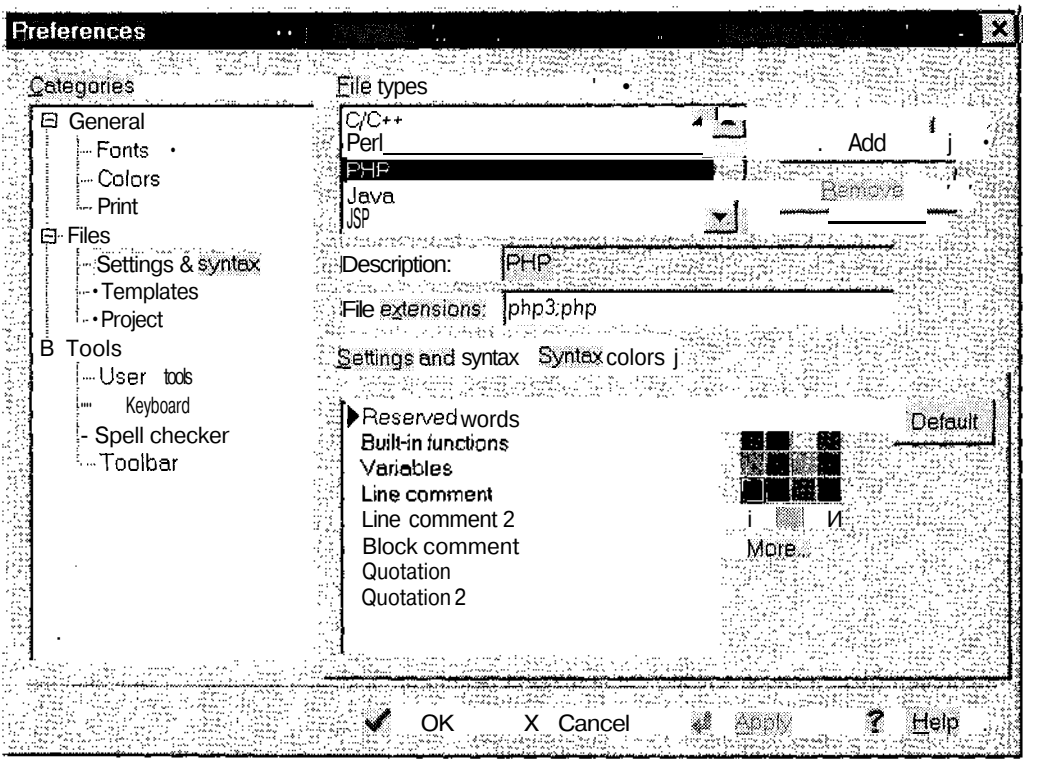


Рис. 4.22. Окно опции Permanent Settings

4.8. Меню Project

Как и все передовые редакторы, EditPlus осуществляет работу с проектами. Это очень важная и иногда весьма необходимая особенность. На рис. 4.23 представлено меню **Project.**

Меню **Project** имеет следующие опции:

- **Select Project** — выбор текущего проекта:
  - **Add to Project** — добавление текущего документа в проект;
  - **Add All to Project** — добавление всех открытых документов в проект;
  - **Edit Project** — установка параметров проекта (рис. 4.24). Задание параметров проекта осуществляется при помощи следующих опций:

    Projects — содержит зарегистрированные проекты;

    Add — добавляет проект;

    Remove — удаляет выделенные проекты.

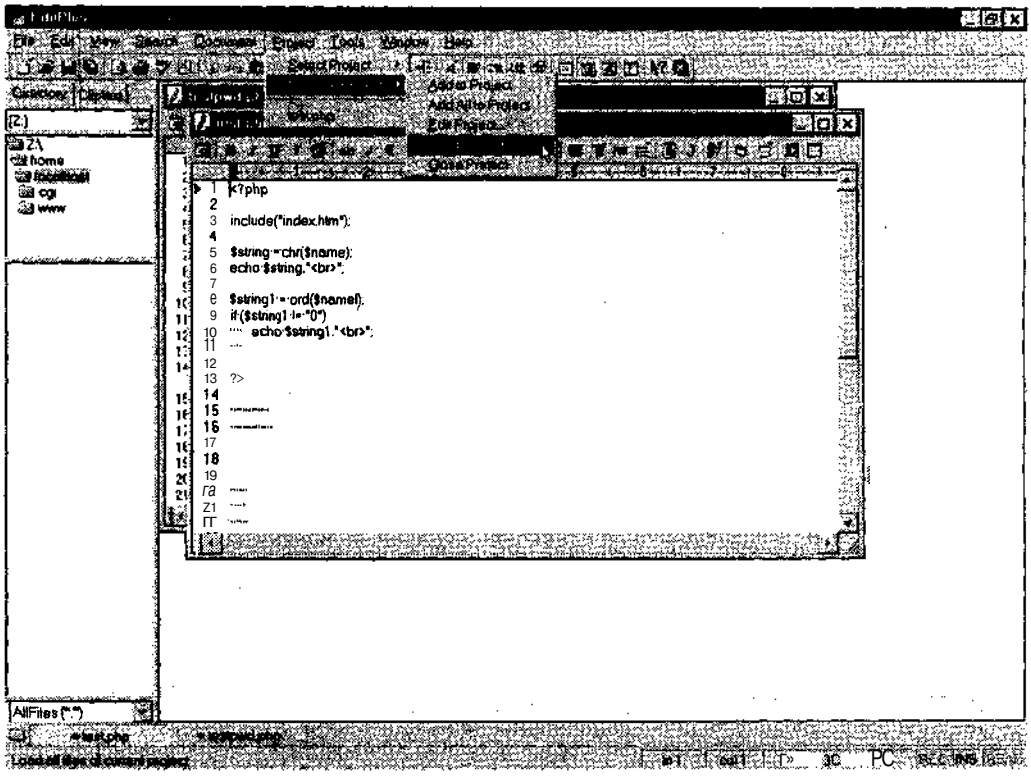


Рис. 4.23. Меню Project

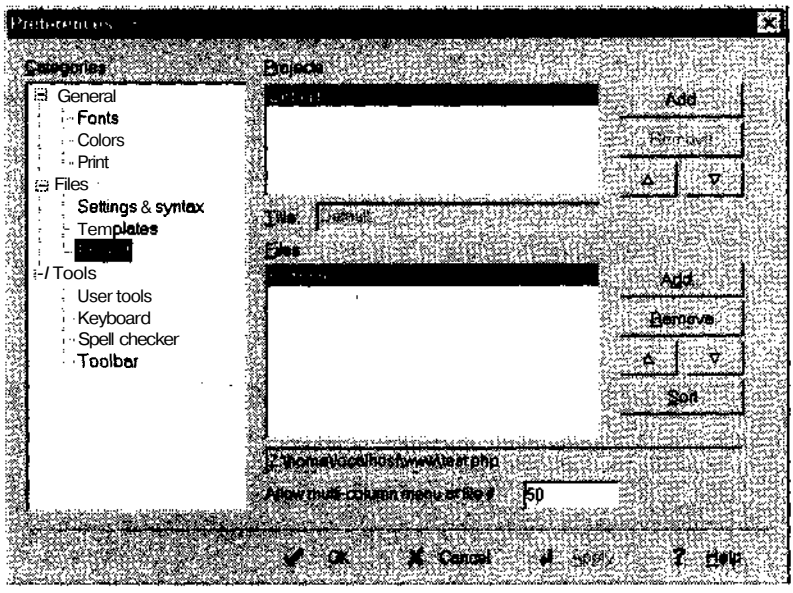


Рис. 4.24. Установка параметров проекта

ВНИМАНИЕ

Если используется несколько проектов, то на проект с параметром Default не будут распространяться установки других проектов.

Up — передвижение вверх по проектам;

Down — передвижение вниз по проектам;

Titles — описание выбранного проекта, которое будет показано в пункте меню;

Add (File) — добавить выбранный файл в текущий проект;

Remove (File) — удаление выбранного файла из проекта;

Sort — сортировка листа файлов по имени;

Allow multi-column menu at file # — показывает меню проекта в колонках в случае, когда количество файлов превышает заданную величину. Эта опция полезна, когда имеется так много файлов в проекте, что меню не может соответствовать высоте экрана (рис. 4.25);

- **Load Project** — чтение всех файлов текущего проекта;
- **Close Project** — закрывает все файлы текущего проекта;
- **Project File** — чтение одного файла текущего проекта.

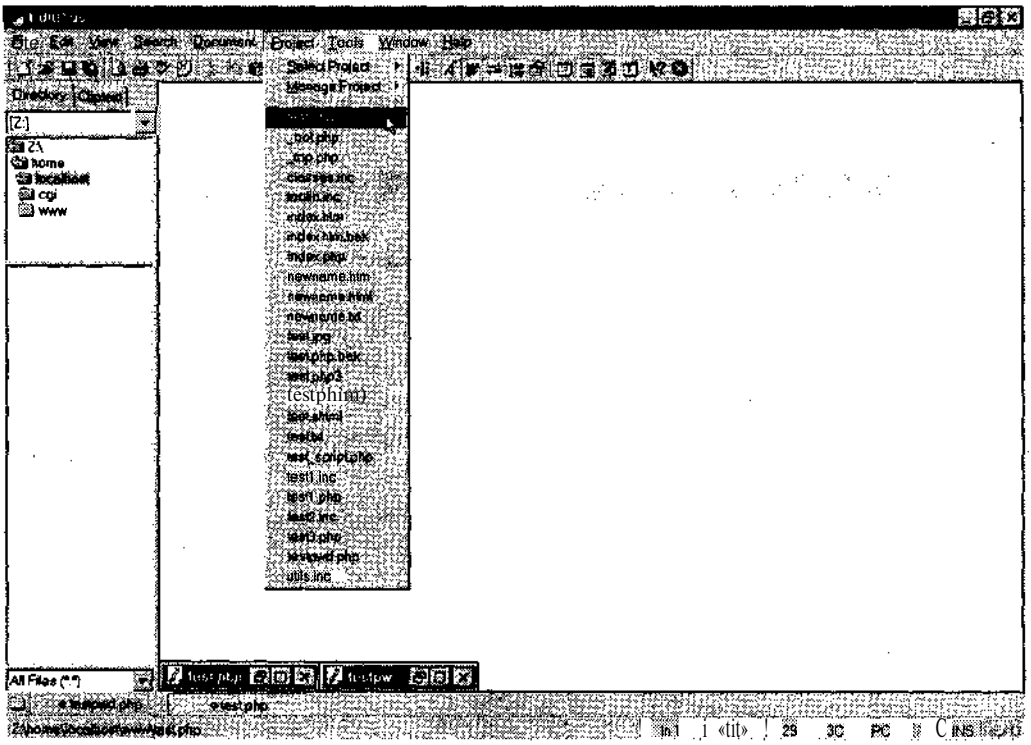


Рис. 4.25. Использование опции Allow multi-column menu at file #

4.9. Меню Tools

Меню Tools содержит различные инструменты, помогающие при создании программ (рис. 4.26).

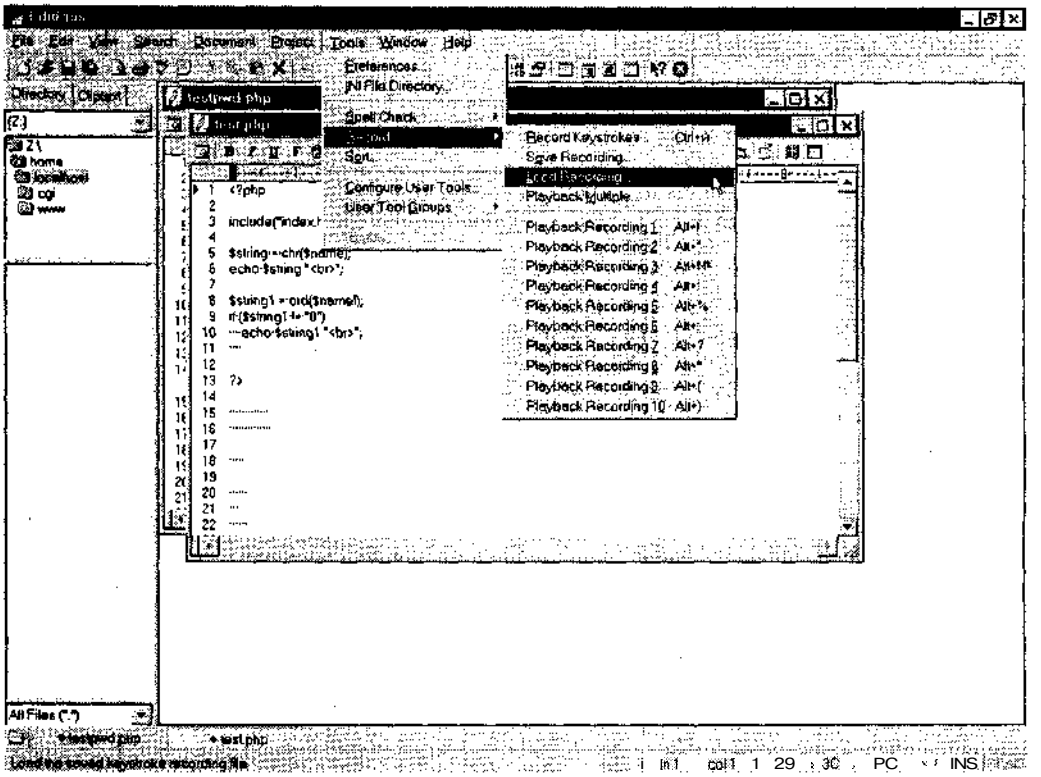


Рис. 4.26. Меню Tools

Меню Tools содержит следующие опции:

- **Preferences** — настройка параметров;
- **INI File Directory** — установка директории файла конфигурации (*.ini) (рис. 4.27);

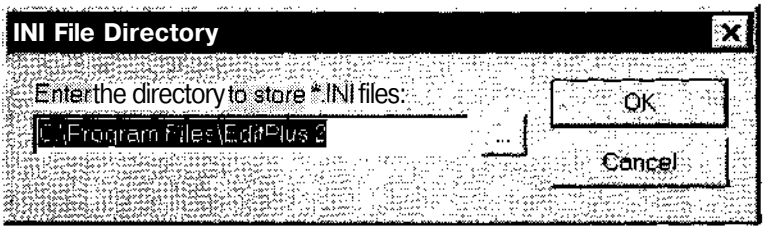


Рис. 4.27. Окно опции INI File Directory

- **Spell Check** — проверка орфографических ошибок в активном документе:
  - **Spell Check from Current Position** — проверка ошибок с текущей позиции;
  - **Spell Check Selection** — проверка ошибок в текущем выделении;
  - Spell Check Word — проверка ошибок в выбранном слове;
- **Record** — установка комбинаций клавиш;
- **Sort** — сортировка документа или выбранного текста;
- **Configure User Tools** — добавление или удаление инструментов (рис. 4.28);

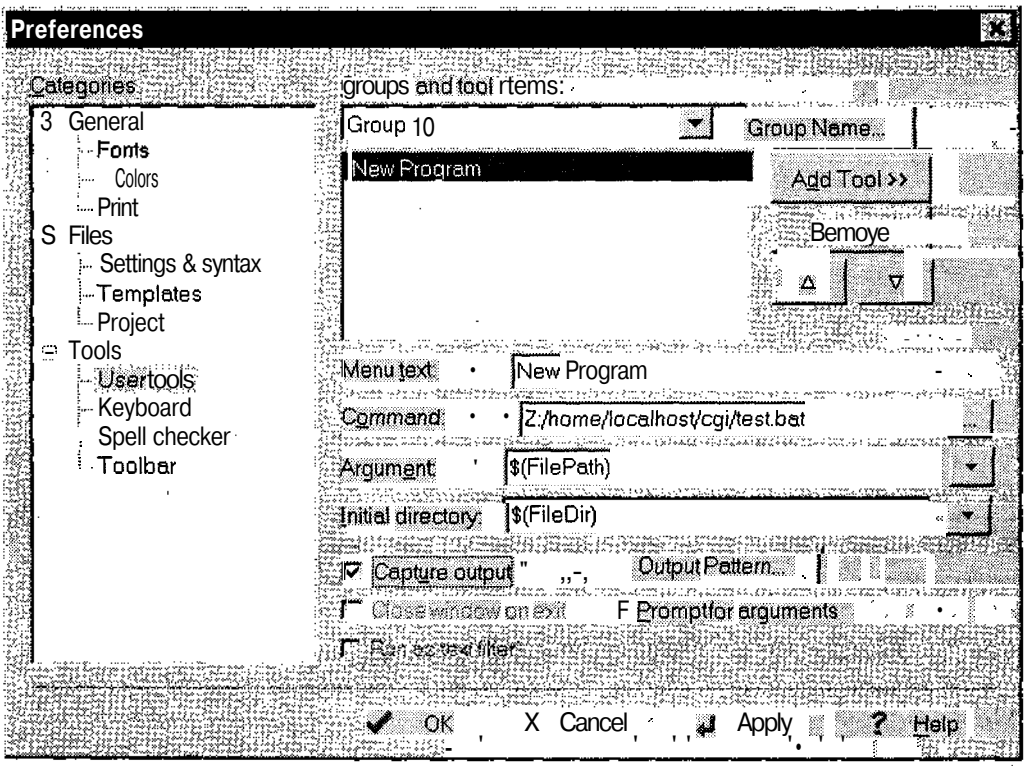


Рис. 4.Ж. Окно опции Configure User Tools

- **User Tool Groups** — выбор группы инструментов;
- **User Tools** — вызов инструментов.

4.10. Меню Window

Меню Window (рис. 4.29) позволяет выбирать, как окна будут располагаться в окне редактора.

Варианты здесь аналогичны тем, которые вы используете, размещая окна в диспетчере программ. Можно выбирать между **Cascade, Title Vertical, Title Horizontal** или

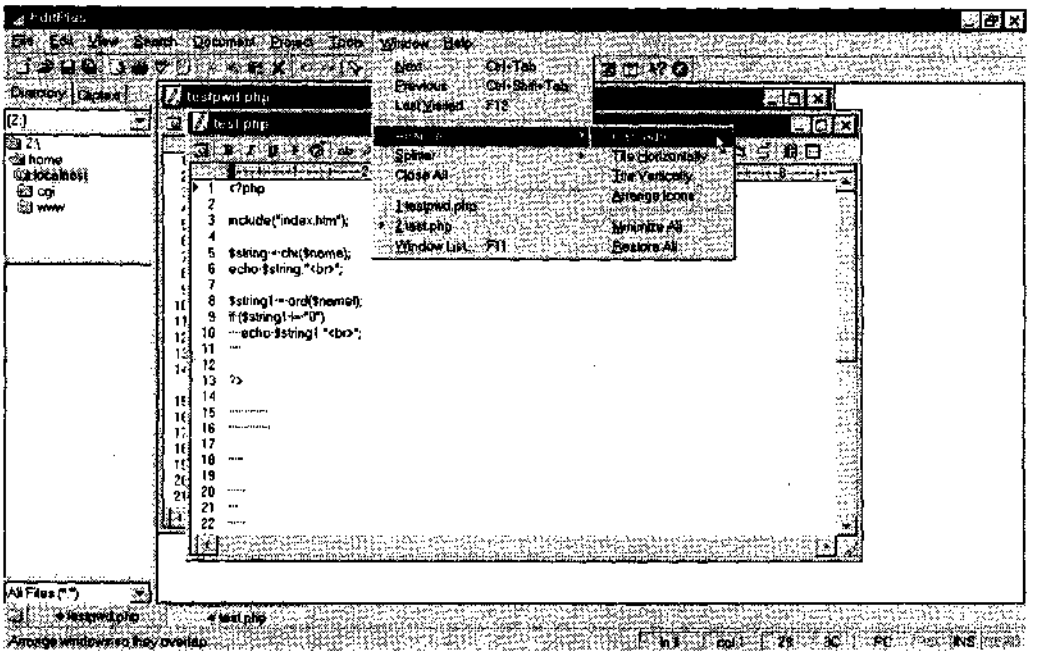


Рис. 4.29. Меню Window

Arrange Icons. Для удобства добавлены **Restore All и Minimize All.** В конце меню **Window** содержится список открытых файлов, между которыми можно переключаться щелчком мыши на соответствующем имени. Также необходимо заметить такую опцию, как **Window List.** Она позволяет получить список всех открытых файлов редактора и произвести некоторые преобразования над ними (рис. 4.30).

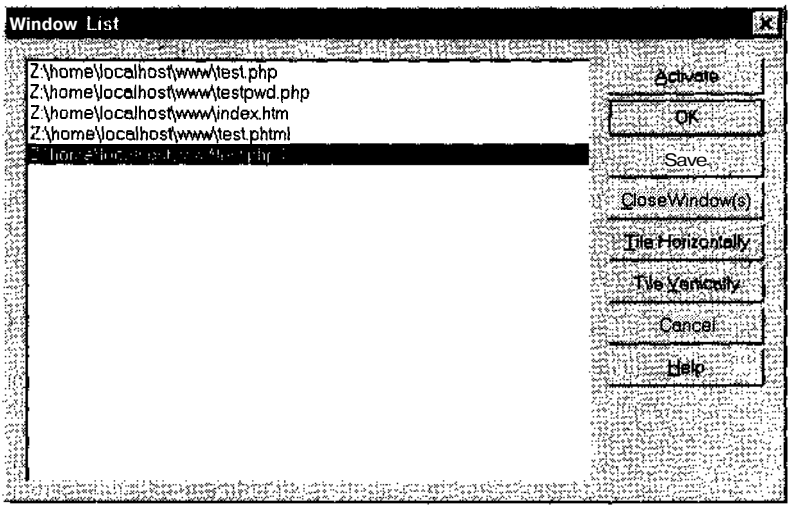


Рис. 4.30. Окно опции Window List

4.11. Меню Help

Пункт меню **Help** находится в самом конце строки меню. Данная опция обеспечивает интерактивную подсказку по всем аспектам редактора EditPlus. Здесь можно найти полную информацию по работе с самим редактором и его оптимальной настройке. Используйте этот пункт меню при ознакомлении с редактором.

4.12. Панель инструментов

Панель инструментов представляет из себя ряд кнопок, расположенных под строкой меню и обеспечивающих быстрый доступ к любым операциям редактора. Панель инструментов может быть настроена на выполнение гораздо большего числа функций, чем установлено по умолчанию, и наоборот, вы можете исключить из нее кнопки, соответствующие редко используемым действиям. На рис. 4.31 курсор стоит на иконке New.

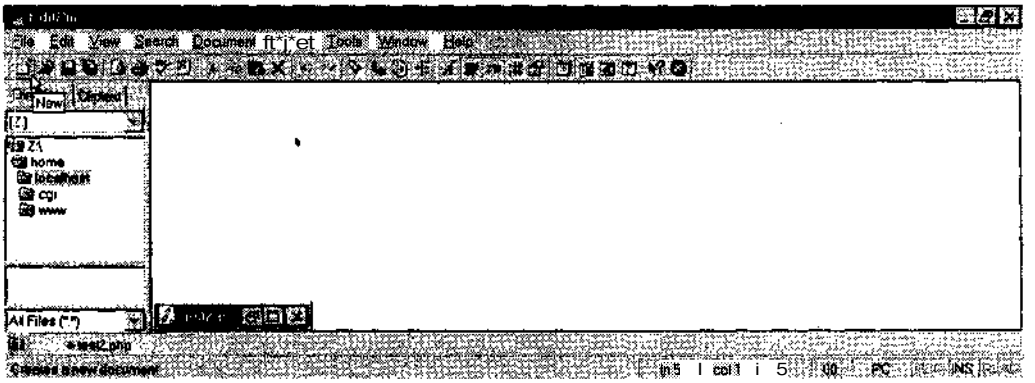


Рис. 4.31. Панель инструментов редактора EditPlus

СОВЕТ

Заметьте, что в нижней части окна редактора EditPlus располагается информационная строка, кратко разъясняющая назначение каждой кнопки.

Заключение

В этой главе вы узнали, как настраивать и использовать редактор EditPlus. Вы получили представление о командах различных меню и панели инструментов, которые позволяют вам производить любые операции с проектами, в том числе создавать собственные и настраивать их, согласно вашим желаниям. Вы научились манипулировать шрифтами и установками программ. Это поможет вам без проблем освоить другие аналогичные редакторы и производить их настройку соответственно собственным вкусам.

Глава 5
Конфигурация

При запуске синтаксического анализатора PHP происходит чтение файла конфигурации, который содержит основные настройки работы интерпретатора. Для версии PHP 3.0 файл называется `php3.ini`, PHP 4 — `php.ini`. Эти файлы несут очень важную информацию, и умением их настраивать вы не должны пренебрегать. Файл конфигурации запускается только один раз при запуске Web-сервера (для версии серверного модуля). Для версии CGI это случается при каждом вызове интерпретатора.

Для каждой директивы, указанной в файле `php3.ini/php.ini`, имеется соответствующая директива Apache в файле `httpd.conf`. Для получения полной информации о большинстве установочных параметров значений конфигурации вы можете воспользоваться функцией `phpinfo()`.

Из этой главы вы узнаете:

- основные директивы конфигурации;
- директивы конфигурации почты;
- директивы конфигурации Safe Mode;
- директивы конфигурации отладчика;
- директивы загрузки расширений (Extension Loading);
- директивы конфигурации MySQL;
- директивы конфигурации mSQL;
- директивы конфигурации Postgres;
- директивы конфигурации Sybase;
- директивы конфигурации унифицированных ODBC;
- директивы конфигурации модуля Apache.

5.1. Основные директивы конфигурации

Как было ранее сказано, файл конфигурации является средством, которое контролирует многие аспекты поведения PHP. В зависимости от версии PHP он может называться `php.ini`, `php3.ini` или по-другому. Интерпретатор PHP ищет эти файлы в текущей рабочей директории, т. е. в пути, обозначенном системной переменной PHPPC, и в пути, определенном в процессе компиляции. Для операционной системы Windows путь во время компиляции устанавливается как путь к каталогу Windows (например, `C:/Windows`).

Синтаксис файла чрезвычайно прост: строка и необходимая конфигурация, установленная пользователем, либо по умолчанию соответствующее ей значение. Если

строка начинается с точки с запятой, то все, что идет после, игнорируется. Квадратные скобки — [ ] — (например, [mail function], [java] и др.) также игнорируются, несмотря на то что они могли бы значить что-то в будущем.

Директивы определяются использованием следующего синтаксиса:

```
Directive = value
```

Направляющие названия зависят от регистра — java=bar отличается от JAVA=bar.

Переменная может обозначаться: строка, номер, PHP-константа (например, E_ALL или P_PI), одной из INI-констант (on, off, true, false, yes, no и none) либо выражением (например, E_ALL & ~E_NOTICE) или цитируемой строкой («java»).

Выражения в INI-файлах ограничены поразрядными операторами и круглыми скобками:

|	Поразрядный OR (или)
&	Поразрядный AND (и)
~	Поразрядный NOT (не)
!	Булевый NOT (не)

Булевые флаги можно включить, используя значения 1, on, true или yes. Соответственно, их можно выключить, используя значения 0, off, false или no. Пустая строка может быть обозначена, если вы не запишете ничего после знака «равно» или воспользуетесь ключевым словом none:

foo = ; устанавливает foo значение пустой строки,

foo = none устанавливает foo значение пустой строки,

foo = "none" устанавливает foo значение "none".

Если используемые значения константы принадлежат динамически загружаемым распространителям (возможно, PHP-распространителям или Zend-расширениям), вы можете использовать эти константы только после after — строки, которая читает расширения.

Все значения в файле php.ini строго соответствуют значениям по умолчанию (т. е. php.ini не используется, а при удалении каких-либо строк значения по умолчанию будут идентичны).

Опишем наиболее важные директивы, используемые в файле php.ini:

```
auto_append_file string
```

определяет имя файла, который автоматически проверяется после основного файла. При вызове функции include() с использованием include_path можно также включить этот файл. Специальное значение none запрещает автодобавление.

ВНИМАНИЕ

Если сценарий обрывается функцией exit(), автодобавление не произойдет.

`auto_prepend_file string`

определяет имя файла, который автоматически проверяется перед основным файлом. Файл включается так же, как если бы была вызвана функция include() с использованием include_path. Специальное значение none запрещает автодобавление.

`cgi_ext string`

`display_errors boolean`

определяют, должны ошибки печататься на экране как часть HTML-вывода или нет.

`doc_root string`

главный каталог (root directory) PHP на сервере. Используется, только когда он не пустой. Если PHP сконфигурирован при помощи safe mode, то никакие другие файлы за пределами этого каталога не обслуживаются.

`engine boolean`

эта директива действительно полезна только в модуле PHP под Apache. Используется на сайтах, где необходимо включать и выключать синтаксический анализ PHP на определенные каталоги и виртуальные серверы. Установкой php3_engine off в допустимом месте файла httpd.conf анализатор PHP можно запрещать и разрешать.

`error_log string`

имя файла, с которым сохраняется журнал ошибок сценариев (log file). Если используется специальное значение syslog, ошибки отправляются в системный журнал (system logger). В UNIX это syslog (3), а в Windows NT это журнал событий (event log). Системный журнал не поддерживается в Windows 95.

`error_reporting integer`

устанавливает уровень сообщений об ошибках. Параметр является целым, представляющим битовую область. Добавьте те значения уровней сообщений об ошибках, которые вы хотите (табл. 5.1).

Таблица 5.1. Уровни сообщений об ошибках

Значения в битах	Допустимые сообщения
1	Нормальные ошибки
2	Нормальные предупреждения
4	Ошибки синтаксического анализа
7	Нормальные ошибки, нормальные предупреждения и синтаксические ошибки
8	Некритичные предупреждения стиля

```
open_basedir string
```

ограничивает файлы, которые могут открываться PHP в определенной директории.

Когда сценарий пытается открыть файл с помощью, например, fopen() или gzopen(), наличие/расположение файла проверяется. Когда файл находится за пределами определенной директории, PHP откажется открыть его. Все символьные ссылки определены, так что нет возможности избежать этого ограничения с помощью symlink. Специальное значение показывает, что каталог, в котором находится сценарий, используется как основной каталог. По умолчанию допускается открытие всех файлов.

```
gpc_order string
```

устанавливает допустимость Get/Post/Cookie в анализе. Установка по умолчанию этой директивы — «GPC». Установка ее в «GP», например, вынудит PHP полностью игнорировать Cookies и перезапишет любые переменные метода Get переменными метода Post с одинаковыми именами.

```
include_path string
```

определяет список каталогов, в котором будут располагаться файлы для функций require(), include() и fopen_with_path(). Формат подобен формату системной переменной окружения PATH: список каталогов разделяется двоеточием в UNIX или точкой с запятой в Windows. Например:

```
UNIX include_path
include_path=.:/home/httpd/php-library
Windows include_path
include_path=".;c:\www\phplibrary"
```

Значением по умолчанию этой директивы является только текущий каталог.

```
isapi_ext string
```

```
log_errors boolean
```

обозначают, что независимые сообщения об ошибке сценария должны регистрироваться в журнале ошибок сервера. Этот выбор является специфичным для сервера.

```
magic_quotes_gpc boolean
```

устанавливает состояние magic_quotes для GPC (Get/Post/Cookie) операций. Когда magic_quotes включено (on), все ' (одиночные кавычки), " (двойные кавычки), \ (обратные слеши) и нулевые значения записываются с обратной косой чертой автоматически. Если также включена magic_quotes_sybase, одиночная кавычка записывается с дополнительной одиночной кавычкой вместо обратной косой черты.

magic_quotes_runtime boolean

если magic_quo'tes_runtime разрешена, большинство функций, которые возвращают данные из любого внешнего источника разной природы, включая базы данных и текстовые файлы, будут иметь кавычки, записанные с обратной косой чертой. Если magic_quotes_sybase также включены, одиночная кавычка записывается с дополнительной одиночной кавычкой вместо обратной косой черты.

magic_quotes_sybase boolean

если magic_quotes_sybase **включена, при включенных** magic_quotes_gpc **или** magic_quotes_runtime, **то одиночная кавычка записывается с дополнительной одиночной кавычкой вместо обратной косой черты.**

max_execution_time integer

определяет максимальное время в секундах, допустимое для сценария, прежде чем он будет прекращен анализатором. Это помогает защититься от некорректно написанных сценариев.

memory_limit integer

определяет максимальный размер памяти в байтах, допустимый для этого сценария. Это помогает запретить некорректно написанным сценариям использовать всю доступную память на сервере.

nsapi_ext string

short_open_tag boolean

задают допустимость короткой формы тегов PHP (<? ?>). Если вы хотите использовать PHP совместно с XML, эту опцию необходимо отключить. При этом вы должны использовать длинную форму тегов (<?php ?>).

sql.safe_mode boolean

track_errors boolean

если опции разрешены, последнее сообщение об ошибке всегда будет представлено в глобальной переменной $php_errormsg.

track_vars boolean

если опция разрешена, входящая информация Get, Post и Cookie может быть найдена в глобальных ассоциативных массивах $HTTP_GET_VARS, $HTTP_POST_VARS и $HTTP_COOKIE_VARS, соответственно.

upload_tmp_dir string

временный каталог, используемый для хранения файлов при их загрузке на сервер. Должен допускать запись независимо оттого, каким пользователем PHP применяется.

`user_dir string`

основное имя каталога, используемого в домашнем каталоге пользователей для файлов PHP, например `public_html`.

`warn_plus_overloading boolean`

если разрешено, опция выдает при выводе PHP сообщение, когда оператор плюс (+) используется в строке. Это облегчает поиск сценариев, которые должны быть перезаписаны с заменой на (.).

5.2. Директивы конфигурации почты

Данные директивы предназначены для настройки работы PHP с почтовыми функциями. Смысл каждой из них можно понять из названия, мы приведем наиболее распространенные директивы.

`SMTP string`

имя DNS или IP-адрес сервера SMTP, который должен использоваться PHP под Windows для отправки сообщения функцией `mail()`.

`sendmail_from string`

определяет «From:» — почтовый адрес, который используется в сообщении, отправленном PHP под Windows.

`sendmail_path string`

указывает, где располагается программа `sendmail`, обычно это `/usr/sbin/sendmail` или `/usr/lib/sendmail`.

Конфигурация обычно определяет это за вас и устанавливает значение по умолчанию, но в случае неудачи вы можете установить это здесь.

Системы, не использующие sendmail, должны установить оболочку/замену `sendmail` другой системой почты, если имеется. Например, пользователи могут указать `Qmail /var/qmail/bin/sendmail`.

5.3. Директивы конфигурации Safe Mode

`safe_mode boolean`

устанавливает допустимость PHP safe mode (см. гл. 6).

`safe_mode_exec_dir string`

если PHP используется в `safe` mode, `system()` и другие функции, выполняющие системные программы, отказываются запускать программы, которые находятся не в этом каталоге.

5.4. Директивы конфигурации отладчика

`debugger.host string`

DNS-имя или IP-адрес хоста, используемого отладчиком.

`debugger.port string`

номер порта, используемого отладчиком.

`debugger.enabled boolean`

задает допустимость использования отладчика.

5.5. Директивы загрузки расширений (Extension Loading)

`enable_dl boolean`

полезна только в модуле PHP под Apache. Вы можете разрешать/запрещать динамическую загрузку расширений PHP-функцией `dl()` раздельно по каталогам и/или по виртуальным серверам.

Основная причина для выключения динамической загрузки — безопасность. С динамической загрузкой можно игнорировать все ограничения `safe_mode` И `open_basedir`.

По умолчанию динамическая загрузка должна допускаться, за исключением случаев, когда используется safe mode. В safe mode всегда недопустимо использование `dl()`.

`extension_dir string`

определяет, в каком каталоге PHP должен искать динамически загружаемые расширения.

`extension string`

показывает, какие динамически загружаемые расширения загрузить при запуске PHP.

5.6. Директивы конфигурации MySQL

`mysql.allow_persistent boolean`

допускает постоянные устойчивые MySQL-соединения.

`mysql.max_persistent integer`

максимальное число постоянных MySQL-соединений за один процесс.

`mysql.max_links integer`

максимальное число MySQL-соединений за процесс, включая постоянные соединения.

5.7. Директивы конфигурации mSQL

`msql.allow_persistent` boolean

допускает постоянные mSQL-соединения.

`msql.max_persistent` integer

максимальное число постоянных mSQL-соединений за один процесс.

`msql.max_links` integer

максимальное число постоянных mSQL-соединений за один процесс.

5.8. Директивы конфигурации Postgres

`pgsql.allow_persistent` boolean

допускает постоянные устойчивые соединения Postgres.

`pgsql.max_persistent` integer

максимальное число постоянных соединений Postgres за процесс.

`pgsql.max_links` integer

максимальное число соединений Postgres за процесс, включая постоянные соединения.

5.9. Директивы конфигурации Sybase

`sybase.allow_persistent` boolean

допускает постоянные соединения Sybase.

`sybase.max_persistent` integer

максимальное число постоянных соединений Sybase за процесс.

`sybase.max_links` integer

максимальное число соединений Sybase за процесс, включая постоянные соединения.

5.10. Директивы конфигурации унифицированных ODBC

`uodbc.default_db` string

источник данных ODBC используется, если ничего не определено в `odbc_connect()` или `odbc_pconnect()`.

uodbc.default_user string

имя пользователя, используемое, если нет определений в odbc_connect() или odbc_pconnect().

uodbc.default_pw string

пароль, используемый, если не определено в odbc_connect() или odbc_pconnect().

uodbc.allow_persistent boolean

допускает постоянные, устойчивые ODBC-соединения.

uodbc.max_persistent integer

максимальное число постоянных ODBC-соединений за процесс.

uodbc.max_links integer

максимальное число ODBC-соединений за процесс, включая постоянные соединения.

5.11. Директивы конфигурации модуля Apache

Выполнение PHP в виде модуля Apache — наиболее эффективный способ использования пакета. Если пакет выполняется в виде модуля, то это означает, что функциональные возможности PHP объединены с функциональными возможностями сервера Apache в одной программе.

Директивы, приведенные ниже, могут быть помещены или в файл srm.conf, или внутри тегов <Directory> . . . </Directory> в файле access . conf, или внутри тегов <Location /path> ... <Location> в файле access.conf, или в индивидуальных файлах. htaccess:

phpShowInfo on|off

включает или выключает нижние информационные колонтитулы PHP.

phpLogging on|off

включает или выключает регистрацию.

phpUploadTmpDir directory

устанавливает каталог, в который будут помещены переданные с помощью форм файлы.

phpDbmLogDir directory

устанавливает каталог, в котором будут размещаться файлы регистрации на базе dbm.

phpMsqlLogDB database

устанавливает имя базы данных mSQL, которая будет использоваться для регистрации.

```
phpAccessDir directory
```

устанавливает каталог, в котором хранятся внутренние файлы PHP для управления доступом.

```
phpMaxDataSpace KiloBytes
```

максимальный размер, до которого может расти внутренний подпул модуля PHP. Уменьшение этого значения минимизирует использование ресурсов mod_php на нашей системе, но это может также ограничить программистов при написании сложных скриптов.

Все эти директивы необязательные. Если директива не определена, будет использоваться значение по умолчанию, заданное во время компиляции.

Заключение

Файл php.ini читается при запуске синтаксического анализатора PHP. Для версий серверного модуля PHP это случается только один раз, когда запускается Web-сервер. Для версии CGI это случается при каждом вызове.

Посмотреть установочные параметры большинства значений конфигурации можно, вызвав функцию phpinfo().

Глава 6
Проблемы безопасности

PHP — мощный язык и интерпретатор. Независимо оттого, включен он в Web-сервер как модуль-или выполняется как разделение исполняемых файлов CGI, он может иметь доступ к файлам, выполнять команды и открывать сетевые соединения на сервере. Эти свойства дают возможность выполнять что-либо небезопасное на Web-сервере.

PHP разработан специально для того, чтобы быть более безопасным языком при написании программ CGI, чем Perl или C. При правильном выборе compile-time и runtime-опций конфигурации он дает как раз ту комбинацию свободы и безопасности, которая вам нужна.

Существует много разных путей использования PHP, есть также и большой выбор конфигураций, управляющих поведением PHP. Это гарантирует, что вы можете использовать PHP для многих целей, но означает также, что есть комбинации этих опций и конфигураций сервера, которые заканчиваются небезопасной установкой. Эта глава объясняет различные комбинации опций конфигурации и ситуации, в которых они могут быть удачно использованы.

В этой главе мы рассмотрим:

- использование PHP как бинарного CGI;
- установка модуля Apache;
- безопасность файловой системы;
- создание VirtualHost с разумными ограничениями безопасности PHP.

6.1. Использование PHP как бинарного CGI

Использование PHP как исполняемых файлов CGI является выбором установок, которые по некоторой причине не хотят внедрить PHP как модуль в программное обеспечение сервера (подобно Apache). Также PHP используют с другими типами оболочек CGI, чтобы создать надежное окружение chroot и `setuid` для сценариев. В таком случае обычно включается установка выполняемого PHP в каталог `cgi-bin` на Web-сервере. Правила хорошего тона рекомендуют, кроме того, устанавливать любые интерпретаторы в `cgi-bin`. Исполнимый PHP может быть использован в качестве автономного интерпретатора, но разработан для того, чтобы предохранить от атаки, которую эта установка делает возможной, — доступ к системным файлам: `http://my.host/cgi-bin/php?/.../passwd`.

Информация запроса в URL после знака вопроса (?) проходит как аргументы командной строки интерпретатору через интерфейс CGI. Обычно интерпретаторы открывают и выполняют файл, указанный как первый аргумент в командной строке.

ВНИМАНИЕ

Вызванный как исполняемый CGI-файл PHP отказывается интерпретировать командные аргументы строки.

Также установка выполняемого PHP может вызвать доступ к любым Web-документам на сервере: `http://my.host/cgi-bin/php/secret/doc.html`. Часть URL с информацией о пути, стоящая после имени PHP-файла (`/secret/doc.html`), обычно используется, чтобы определить имя файла, который должен открываться и интерпретироваться CGI-программой. Некоторые директивы конфигурации Web-сервера (Apache: Action) используются, чтобы перенаправить запросы к документам подобно `http://my.host/secret/script.php` на PHP-интерпретатор. С такой установкой Web-сервер сначала проверяет разрешение доступа в каталоге `/secret` и потом создает запрос перенаправления `http://my.host/cgi-bin/php/secret/script.php`. Если запрос не дается изначально в этой форме, Web-сервер не проверяет доступ к файлу `/secret/script.php`, но только для файла `/cgi-bin/php`. Таким образом, любой пользователь, имеющий доступ к `/cgi-bin/php`, получает доступ к любым защищенным документам на сервере.

СОВЕТ

В PHP опция compile-time конфигурации enable-force-cgi-redirect и директивы runtime-конфигурации doc_root и user_dir может использоваться для того,

чтобы отразить эту атаку, если дерево документов сервера имеет любые директории с ограничениями доступа. Ниже приведены полные описания других комбинаций.

Если сервер не имеет информации, которая ограничивается паролем или управлением доступа на основе IP, нет потребности в этих опциях конфигурации. Если Web-сервер не позволяет производить перенаправление или не имеет пути, чтобы связаться с исполнимым PHP, который запрашивает успешно перенаправленный запрос, можно указать опцию `disable-orce-cgi-redirect` для конфигурирования сценария.

Однако нужно убедиться, что ваши сценарии PHP не ссылаются ни на адрес `http://my.host/cgi-bin/php/dir/script.php`, ни на адрес `http://my.host/dir/script.php`.

Перенаправление может быть сконфигурировано, например в Apache, директивами AddHandler и Action (см. ниже).

Использование `-enable-force-cgi-redirect`

Эта compile-time опция предохраняет от вызова PHP напрямую с URL подобно `http://my.host/cgi-bin/php/secretdir/script.php`. Вместо того чтобы выполнить запрос, PHP производит только грамматический разбор в этом способе, если выполнено правило перенаправления Web-сервера. Обычно переадресация в конфигурации Apache сделана со следующими директивами:

```
Action php-script /cgi-bin/php
AddHandler php-script .php
```

Эта опция была протестирована только с Web-сервером Apache и пользуется поддержкой Apache, чтобы установить нестандартную внешнюю переменную CGI `REDIRECT_STATUS` для перенаправленных запросов. Если ваш Web-сервер не может сообщать, что запрос прямой или перенаправленный, вы не можете использовать эту опцию и должны использовать один из других путей запуска версии CGI.

Установка `doc_root` или `user_dir`

Размещение активного содержания, такого как скрипты и модули, в каталогах документов Web-сервера иногда является небезопасным. Если при некоторой ошибке конфигурации сценарии не выполняются, они отображаются как обычные HTML-документы, что может закончиться утечкой интеллектуальной собственности или информации безопасности, например, паролей. Поэтому многие системные администраторы предпочитают устанавливать другие каталоги для сценариев, которые будут доступны только через PHP CGI и, следовательно, всегда проинтерпретированы.

Если недоступен метод перенаправления неуверенных запросов, как описано выше, также необходимо установить корневой каталог сценариев (doc_root), который отличается от корневого каталога Web-документов.

Вы можете определить корневой каталог для скриптов директивой конфигурации doc_root в файле php.ini или установить переменную окружения PHP_DOCUMENT_ROOT. В таких случаях CGI-версия PHP всегда будет добавлять doc_root и путь к файлу в запросах, так что вы всегда будете уверены, что за пределами этого каталога скрипты выполняться не будут (кроме user_dir, см. ниже).

Другая используемая опция — user_dir. Когда она не установлена, открытием файла управляет только doc_root. Открытие URL, подобно http://my.host/~user/doc.php, не даст результата при открытии файла из каталога пользователя, но вызывает файл ~user/doc.php из каталога doc_root (имя каталога начинается с тильды [~]).

Если user_dir установлена, например, как public_php, запрос, подобный http://my.host/~user/doc.php, откроет файл doc.php в каталоге public_php домашнего каталога пользователя. Если это /home/user, то выполняется /home/user/public_php/doc.php.

Опция user_dir задается независимо от doc_root, так что вы можете контролировать доступ к document root и user directory отдельно.

Синтаксический анализатор PHP

Безопасная опция должна установить синтаксический анализатор PHP вне корневого каталога, например в /usr/local/bin. Отрицательная сторона этой опции заключается в том, что вы должны вставлять в первую строку любого документа, содержащего PHP-теги, строку подобно:

```
#!/usr/local/bin/php
```

Кроме того, вы должны сделать файлы выполнимыми. Точно так же, как вы поступаете с любым другим сценарием CGI, писанным на Perl или Shell или любом другом языке, который использует #! shell-escape механизм для самозапуска.

Чтобы PHP получил возможность корректно работать с PATH_INFO и PATH_TRANSLATED при такой установке, PHP-анализатор должен быть скомпилирован с опцией конфигурации — enable-discard-path.

6.2. Установка модуля Apache

Когда PHP используется в качестве модуля Apache, возникает необходимость в разрешении для доступа пользователей. Это приводит к некоторым противоречиям при защите и авторизации. Например, вы используете PHP, чтобы обратиться к базе данных. При условии, что база данных не имеет встроенного управления доступом, вы должны будете создать базу данных, доступную пользователю «nobody». В таком случае сценарий может получать доступ и изменять базу данных даже без имени пользователя и пароля. Вы можете защититься от этого при помощи Apache-авторизации или проектировать вашу собственную модель доступа, используя LDAP, .htaccess файлы и т. д.

6.3. Безопасность файловой системы

PHP подчинен безопасному построению многих серверных систем с доступом к файлам и директории каталогов. Это позволяет получить доступ к тем файлам, которые можно прочесть. Защита должна быть организована для любого из файлов, доступ к которому может принести очень много неприятностей.

С тех пор как был разработан PHP, начались проблемы следующего характера: скрипт, написанный на PHP, может позволить вам читать системные файлы типа `/. . ./password`, изменить связи вашей локальной сети на основе протокола CSMA-CD, может послать массивные задания на принтер и т. д. Это вызывает неприятности, связанные с системой обеспечения безопасности серверов.

Рассмотрим следующий пример, в котором при реализации скрипта происходит удаление файла из каталога:

```
<?php
// удаление файла из директории php
$username = $user_submitted_name;
$homedir = "/home/$username";
$file_to_delete = $userfile;
unlink ("$homedir/$userfile");
echo "$file_to_delete has been deleted!";
?>
```

Если же пользователь знает расположение каталогов на жестком диске, то возникает более серьезная угроза. В этом случае он без каких-либо проблем сможет удалить файл, содержащий пароли или любую другую важную информацию. Например:

```
<?php
$username = "../apache/";
$homedir = "/home/$username;
$file_to_delete = "passwd";
unlink ($homedir.$file_to_delete);
echo "$file_to_delete has been deleted!";
?>
```

В данном примере переменной `$username` задается имя каталога пользователя, в котором находится необходимый для удаления файл. Переменной `$homedir` задается путь направления поиска. Функция `unlink ($homedir.$file_to_delete);` производит по указанному пути удаление. `Echo ()` выводит результат.

Хотелось бы выделить два важных пункта, запомнив которые вы сможете предотвратить эти проблемы:

- необходимо в некоторой степени ограничивать возможности пользователей;
- проверяйте программы, которые будут запускаться на сервере.

Соблюдая эти пункты, вы избавитесь от проблем, связанных с употреблением файловых функций.

Существует еще множество примеров, при реализации которых можно получить доступ к тому или иному файлу. Все зависит от администратора и его способностей. Анализируйте каждый скрипт логическими путями, и тогда проблем с безопасностью у вас не будет.

6.4. Создание virtualHost с разумными ограничениями безопасности PHP

В данном параграфе приведены практические способы настройки virtualHost в сервере Apache. Создание наиболее безопасного сервера, настроенного на работу с PHP, является целью создателя сервера, поэтому мы решили включить этот практический материал в нашу книгу.

Рабочий пример с комментариями:

```
<VirtualHost 192.168.1.1:*>
# электронный адрес системного администратора
ServerAdmin test@your.domain.name.by
# имя пользователя и группы для выполнения скриптов под suexec
User project
Group project
# домен виртуального хоста и его алиасы
ServerName project.tyumen.ru
ServerAlias  www.project.tyumen.ru
# куда записывать логи
CustomLog /usr/local/apache/logs/project_access combined
ErrorLog /usr/local/apache/logs/project_error_log
# описание директории с HTML-файлами
<Directory "/home/project/htdocs">
# HTML, выдаваемые пользователю при ошибках доступа
ErrorDocument 404 /missing.html          # файл не найден
```

```
ErrorDocument 403 /access.html      # ошибка авторизации
# разрешаем SSI без возможности запуска локальных программ
IncludesNoExec),
# показ файлов в директории без index.html (Indexes),
# переход по символическим ссылкам (SymLinksIfOwnerMatch)
Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
# определяем индексные файлы:
DirectoryIndex index.html index.shtml index.htm index.php index.phtml
# разрешаем использование PHP 4 на сайте и определяем типы
фалов, интерпретируемых как PHP:
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps
AddType application/x-httpd-php .php3
AddType application/x-httpd-php .phtml
php_admin_flag engine on
php _admin_flag expose_ php off
# настройки безопасности и определение рабочих директорий:
php _admin_flag safe_mode on
php _admin_flag track_vars on
php _admin_value doc_root /home/project/htdocs
php _admin_value open_basedir /home/project/htdocs
php _admin_value safe_mode_exec_dir /home/project/bin
php _admin_value upload_tmp_dir /home/project/tmp
php _admin_value max_execution_time 10
php_admin_value upload_max_filesize 1024000
php_admin_flag magic_quotes_runtime on
order allow,deny
Allow from all
# определяем, какие настройки можно переопределять в .htaccess:
AllowOverride FileInfo AuthConfig Limit.
</Directory>
# корень дерева HTML-документов
```

```
DocumentRoot /home/project/htdocs
f разрешаем и описываем cgi-bin:
<Directory "/home/project/cgi-bin">
Options ExecCGI
</Directory>
ScriptAlias /cgi-bin/ "/home/project/cgi-bin/"
f определяем кодировку документов на сервере и кодировку, выдаваемую пользователям по умолчанию:
CharsetSourceEnc koi8-r
CharsetDefault koi8-r
CharsetSelectionOrder Portnumber
</VirtualHost>
```

Заключение

PHP является мощным инструментом. Как и в случае с другими мощными средствами, вы можете навредить сами себе. PHP функционирует таким образом, что если его небрежно использовать, то могут возникнуть проблемы с безопасностью вашей системы. Лучший путь для предотвращения такой ситуации — всегда знать, что вы делаете.

Часть II

Программирование на PHP

Глава 7

Основной синтаксис

У любого языка программирования, в том числе и у PHP, есть свой синтаксис. Синтаксис PHP очень прост и чем-то похож на синтаксис языка C или Perl. Он включает в себя операторы, разделенные между собой точкой с запятой.

ВНИМАНИЕ

Самая распространенная ошибка начинающих программистов — отсутствие точки с запятой между операторами.

PHP-код всегда заключается между

```
<?php
/*код программы
?>
```

Например:

```
<?php
echo "Hello world!";
?>
```

Все ошибки в PHP выдаются в окно браузера по умолчанию (чего нельзя сказать о CGI, где все ошибки записываются в лог-файл), и найти их при внимательном поиске и определенном опыте не составит труда. Тем более, что интерпретатор подскажет номер строки, в которой произошла ошибка.

Писать программу на PHP можно в любом текстовом редакторе, но желательно делать это в программе, которая бы обеспечивала выделение цветом синтаксиса

и нумерацию строк. Мы предлагаем вам воспользоваться FTP-клиентом CuteFTP, SNK Visual HTML, HotDog и др. Также вам понадобится комплект программ для работы с PHP. Как правило, используется Apache с модулем PHP, хотя это и не обязательно. Как было выше сказано, подходит любой сервер, например Xitami. Но первый вариант бесплатен и имеет большую поддержку документацией (в том числе на русском языке) и форумами, где можно узнать ответ на любой вопрос.

Файл, содержащий код PHP, может иметь любое расширение. Файлы с каким расширением будут обрабатываться с помощью PHP, указывается, например, в файле `httpd.conf` в строке Add Type. Мы рекомендуем вам в целях совместимости всегда использовать `phtml`.

В этой главе вы изучите:

- какие существуют способы ввода PHP-кода в документ;
- как разделяются инструкции;
- для чего применяют комментарии и как их используют.

7.1. Способы ввода PHP-кода в документ

На данный момент существует четыре способа ввода PHP-кода в HTML-документ:

```
1.<? echo("простейший способ, инструкция обработки SGML\n"); ?>
2. <?php echo("при работе с XML документами делайте так\n"); ?>
```

СОВЕТ

Этот способ является самым распространенным, хотя принципиально условия использования того или иного способа введения PHP-кода в документ никто не устанавливал. Нижеприведенные способы так же эффективны, как и этот. Выбор того, которым вы будете пользоваться, зависит только от вас.

```
3. <script language="php">
echo ("некоторые редакторы (подобные FrontPage) не любят обрабатывающие инструкции");
</script>;
4. <% echo("для совместности с визуальными средствами формирования страниц"); %>
```

Вот обычный пример использования первого способа:

```
<? include('info1.html') ?>
<h3>Здесь текст страницы</h3>
<? include('info2.html') ?>
```

Посетителю такой страницы будет показан документ с картинками меню и текстом. Отдельные файлы info1.*html* и info2.html будут просто включены в тот документ, который посылается сервером посетителю страницы. Таким образом можно отделить оформление страницы от ее наполнения, сильно облегчая работу по внесению информации на сайт даже неквалифицированным работникам.

Пример 7.1. Условные операторы

```
Ваше имя
<? if ($name = = 'Саша'):?>
Саша
<? else:?>
Петр I
<? endif ?>
```

В описанном выше примере мы получим текст «Ваше имя Саша» или «Ваше имя Петр I» в зависимости от значения переменной $name.

В языке PHP есть несколько функций вывода — echo, printf и др.

Пример 7.2. **Функция вывода**

```
If ($name = = "Саша") echo "Саша";
```

Перевод строки, идущей после закрытия тега ?>, интерпретатор пропускает. Сделано это для удобства форматирования исходных текстов скриптов.

7.2. Разделение инструкций

При написании программы на PHP вам необходимо пользоваться разделением инструкций. В этом нет ничего сложного — в PHP инструкции разделяются так же, как в C или Perl, — точкой с запятой.

Необходимо заметить, что закрывающий тег ?> также подразумевает завершение инструкции, другими словами, если после инструкции следует закрывающий тег ?>, то можно не использовать точку с запятой в конце инструкции. Например:

```
<?php
echo( "Welcome to PHP");
?>
```

эквивалентно выражению:

```
<?php echo("welcome to PHP") ?>
```

Обратите внимание, что в первом случае используется точка с запятой в конце инструкции, а во втором — нет.

7.3. Использование комментариев

Известно, как важны комментарии для документирования исходного текста и насколько удобнее и привлекательнее делают его понятные имена. Написанная вами программа, состоящая из большого количества строк, будет идеально работать и без каких-либо комментариев. Возможно, некоторое время вы будете подробно помнить каждую строчку программы и ее смысл. Но потом вы это забудете, так как сам код программы вам больше изменять не придется.

Но представьте ситуацию, при которой с развитием новых продуктов либо технологий доступа и обработки информации у вас возникнет необходимость изменить программу. Тут вам можно будет только посочувствовать. Без комментариев будет довольно сложно разобраться в программе, а тем более, если понадобится изменить программу не вам, а вашему знакомому или товарищу по работе. На то, чтобы понять, как же работает этот код, придется затратить немалое количество часов. Чтобы помочь вам избежать подобной ситуации, в этом параграфе излагаются некоторые несложные соглашения об именах и комментариях.

Комментарии во всех языках программирования призваны помочь исчерпывающе документировать исходный текст программ.

В PHP, впрочем, как и в C, допустимы две разновидности комментариев. Комментарии традиционного стиля начинаются знаком /* и заканчиваются знаком */. Например:

```
/* это комментарий в одну строку*/
/* этот комментарий занимает
несколько строк*/
```

Комментарии этого типа не могут быть вложенными, т. е. конструкция

```
V* один комментарий, /* а здесь начинается другой */*/
```

недопустима. Эта ситуация часто возникает, когда комментарий используется в процессе отладки.

Комментарии второго типа — односторонние — начинаются знаком // и заканчиваются в конце строки, в которой этот знак появляется, т. е. эти комментарии действуют только на протяжении одной строки. Если же вам надо написать несколько строк комментариев, просто начинайте каждую новую строку с //:

```
// этот комментарий занимает
// несколько строк
```

Односторонние комментарии не обязательно занимают всю строку и могут следовать за текстом кода, например:

```
<?php
while (!feof($fp) // цикл будет продолжаться,
                  // пока не найдена
{                 // закрывающая скобка
...               // при недостигнутом конце
}                 // строки
```

Заметьте также, что комментарий типа // можно включить в комментарии /*...*/. Если вы ограничитесь использованием в тексте только комментариев типа //, то при отладке вы сможете использовать /*...*/ для того, чтобы закомментировать большие фрагменты текста, не беспокоясь об ошибках компиляции из-за вложенных комментариев. Также если комментарий маленький и занимает только одну строчку, можно поставить // и таким образом легко закомментировать любую строчку до ее конца. Как обычно, пробелы, символы табуляции и перевод строки просто игнорируются и могут применяться для улучшения удобочитаемости кода PHP.

ВНИМАНИЕ

Классы тоже могут использовать односторонние комментарии — краткое описание функциональности, заключенной в классе.

Заключение

Данная глава позволила вам получить полное представления о способах ввода кода PHP в документ. Хотелось бы выделить из всего вышесказанного следующее: каждая команда PHP обычно начинается с тега <?php и заканчивается ?>. Если вы используете несколько команд подряд, они могут быть объединены внутри одной пары тегов. В этом случае каждую команду необходимо отделять друг от друга символом ;. В любом месте PHP-скрипта можно размещать комментарии. Для начала комментария используется символ /*, а для его завершения — */. Если комментарий небольшой, удобней использовать символы // — тогда все, что следует за ними до конца строки, будет игнорироваться, подобно лишним символам пробела, табуляции и новой строки. И самое главное, на чем еще хотелось бы остановить ваше внимание, — операторы в PHP разделяются точкой с запятой. Далее на примерах программ вы сможете более детально это освоить и научиться применять на практике.

Глава 8
Типы данных

PHP поддерживает переменные следующих типов:

- integer — целое,
- double — число с дробной частью,
- string — строковая переменная,
- array — массив,
- object — объектная переменная,
- boolean — булев.

В PHP тип переменной обычно устанавливается не программистом, а определяется самим PHP во время выполнения программы в зависимости от контекста, в котором данная переменная используется.

Если вам нужно указать тип переменной самим, то вы можете использовать для этих целей инструкцию cast либо функцию settype(). Но учтите, что переменная в определенных ситуациях может вести себя по-разному. Все зависит от того, какой тип определен для нее на данный момент. В данной главе мы подробно рассмотрим следующие вопросы:

- имена переменных;
- строки;
- преобразование строк;
- массивы;
- указатель array pointer;
- изменение типа;
- определение типов переменных;
- приведение типа.

8.1. Имена переменных

В PHP все переменные имеют имя, которые начинаются со знака $, например $variable. Если взять для сравнения C/C++ и подобные языки, то людям, привыкшим к этим языкам, такой принцип именования переменных скорее всего покажется странным. Но на практике такой подход имеет все же определенные преимущества. Рассмотрим их подробнее.

1. При таком способе формирования имен переменных их очень легко отличить от остального кода. Если в других языках иногда может возникать путаница с тем, что при первом взгляде на код не всегда ясно, — где здесь переменные, а где функции, то в PHP этот вопрос даже не встает.

2. Данный принцип позволяет очень просто и элегантно реализовать функциональность, просто недоступную иначе. Например, ссылка на переменную по ее имени, хранящемуся в другой переменной:

```
$name = 'value';
$value = 5;
echo $$name;
```

В данном примере PHP выведет результат 5. Такие особенности стали возможны из-за заложенного в PHP принципа: *«anything is an expression»* (любая вещь является выражением). Исходя из этого, PHP воспринимает выражение `$$name` следующим образом:

- интерпретатор встречает символ $, а это значит, что следующее выражение является именем переменной;
- интерпретатор встречает еще один символ $ и продолжает поиски имени уже другой переменной, значение которой необходимо для вычисления значения предыдущего выражения;
- интерпретатор получает имя name, находит переменную с таким именем, берет ее значение (`'value'`) и возвращается назад;
- поскольку на предыдущем шаге мы искали имя переменной, то значение только что вычисленного выражения воспринимается именно как имя переменной. Интерпретатор ищет переменную с таким именем (value) и возвращает ее результат (5).

Как вы сами видите, это очень удобная и гибкая система. Достаточно небольшой тренировки, чтобы вы смогли творить чудеса с ее помощью!

Но самое интересное здесь то, что точно так же вы можете оперировать с любыми элементами языка, имеющими имя, т. е., например, вызывать метод класса по его имени, содержащемуся в переменной (см. гл. 15).

8.2. Строки

Однозначная идентификация переменной позволила создателям PHP дать возможность программистам использовать переменные непосредственно внутри строк.

Например:

```
$name = 'John';
$age = 23;
echo "$name is $age years old";
```

После выполнения данного примера мы получим строку: «John is 23 years old». Думаем, вы согласитесь, что это исключительно удобно и может значительно облегчить работу. Обратите внимание на очень немаловажную деталь — в PHP, в отличие от многих других языков программирования, различаются строки, за-

ключенные в одинарные и двойные кавычки. Подобное замещение имен в переменных их значениями производится только в строках, заключенных в двойные кавычки! Это очень важно запомнить, дабы избежать ошибок в будущем.

ВНИМАНИЕ

Строки, заключенные в двойные кавычки, могут содержать и более сложные выражения. Для этого необходимо заключить их в фигурные скобки:

```
$man = array('John',23);
echo "{$man[0]} is {$man[1]} years old";
```

Результат будет таким же, как и в предыдущем примере.

У PHP есть еще одна интересная особенность работы со строками — указатель here docs.

Пример 8.1. Маркер TEXT

```
$str = <<<TEXT
There can be any text
which will be stored
into a string
right in the same way
as it
typed here
TEXT;
```

В данном примере переменной будет весь текст, содержащийся между начальным и конечным маркерами (здесь — строка TEXT, но этот маркер может быть любой последовательностью символов). На то что это here doc, указывает наличие <<< непосредственно после знака присваивания.

В примере продемонстрировано удобство того, что если строке необходимо присвоить какой-то сложный текст, особенно, если необходимо сохранить его форматирование, то here doc окажется незаменимым помощником.

СОВЕТ

В случае использования here docs есть возможность использования переменных внутри строки, как это было описано ранее.

Строки могут задаваться с использованием одного или двух наборов разделителей.

Если строка заключена в двойные кавычки ("), то переменные в строке будут раскрываться (в соответствии с некоторыми ограничениями синтаксического разбо-

ра). Как в C и Perl, при указании специальных символов может использоваться обратный слеш (\) (табл. 8.1).

Таблица 8.1. Символы, кодируемые с использованием обратного слеша

Последовательность	Значение
\n	Перевод строки (LF или 0x0A в кодировке ASCII)
\r	Возврат каретки (CR или 0x0D в кодировке ASCII)
\t	Горизонтальная табуляция (HT или 0x09 в кодировке ASCII)
\\	Обратный слеш
\$	Знак доллара
\"	Двойная кавычка
\[0-7]{1,3}	Последовательность символов, соответствующая регулярному выражению, является символом в восьмеричной нотации
\x[0-9A-Fa-f]{1,2}	Последовательность символов, соответствующая регулярному выражению, является символом в шестнадцатеричной нотации

Если вы попытаетесь представить символ таким образом, будут выведены и обратный слеш, и сам символ. Если в PHP 3 при этом будет выдано предупреждение на уровне E_NOTICE, то в PHP 4 предупреждающее сообщение не генерируется.

Второй способ разделения строк, который мы рассмотрим, — использование одинарных кавычек (' '). Если строка заключается в одинарные кавычки, то в ней можно использовать только символы "\\" и "\'". Таким образом, вы сможете указывать одинарные кавычки и обратные слеши в строке, отделенной одинарными кавычками. Переменные в строке, отделенной одинарными кавычками, обрабатываться не будут.

Еще один способ разделения строк — с помощью синтаксиса документов ("<<<"). После <<< укажите идентификатор, затем строку, а затем тотже самый идентификатор, чтобы закрыть "кавычки".

Закрывающий идентификатор должен начинаться непосредственно в первом столбце строки. Кроме того, идентификатор должен удовлетворять правилам именования, как и любая метка в PHP: он должен содержать только алфавитно-цифровые символы и подчеркивания и не должен начинаться с цифры или подчеркивания.

Такой текст ведет себя, как строка в двойных кавычках, но без двойных кавычек. Это означает, что вам не придется кодировать кавычки с помощью слеша, но использовать коды со слешем все равно можно. Переменные раскрываются, но следует проявлять осторожность при выражении сложных переменных в таких строках.

Пример 8.2. Кавычки в строке типа документа

```
<?php
$str = <<<EOD
```

```
/* пример строки,
занимающей несколько строк
и использующей синтаксис документа */
EOD;
/* более сложный пример с переменными */
class foo {
var $foo;
var $bar;
function foo() {
$this->foo = 'Foo';
$this->bar = array('Bar1', 'Bar2', 'Bar3');
}
}
$foo = new foo();
$name = 'MyName';
echo <<<EOT
Меня зовут "$name". Я печатаю $foo->foo.
Теперь я печатаю {$foo->bar[1]}.
Здесь должна напечататься заглавная буква 'A': \x41
EOT;
?>
```

Результат выполнения программы представлен на рис. 8.1.

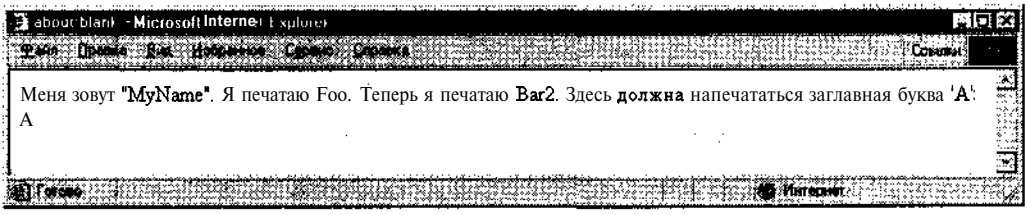


Рис. 8.1. Результат выполнения скрипта

ВНИМАНИЕ

Поддержка такой записи строк добавлена в PHP 4.

Объединять строки можно с помощью оператора точки «.». Но обратите внимание, что оператор «+» (сложение) не будет работать при объединении строк.

Доступ к символам в строках можно получить, обрабатывая строки как индексированный массив символов, используя синтаксис языка C (пример 8.3).

Пример 8.3. Работа со строками

```
<?php
/* назначение строки */
$str = "Это строка";
/* добавление к ней другой строки */
$str = $str . " с дополнительным текстом";
/* еще один способ добавления включает перевод строки */
$str .= " и перевод строки в конце.\n";
/* эта строка закончится текстом '<p>Число: 9</p>' */
$num = 9;
$str = "<p>Число: $num</p>";
/* эта будет '<p>Число: $num</p>' */
$num = 9;
$str = '<p>Число: $num</p>';
/* получение первого символа строки */
$str = 'Это тест.';
$first = $str[0];
/* получение последнего символа строки */
$str = 'Это все еще тест.';
$last = $str[strlen($str)-1];
?>
```

8.3. Преобразование строк

Если строка оценивается как числовое значение, то результирующие значение и тип будут определяться следующим образом.

Строка оценивается как число двойной точности, если она содержит любой из символов «.», «e» или «E». В противном случае она будет оценена как целое число.

Далее значение определяется начальной частью строки. Если строка начинается с допустимых числовых данных, они будут использоваться в качестве значения.

В противном случае значение будет равно нулю. Допустимые числовые данные — это необязательный знак, за которым следует одна или несколько строк цифр (также может присутствовать десятичная точка), а за ними необязательная экспонента. Экспонента — это «e» или «E», за которой следуют одна или несколько цифр.

Если первое выражение является строкой, тип переменной зависит от второго выражения. Например:

```
$foo = 1 + "10.5"; // $foo имеет двойную точность (11.5)
$foo = 1 + "-1.3e3"; // $foo имеет двойную точность (-1299)
$foo = 1 + "bob-1.3e3"; // $foo - целое (1)
$foo = 1 + "bob3"; // $foo - целое (1)
$foo = 1 + "10 Small Pigs"; // $foo - целое (11)
$foo = 1 + "10 Little Piggies"; // $foo - целое (11)
$foo = "10.0 pigs " + 1; // $foo - целое (11)
$foo = "10.0 pigs " + 1.0; // $foo имеет двойную точность (11)
```

Данные примеры можно протестировать, дополнив их следующей строкой:

```
echo "\$foo==$foo; тип " . gettype ($foo) . "<br>\n";
```

8.4. Массивы

Массив — это ряд переменных, упорядоченных по имени и имеющих различный индекс. Для примера представьте, что у вас есть 20 названий, которые нужно внести в программу. Можно для удобства назначить переменным одинаковые имена и ставить в конце каждого имени число в соответствии с номером названия. Получится простейший одномерный массив. PHP предоставляет большой выбор средств для детальной и удобной работы с таким набором — массивом. Число (так называемый индекс) нужно заключать в квадратные скобки. Допустим, есть ряд строковых переменных — компьютер, Интернет, модем, монитор. Выберем для массива имя $m, хотя доступно любое, как и для обычной переменной. Индекс в массивах начинается не с единицы, а с нуля, и, таким образом, для внесения наших слов в массив надо сделать так:

```
$m[0] = "компьютер";
$m[1] = "Интернет";
$m[2] = "модем";
$т[3] = "монитор";
```

Теперь у нас создан массив с именем $т и максимальным индексом (количество элементов в массиве) — 4. Именно 4, хотя последний заполненный элемент — 3. Если мы попробуем считать элемент с четвертым индексом, результат будет равен пустой строке, так как там просто ничего нет. Обращаться к элементам массива

нужно по имени массива и его индексу, что и составляет основное удобство. Например, мы можем вывести на экран все элементы массива:

```
$i = 0; while ($i < count($m)) { echo $m[$i]."<br>"; $i++; }
```

Функция count ($m) выдает нам количество элементов массива. Таким образом, у нас всегда есть возможность знать, сколько элементов в данном массиве.

Массивы в PHP — это очень мощный и гибкий механизм. Он позволит вам сделать практически все, что вы только можете пожелать сделать с массивами. Поддерживаются вложенные массивы, их вложенность никак явно не ограничена. В PHP существует большое количество функций для работы с массивами, они помогут вам выполнить необходимые операции без лишних затрат времени и сил.

Одномерные массивы

Одномерный массив состоит из одной строки.

PHP поддерживает как скалярные (0, 1, 2...), так и ассоциативные массивы ("a", "b", "c" ...). Между ними, практически, нет никакой разницы. Создать массив можно с помощью функций list() или array () или просто определить значения элементов массива. Например:

```
$a[0] = "abc";
$a[1] = "def";
$b["foo"] = 13;
```

Есть еще один способ создания массивов: путем простого добавления в него элементов. При присвоении значения переменной массива с пустыми скобками это значение просто добавляется в конец массива. Например:

```
$a[] = "hello"; // $a[2] == "hello"
$a[] = "world"; // $a[3] == "world"
```

Сортировать массивы можно с помощью функций asort(), arsort(), ksort(), rsort(), sort(), uasort(), usort() и uksort() в зависимости от необходимого типа сортировки (см. п. 20.6).

Определить число элементов в массиве можно с помощью функции count().

Перемещаться по массиву можно с помощью функций next() (следующий элемент) и prev() (предыдущий элемент). Еще один способ перемещения по массиву — использование функции each() (вывод необходимого элемента).

Многомерные массивы

Несмотря на пугающее название, многомерные массивы на самом деле очень просты. Для каждого измерения массива в конце добавляется дополнительный индекс. Например:

```
$a[1] = $f; // примеры одномерных массивов
```

```
$a["foo"] = $f;
$a[1][0] = $f; // двумерный массив
$a["foo"][2] = $f; // числовые и ассоциативные индексы
$a[3]["bar"] = $f; //. можно смешивать
$a["foo"][4]["bar"][0] = $f; // четырехмерный массив
```

В PHP 3 нельзя ссылаться на многомерные массивы в строках. Например, следующая строка не даст желаемого результата:

```
$a[3]['bar'] = 'Bob';
echo "Так работать не будет: $a[3][bar]";
```

В PHP 3 будет выведена строка «Так работать не будет: Array [bar]». Однако для получения нужного результата можно использовать оператор конкатенации строк («.»):

```
$a[3]['bar'] = 'Bob';
echo "Так работать будет: " . $a[3][bar];
```

В результате будет выведено: «Так работать будет».

В PHP 4 эту проблему можно решить, заключив ссылку на массив (в строке) в фигурные скобки:

```
$a[3]['bar'] = 'Bob';
echo "Так работать будет: {$a[3][bar]}";
```

Заполнять многомерные массивы можно по-разному. Самым тяжелым для понимания является использование команды array() для ассоциативных массивов. Следующие два примера кода одинаково заполняют одномерный массив. Первый пример:

```
$a["color"] = "red";
$a["taste"] = "sweet";
$a["shape"] = "round";
$a["name"] = "apple";
$a[3] = 4;
```

Второй пример:

```
$a = array(
"color" => "red",
"taste" => "sweet",
"shape" => "round",
"name" => "apple",
```

```
3 => 4
);
```

Функция array() для многомерных массивов может быть вложенной (пример 8.4).

Пример 8.4. Многомерные массивы

```
<?php
$a = array( .
"apple" => array(
"color" => "red",
"taste" => "sweet",
"shape" =>-"round"
),
"orange" => array(
"color" => "orange",
"taste" => "tart",
"shape" => "round"
),
"banana" => array(
"color" => "yellow",
"taste" => "pasty",
"shape" => "banana-shaped"
)
);
echo $a["apple"]["taste"];
?>
```

Результат выполнения программы представлен на рис. 8.2.

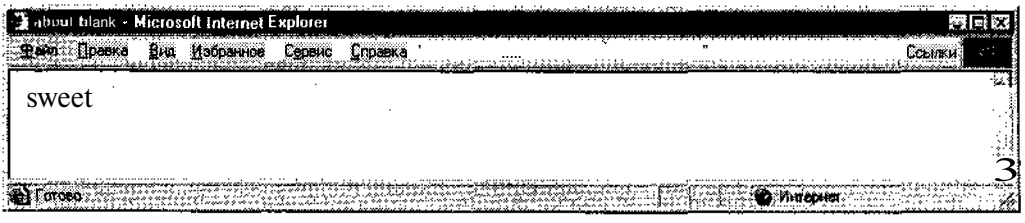


Рис. 8.2. Результат выполнения скрипта

Array — тип данных, такой же, как integer или string. Если переменная `$array` — массив, то gettype (`$array`) выдаст array (имя переменной выбрано случайным образом и к типу данных оно отношения не имеет), а `is_array`($array) выдаст true.

ВНИМАНИЕ

Индексы в массиве — это указатели на его элементы.

PHP поддерживает как скалярные, так и ассоциативные индексы массивов. Ассоциативными массивами пользуются, например, в функциях работы с MySQL:

```
$row = mysql_fetch_array($result);
print ($row["field1"]. $row["field2"]. $row["field3"]).
```

Теперь поговорим о создании массивов поподробнее.

Первое средство — функция array:

```
$array = array($val1, $val2, $val3);
$array2 = array($key1 => $val1, $key2 => $val2, $key3 => $val3);
```

В первом случае получится скалярный массив, во втором — ассоциативный.

Можно создать массив, просто указав индекс и значение.

```
$array[$key] = $val;
```

Переменная $array, если она не существовала до этого, станет массивом, а в ячейку с индексом $key (не обязательно 0) помещается значение $val. Несомненным достоинством PHP является отсутствие необходимости указывать индекс ячейки.

```
$new_array[] = "0"; // то же самое, что $new_array[0] = "0";
$new_array[] = "1"; // то же самое, что $new_array[1] = "1";
```

Вывод переменной в тексте часто делают таким образом:

```
print ("test $variable text");
```

Но если вы попытаетесь вывести элемент массива таким же образом, то у вас ничего не получится. В официальной документации по PHP (на сайте www.php.net) приведены наглядные примеры многомерных массивов.

Функции, которые наиболее часто используются для работы с массивами — это sizeof (размер массива) и list & each (выбирают из каждого массива имя элемента и его значение).

Чтобы обработать все элементы скалярного массива, в котором индексы элементов представляют собой числовой ряд (от 0 до `$n`), достаточно использовать функцию sizeof и цикл:

```
for ($i=0;$i<sizeof($array);$i++) {
...
};
```

Если элементы начинаются не с 0, а с известного $m, то можно действовать так:

```
for ($i=$m;$i<sizeof($array)+$m;$i++) {
...
};
```

Но если массив ассоциативный (возможно, смешанный) или мы вообще не знаем, есть в ряду пустые места или нет, то нужно воспользоваться функциями list И each:

```
while (list($key, $val) = each($array)) {
```

Если надо вывести значения элементов в документ, то делается так:

```
print ("...$val...");
```

Чтобы обработать массив, обращаемся к его элементам:

```
$array[$key] = somefunction($val);
```

Мы специально отметили возможность разрывов в скалярных индексах. Дело в том, что те, кто работал с массивами, например, в Паскале, знают, что там массивы объявляются по принципу «от сих до сих». Но в отличие от того же Паскаля, в PHP массивы ничем не ограничены, и есть возможность создавать массивы хоть с сотого элемента, предыдущих девяносто девяти просто не будет существовать (соответственно, sizeof такого массива будет 1). В любой момент можно создать новый элемент или удалить существующий (если его не будет, программа «ругаться» не станет). Поэтому, если при отладке программы вы не уверены, что содержится в массиве, то сделайте следующее:

```
while (list($k, $v)=each($array))
print ("$k - $v<br>");
```

Другие функции, работающие с массивами: array_diff (сравнение массивов), array_rand (получение случайных элементов массива), array_unique (удаление повторов элементов) и in_array (проверка, есть ли заданное значение в массиве).

Нужно отметить еще одну особенность PHP при работе с массивами: в отличие от других языков, PHP позволяет задавать массивы практически любой сложности непосредственно в теле самой программы. Может быть, с первого взгляда эта возможность не покажется вам такой уж важной, однако это в корне неверное мнение. На практике вам не раз придется столкнуться с необходимостью описания какой-нибудь сложной структуры и наполнением этой структуры данными.

В других языках для этого, как правило, приходится писать дополнительный код, что не всегда удобно. В PHP же вы можете сделать это очень просто и элегантно:

```
$data = array(1,10,100,1000, // числовые данные
'Some text','Another text', // строковые данные
'name'=>'John','age'=>23, // ассоциативные связи в массиве
'date'=>array('day'=>10,'month'=>'may','year'=>2001)); //вложенный массив
```

Как видно, различные данные могут быть совмещены вместе в единой структуре без каких-либо проблем. Посмотрим, как можно обратиться к этим данным:

```
echo $data[1]; // результат — 10
echo $data[5]; // результат — 'Another text'
echo $data['age']; // результат - 23
echo $data['date']['month']; // результат - 'may'
```

8.5. Указатель array pointer

Как вы, наверняка, заметили, функции each и list работают без указания индекса. Естественно, что программа «знает», с каким элементом она работает. Указатель того, где в массиве программа находится в данный момент, называется array pointer. Он не хранится ни в какой переменной, его никак нельзя получить (можно только получить индекс элемента, на котором стоит указатель. Но при необходимости его можно двигать в начало, в конец массива, на следующий, предыдущий элемент. Впрочем, эти функции используются довольно редко. Но на всякий случай после обращений к элементам массива и перед работой с функциями, использующими aray pointer, выполняйте команду reset ($array), чтобы указатель встал в начало массива.

8.6. Изменение типа

PHP не требует явного определения типа при объявлении переменной, тип переменной определяется по контексту, в котором она используется. Например, если присвоите строковое значение переменной $var, $var станет строкой. А если затем присвоить переменной $var целое значение, то она станет целым.

Примером автоматического преобразования типа в PHP может служить оператор сложения «+». Если какой-либо из операндов является числом с дробной частью (тип double), то затем все операнды оцениваются, как double, и результат будет иметь тип double. Если эти операнды будут интерпретированы как целые (integers), то и результат будет также иметь тип integer. Отметим, что при этом не меняются типы самих операндов, меняется только оценка этих операндов. Например:

```
$foo = "0"; // $foo является строкой (в таблице символов ASCII— 48)
```

```
$foo++; // $foo является строкой "1" (в таблице символов ASCII - 49)
$foo += 1; // $foo сейчас является целым (2)
$foo = $foo + 1.3; // $foo сейчас имеет тип double (3.3)
$foo = 5 + "10 Little Piggies"; // $foo является целым (15)
$foo = 5 + "10 Small Pigs"; // $foo является целым (15)
```

СОВЕТ

Если последние два примера, приведенные выше, кажутся вам не совсем ясными, перечитайте еще раз раздел «Преобразование строк» (см. п. 8.3).

8.7. Определение типов переменных

Поскольку PHP определяет типы переменных и преобразует их (в общем) по мере необходимости, то не всегда очевидно, какой тип данная переменная имеет в какой-то определенный момент. Поэтому PHP включает в себя несколько функций, которые позволят вам определить текущий тип переменной. Это функции: `gettype()`, `is_long()`, `is_double()`, `is_string()`, `is_array()`, `is_object()`, `is_bool`, `is_interger`, `is_null`, `is_numeril`.

8.8. Приведение типа

Приведение типа работает в PHP во многом так же, как в C: название требуемого типа записывается в круглых скобках перед переменной, которая должна быть приведена к данному типу.

```
$foo = 10; // $foo является integer
$bar = (double) $foo; // $bar является double
```

Допускается следующее приведение типов:

- `(int)`, `(integer)` — приведение к целому,
- `(real)`, `(double)`, `(float)` — приведение к дробному типу,
- `(string)` — приведение к строке,
- `(array)` — приведение к массиву,
- `(object)` — приведение к объектной переменной.

Заметим, что табуляция и пробелы допускаются внутри круглых скобок, поэтому следующее функционально эквивалентно:

```
$foo = (int) $bar;
$foo = ( int ) $bar;
```

Заключение

Данная глава позволила вам разобраться в основных аспектах типов переменных, понять принципы построения строк и массивов. Основное, что вам необходимо запомнить из этого, заключается в следующем.

В PHP тип переменной обычно устанавливается не программистом, а определяется PHP во время выполнения программы в зависимости от контекста, в котором данная переменная используется.

В PHP все переменные имеют имя. Имена переменных в PHP начинаются со знака $, например $variable.

Возможность однозначно идентифицировать переменную позволила создателям PHP дать возможность программистам использовать переменные непосредственно внутри строк.

Объединять строки можно с помощью оператора точки «.». Оператор «+» (сложение) не будет работать при объединении строк.

PHP поддерживает как скалярные, так и ассоциативные массивы. В отличие от других языков PHP позволяет задавать массивы практически любой сложности непосредственно в теле самой программы.

Указатель того, где в массиве программа находится в данный момент, называется array pointer.

Приведение типа работает в PHP во многом так же, как в C: название требуемого типа записывается в круглых скобках перед переменной, которая должна быть приведена к данному типу.

Глава 9
Переменные

Как было сказано ранее переменная характеризуется именем, типом и значением. Имя переменной может быть любым и включать в себя цифры, буквы английского алфавита, а также разрешенные символы (например, символ подчеркивания или тире), об этом и другом мы поговорим более подробно в этой главе. Что касается типов, можно сказать, что значение переменной в соответствии с типом может быть практически любым, например переменная Ь = 5. Это говорит нам о том, что имя переменной — b, тип — целочисленный, значение — 5. Для более детального разъяснения приведем пример:

```
<?php
$number = 8;
$number2 = 19.82;
```

```
$path = "…/index.phtml";
$str • ="string";
?>
```

Напомним, все переменные в PHP должны начинаться с символа $, что позволяет интерпретатору безошибочно отличать их от команд PH P. В первой строчке нашего скрипта переменной $number присваивается значение 8, что автоматически делает эту переменную целочисленной. Специально описывать тип переменной, как в языках программирования Pascal и Visual Basic, не требуется. Хотя разделение на типы чисто условное, PHP автоматически стремится использовать правильный тип соответственно значению. Вторая строчка кода присваивает переменной $number2 значение 19.82, которое является значением с плавающей запятой. Третья и четвертая строчки кода присваивают своим переменным значения, являющиеся символьными строками. Все, что заключено в кавычки (включая цифры), будет интерпретировано как символьная строка. Если переменные используются, но не определены ранее, их значение принимается равным нулю либо пустой строке в зависимости от типа. В данной главе описаны:

- основные понятия при использовании переменных;
- операция получения адреса;
- область видимости (scope);
- переменные переменных;
- передача параметров скрипту при запуске из командной строки;
- формы HTML (Get/Post);
- передача значений переменных, соответствующих кнопкам формы;
- HTTP-Cookies;
- системные переменные;
- точки в именах входящих переменных;
- как проверить, был ли отмечен checkbox в форме;
- определение типов переменных.

9.1. Основные понятия при использовании переменных

Переменные в PHP начинаются со знака доллара $. Как было сказано ранее, это позволяет интерпретатору PHP понять, что та или иная совокупность символов относится к области переменных и все операции выполняются с ними как операции с переменными.

Необходимо заметить, что имя переменной чувствительно к регистру. Например:

```
$first = 'Sasha';
$First = 'Tatiana';
```

```
echo "$first, $First"; // данная функция выведет на экран
"Sasha, Tatiana"
```

Этот пример показывает, что двум похожим переменным можно задать разные значения. Постарайтесь запомнить это, так как именно на этом этапе делается множество ошибок.

Очень часто программисты, не задумываясь, вводят переменные с ничего не выражающими именами. Возьмите, к примеру, переменную

```
$z = 5;
```

В некоторых случаях — это осмысленное имя. Когда вы проделываете элементарные математические операции или же решаете уравнение, $z — замечательное имя для переменной, ясное, емкое и соответствующее проблемной области. Если вы строите двухмерный график в плоскости *x-z,* и здесь такое имя подходит. Однако оно далеко не лучшее, когда им начинают обозначать количество посещений сайта. Хорошим тоном среди программистов считается использование комментариев при определении переменной. Например:

```
$posetili = 5; // количество посещений
```

Заметьте, как удачно выбраное имя сразу делает ваш код не требующим комментариев — глядя на эту строку, вы уже можете понять, что происходит. Это требует немного больше времени на обдумывание подходящих имен, но если вы рассчитываете хоть на какую-то долговечность вашего кода (означающую, что он еще может понадобиться вам или кому-либо еще), то это стоит делать.

Приведем основные правила, необходимые при создании имен переменных:

- имя переменной должно начинаться с буквы или со строки подчеркивания. Например:

```
$name = "Sasha";
$_file = "script";
```

- в начале имени переменной не могут употребляться цифры. Например:

```
$4name = "Vitalik";
```

Такое имя переменной вызовет ошибку при выполнении программы.

ВНИМАНИЕ

В данном случае к символам относятся буквы: a-z, A-Z, а также символы таблицы ASCII, начинающиеся с 127 до 255 (0x7f-0xff).

Приведем пример, который пояснит все вышеизложенное:

```
$name = 'Igor';
$Name = 'Slava';
$string = 35;
```

```
$_4class = 5;
$4cout = 76;
$täty = 'minal';
```

Переменные $name и $Name отличаются только регистром, при выводе этих переменных мы получим следующее: «Igor, Slava» (для этого воспользуемся функцией echo"$name, $Name";). Переменная $string = 35; начинается с символа, эта переменная будет использована без ошибки, то же самое касается переменных $_4class и $täty. Символ «а» является вставляемым символом из таблицы ASCII под номером 228. Символы таблицы ASCII могут использоваться от 127 до 255, а последний попадает именно в этот интервал. Переменная $4cout имеет имя, которое начинается с цифры, что при ее последующем использовании приведет к ошибке. В этом случае в окно браузера будет выведена ошибка следующего содержания:

```
Parse error: parse error, expecting 'T_VARIABLE' or "$" in
z:\home\localhost\www\test.php on line 8
```

9.2. Операция получения адреса

Как известно, адрес есть у каждого объекта. Это, в частности, подразумевает возможность создания псевдонимов (aliases) для существующих объектов. Операция получения адреса незаменима во многих аспектах PHP. Для объяснения реализации этой операции рассмотрим программу:

```
<?php
$name = 'Sasha';
$secondname = &$name;
$secondname = "My name is $secondname";
$secondname = "I hope $secondname ";
echo $name;
echo $secondname;
?>
```

Опишем работу данного скрипта. Переменной $name задается значение Sasha. Далее для переменной $secondname определяется не значение, а адрес, о котором говорилось выше, т. е. переменной $secondname = &$name присваивается адрес переменной $name, при этом не важно, какое значение она имеет. Таким способом можно оперировать переменными. Рассмотрим строку $secondname = "My name is $secondname";. До этой строки наша переменная $secondname, согласно адресу переменной, имела значение Sasha. После того как произошло выполнение указанной строчки, переменная получит другое значение. Но при это мы оперируем не самими значениями, складывая их, а именно самим адресом, который принадле-

жит этой переменной. Это все можно описать следующим способом. Представим, что существует два уровня. Уровень, на котором находятся значения переменных, назовем верхним уровнем. Уровень, на котором расположен адрес, — нижним. Таким образом, все, что находится на верхнем уровне, просто прибавляется к имеющемуся, а в нашем примере со строкой `$secondname = "My name is $secondname";` выражение "My name is" прибавляется к значению, которое находится по указанному выше адресу. Следовательно, теперь на адресе, на который указывает переменная `$secondname`, имеется не одно слово, как было раньше, а фраза следующего содержания «My name is Sasha».

ВНИМАНИЕ

При написании строки $secondname = "My name is $secondname"; следите за двойными кавычками. Иначе в случае $secondname = 'My name is $secondname'; переменная получит значение "My name is $secondname".

Далее выполняется строка `$secondname = "I hope $secondname".` На основании имеющихся данных по этому адресу переменная приобретает значения: "I hope My name is Sasha", т. е. по этому же адресу опять происходит изменение. Притом значение ни в коем случае не изменяется, а именно прибавляется к существующему. Строка `echo $name;` выведет не первоначальное значение, которому принадлежит эта переменная, а именно то значение, по адресу которой оно имеется, т. е. будет выведена фраза «I hope My name is Sasha». То же самое касается функции `echo $secondname;`. Получается, что эти переменные имеют одинаковые значения.

Проделав элементарную операцию:

```
$firstname = 'Ira';
$name = 'Sasha';
$secondname = &name;
$secondname = &firstname;
```

можно увидеть, что как только меняется адрес, то и значение сразу меняется, т. е. в данном случае не происходит добавление, как в вышеописанном примере.

Обратите внимание на пример неправильного использования операции получения адреса:

```
<?php
$cout = 68;
$cout1 = &$cout;
$cout1 = &(45*2);
?>
```

Этот пример приведен для того, чтобы выявить, где может возникнуть ошибка и почему. Конечно, у вас будут возникать и другие ошибки, но на основании полу-

ченных знаний вы сможете дать логический анализ реализации программы. Перейдем сразу к строке, в которой неверно описывается работа с адресом. Наверное, вы уже догадались, что это строка `$coutl=& (45*2);`. Именно в ней возникнет ошибка при реализации программы. Дело в том, что в данном месте дается ссылка на выражение, Которое не имеет собственного имени. Когда происходит обращение к конкретному адресу переменной, необходимо, чтобы этот адрес имел имя, через которое к нему можно обратиться. Не обязательно, чтобы имя было постоянным, т. е. адрес может принадлежать сразу одной переменной, а потом, по мере усложнения программы, и многим. При этом само значение, находящееся на высоком уровне данной переменной, может изменяться.

9.3. Область видимости (scope)

В PHP при использования переменных существуют отличия от C/C++. Дело в том, что переменная, определенная глобально, т. е. не в самой функции, а до ее реализации, не может быть использована в функции при условии, что она не определена как global (глобальная) переменная. Например:

```
<?php
$a = 1; /* глобальное определение переменной */
$b = 4;
Function sum () {
echo $b;
echo $a; /* ссылка на локальную переменную */
}
sum();
?>
```

После попытки выполнить этот пример, получим следующие ошибки:

1. `Warning: Undefined variable: b in z:\home\localhost\www\test.php on line 11`
2. `Warning: Undefined variable: a in z:\home\localhost\www\test.php on line 12`

Тут сказано о том, что переменные, которые используются в функции sum(), явно не определены. Здесь видны отличия PHP от C/C++. Чтобы в будущем не совершать такого рода ошибок, необходимо определять переменные в функциях перед их применением. Например:

```
<?php
$a=3;
$b=4;
Function sum()
```

```
{
global $a,$b,$d;
$d=$a+$b;
}
sum();
echo $d;
echo $a;
echo $b;
?>
```

Результат выполнения программы представлен на рис. 9.1.

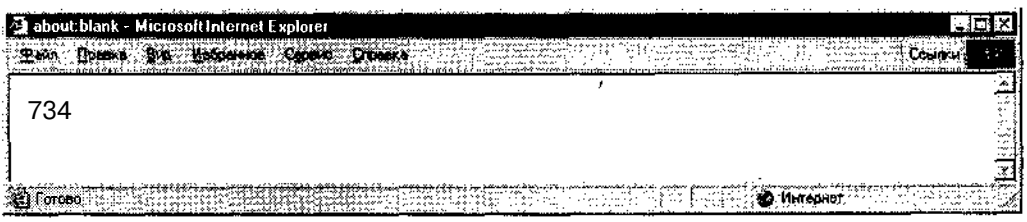


Рис. Р.1. Результат выполнения скрипта

Обратите внимание на строку global $a, $b, $d. В PHP для того чтобы использовать ранее определенные переменные, необходимо в самой функции их также определять как global. При этом значения, которые получили переменные в самой функции, в последующем использовании могут быть изменены. Например, если добавить в вышеописанную нами функцию строку `$a++;`, то результат выполнения программы будет следующий: `$a=4`, `$b=4`, $d=7. В дальнейшем нам придется оперировать с измененным значением переменной.

Существует еще один способ определения и последующего взаимодействия переменных в необходимой функции. Например:

```
$a = 1;
$b = 2;
Function Sum () {
$GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}
sum ();
echo $b;
```

В данном случае `$GLOBALS` является ассоциативным массивом.

ВНИМАНИЕ

Само имя массива должно быть постоянным, оно зависит даже от регистра.

Результат программы будет следующим. Браузер выведет сумму значений переменных $a и $b, программа произведет запись в переменную $Ь, значение которой станет отличной от первоначального задания.

Обратите внимание на то, что в PHP можно использовать статические переменные. Для наглядности приведем несколько примеров и опишем их.

Пример 9.1. Обычные переменные

```
<?php
Function nonestatic()
{
$a=0;
echo $a;
$a++;
}
nonestatic();
nonestatic();
nonestatic();
?>
```

Результат выполнения программы представлен на рис. 9.2.

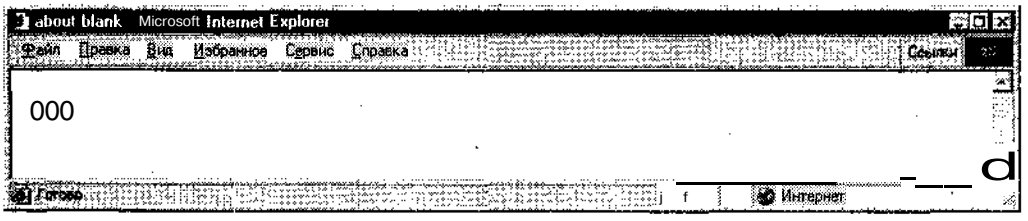


Рис. 9.2. Результат выполнения скрипта

Прежде чем мы опишем результат работы этой функции, скажем пару слов про операцию $a++. Она, как и в языке C++, позволяет прибавлять к значению переменной по единице. Если функция nonestatic () была вызвана 3 раза, то, соответственно, переменная примет значение, равное 3 (если, конечно, она была определена как $a=0). На основании такой операции постараемся объяснить принципы работы функции, в которой применяются переменные, определенные как статические (static). В данном случае переменная в функции не определена как стати-

ческая, а ей просто присвоено значение, равное 0. В результате при выполнении программы получим, что значения переменной $a изменяться не будут. После выполнения функции echo $a; на экран будет выведено значение, равное 0, и так три раза подряд, т. е. «000». Вот что значит не использовать переменную как статическую.

Пример 9.2. Статические переменные

```
<?php
Function statics()
{
static $a=0;
echo $a;
$a++;
}
statics();
statics();
statics();
?>
```

Результатом выполнения этого скрипта будет «012». Можно сделать вывод, что переменная $a после того, как ее обозначили статической, начала менять свои значения и при этом их сохранять. Это может значительно помочь при программировании циклов и рекурсий (явление, когда функция вызывает сама себя), факториалов и других математических функций.

Пример 9.3. Пример рекурсии

```
<?php
Function test() {
static $count = 0;
static $a=1;
$count++;
$a=$a*$count;
echo $count;
echo $a;
if ($count < 5) {
test ();
}
```

```
echo $a;
}
?>
```

Данный скрипт вычисляет факториал 5!. Выполнение этой программы основано на рекурсии, т. е. возможности вызова функцией самой себя.

ВНИМАНИЕ

При написании таких программ нужно всегда помнить: чтобы значения переменных сохранялись, при их изменении необходимо объявлять их как статические.

Опишем пример 9.3. В строке `Function test() {` задали функцию, ее имя и открыли фигурную скобку для программирования внутренней части функции. Строки `static $count = 0; static $a=1;` необходимы для того, чтобы задать переменные с определенными значениями и при этом показать, что эти переменные будут статическими. Значения для переменных `$count = 0` и `$a= 1` выбраны не случайно. Чтобы программа работала правильно, в нашем случае необходимо, чтобы переменная $count на начальном этапе была равна нулю, так как именно к ней будет прибавляться по единице, и ее значение будет проверяться на количество проделанных прибавлений в операторе условия if (см. п. 13.7). Равенство переменной $a=1 также взято не случайно. После оператора `$count++;` используется выражение `$a=$a*$count;`. Именно на его основании и происходит вычисление факториала. Если бы переменная была равна нулю, то и последующие значения также были бы равны нулю, что недопустимо. Далее идет оператор условия `if`, который проверяет, меньше переменная $count пяти или нет. Если меньше, происходит рекурсия, т. е. вызов функцией самой себя, и все идет заново. Если же набранное число превышает указанное значение в операторе условия `if`, то выполнение функции завершается. При этом в самом конце происходит выполнение операции `$count-;`, что приводит к возврату значения переменной к нулю. Результат был выведен раньше, поэтому выполнение программы завершается и результаты, полученные в ходе работы, выводятся на экран.

9.4. Переменные переменных

Иногда при программировании возникает необходимость использовать значение переменной в качестве имени. Это явление называется переменные переменных. В некоторых случаях это является очень удобным способом реализации поставленной задачи. Приведем пример:

```
$a = 'best';
```

Переменной $a задается значение best. В PHP существует такая возможность, что значение самой переменной может использоваться с именем новой переменной, которой в последующем будут присвоены следующие значения. Это очень удобно,

когда прежде всего необходимо сохранить место, т. е. размер программы. Но нужно быть предельно внимательным. Для примера рассмотрим реализацию следующей программы:

```
<?php
$a = "hello";
$$a = "world";
echo "$a $hello";
?>
```

Первая строка присваивает значение переменной `$a = "hello";`. Во второй строке `$$a="world";`, задав таким образом переменную $a, мы тем самым создали новую переменную имени hello, и не только создали, но и присвоили. Теперь в любом коде программы вы без какой-либо проблемы сможете использовать значение переменной $hello = "world". Чтобы убедиться в этом, в самом конце программы приведена функция `echo "$a $hello";`, которая позволит получить результат в виде «hello world».

ВНИМАНИЕ

При использовании такого способа существует возможность определения переменных большого количества (три и более). Но это приводит только к путанице в программе и трудностям в понимании кода. Старайтесь не делать это более, чем для двух разных переменных.

Рассмотрим еще один пример, похожий на предыдущий, но отличающийся реализацией переменных:

```
<?php
$a = "PHP";
$$a = "is";
$$$a = "GOOD!";
echo "$a ${$a} ${${$a}}";
echo "$a $PHP $is";
?>
```

В данном примере обратите внимание на строку `$$$a = "GOOD!";`. Эта строка объясняет то, что было сказано ранее, т. е. количество определяемых таким образом переменных может быть очень большое (в данном примере — три). Также посмотрите, как они определяются — перед переменной стоит три знака доллара. Строка `echo "$a $$a ${${$a}}";` позволяет вывести фразу «PHP is GOOD!». Функция `echo "$a $PHP $is";` выводит этот же результат. Две функции были употреблены для того, чтобы вы смогли увидеть разнообразные способы работы и обращения к таким переменным.

ВНИМАНИЕ

Если само имя переменной вы записываете как $$a[1], то синтаксический анализатор должен знать, желаете ли вы использовать $a[1] в качестве переменной или же вам нужна переменная $$a и индекс [1] от этой переменной. Синтаксис при решении этой неоднозначности будет следующий: ${$a[1]} для первого случая и ${$a}[1] для второго.

9.5. Передача параметров скрипту при запуске из командной строки

Передача параметров скрипту может происходить различными путями, один из них — при запуске из командной строки. Чем хорош и плох этот способ, судить вам. Это просто один из способов быстро ввести необходимые параметры для проверки правильности написания программ.

СОВЕТ

Использование такого метода скорее всего подходит в том случае, если на вашем компьютере отсутствует Web-сервер. Но в любом случае, чтобы научиться более профессиональному программированию, необходимо устанавливать сервер.

Чтобы произошло выполнение PHP-скрипта, прежде всего убедитесь в правильности пути к самому файлу `php.exe`. Если такого файла у вас нет, то и сама программа работать не будет. В этом случае необходимо установить PHP.

Например, `file.php` содержит:

```
<? echo "$argv[1] \n $argv[2]"; ?>
```

При запуске скрипта в случае отсутствия Web-сервера на вашем компьютере значения переменных передаются в сам скрипт следующим путем:

```
C:/Program files/PHP/>php.exe file.php 10 20
```

Это строка вводится в командной строке.

Результатом выполнения этой программы будут следующие строки на экране вашего монитора:

```
X-Powered-By: PHP 4.0.5
Content-type: text\html
10
20
```

Другой способ введения переменных основан на использовании знака амперсанд (&). Например, файл `file.php` содержит:

```
<? echo "$a \n $b"; ?>
```

Если данный файл вызвать, набрав в командной строке

```
C:/Program files/PHP/>php.exe -f file.php &a=10&b=20
```

то, после выполнения скрипта вы увидите:

```
10
20
```

В данном случае не будут выведены такие строки:

```
X-Powered-By: PHP 4.0.5
Content-type: text\html
```

В этом заключается принципиальная особенность реализации того или иного способа. Можно сделать вывод, что второй способ более рационален и выполняется быстрее, сосредоточивая работу центрального процессора на необходимых нам операциях.

9.6. Формы HTML (Get/Post)

Как и для скриптов на языке Perl, в Интернете есть множество сайтов, содержащих десятки готовых PHP-программ. Такие ресурсы делятся на два вида:

- наборы функций,
- классы.

Программа на языке PHP, как правило, не «живет сама по себе», а применяется для обработки запросов через интерфейс CGI. Даже если интерпретатор PHP встроен в сервер как модуль, с точки зрения самой пользовательской программы, она работает через CGI. А для чего творцам чудесных Web-сайтов нужен CGI? Конечно же, для обработки форм. Именно на скрипт указывает параметр action-тега `<form action = "doit.phtml">`.

В PHP программисту не требуется каким-то особым образом извлекать данные формы. В момент начала выполнения скрипта уже существуют и определены переменные, соответствующие одноименным полям. Например, если форма имеет вид:

```
<form action = "doit.phtml">
<input name=f1>
<input name=f2>
<input type=submit name=do value="Поехали">
</form>
```

то при старте скрипта `doit.phtml` в нем уже определены переменные `$f1`, `$f2`, `$do`. Можно указать имя поля в форме как

```
<select name="sel[]" multiple>
...
</select>
```

В таком случае на момент скрипта, обрабатывающего эту форму, будет определен массив $sel, содержащий выбранные пункты списка select.

По умолчанию и чаще всего форма передается HTTP-методом Post, а статические страницы пользователь обычно получает методом Get. В последнем случае вы тоже можете передавать параметры выполняющемуся скрипту через так называемую строкузапроса, т. е. через URL. Выглядит такой URL, например, следующим образом:

```
www.domain.name.ru/script.phtml?a=5&b=no&c=%2f
```

ВНИМАНИЕ

Собственно параметры начинаются после знака «?» и состоят из параметров «имя–значение», разделенных знаком «&».

Как и в случае с полями формы, программа `script.phtml` получит переменные $a, $b, $c в соответствии с собственным содержанием «5», «по» и «/».

СОВЕТ

Обратите внимание на значение переменной $c. Вообще-то лучше не полениться и зайти на сайт www.w3c.org, вдумчиво прочитать документацию о протоколе HTTP 1.1. Это очень поможет вам в профессиональном росте.

Можно комбинировать оба эти метода, т. е. создавать формы и указывать обработчику параметры через URL:

```
<form method = POST action = "doit.phtml?color=black&hold=1">
...
</form>
```

Когда форма представлена сценарию PHP, любые переменные от этой формы станут автоматически доступными сценарию PHP. Если конфигурационная опция track_vars включена, то эти переменные будут зафиксированы в ассоциативных массивах $HTTP_POST_VARS (ДЛЯ Post), $HTTP_GET_VARS (ДЛЯ Get) или $HTTP_POST_FILES согласно источнику рассматриваемой переменной. Например:

```
<form action="test.php" method="post">
Name: <input type="text" name="name1"><br>
<input type="submit">
</form>
```

ВНИМАНИЕ

Когда вышеупомянутая форма используется, значение от текстового ввода будет доступно в $HTTP_POST_VARS['name1']. Если конфигурационная директива register_globals включена, то переменная будет также доступна как $name1 глобальная.

После того как пользователь нажмет в форме кнопку «Submit», в PHP-скрипт test.php методом Post передадутся данные из формы, а обратиться, например, к текстовому полю из этого скрипта можно будет через переменную $name1.

PHP также работает с массивами в контексте переменных формы. Вы можете, например, группировать связанные переменные или использовать эту особенность, чтобы получить значения, выбранные в поле List Box.

Рассмотрим пример. Предлагается создать два файла: первый — index.html. В нем будет размещаться сама форма, переменные через которую будут передаваться скрипту в test.php — во второй созданный файл.

Пример 9.4. Файл index.html

```
<html>
<head>
<title>Программа</title>
</head>
<body>
<form action="test.php" method="post">
Name: <input type="text" name="personal[name]">
Email: <input type="text" name="personal[email]">
Beer:
<select multiple name="beer[]">
<option value="warthog">Warthog
<option value="guinness">Guinness
</select>
<input type="submit">
</form>
</body>
</html>
```

Результат выполнения программы представлен на рис. 9.3.

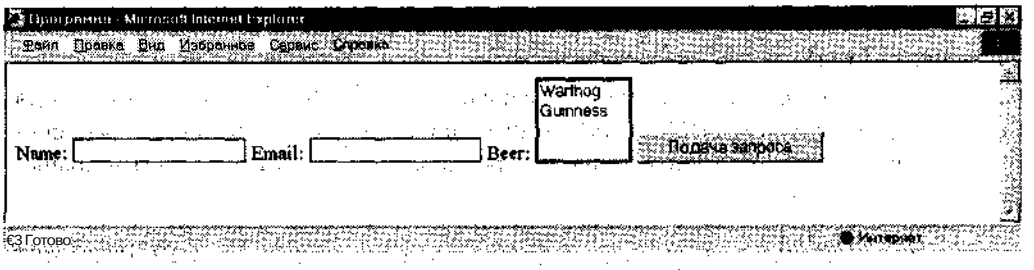


Рис. 9.3. Полученная форма

Пример 9.5. Файл test.php

```
<?php
echo "Имя: ".$personal["name"];
echo "Email: ".$personal["email"];
for ($i=0;$i<sizeof($beer);$i++)
echo $beer[$i]."<br>";
?>
```

Опишем принцип работы программы. В строке `Name: <input type="text" name="personal[name]">` задается переменная как элемент массива. Далее идет подобная строка `Email:<input type="text" name="personal[email]">`, выполняющая ту же функцию. Элемент select используется для создания выбора списка. При отправке формы серверу пересылается выбранная строка `<select multiple name="beer[]">`. Внешне элемент select может быть представлен либо как обычный, либо как раскрывающийся список, в зависимости от того, указан или нет атрибут size, В самом теге используются собственные атрибуты name и multiple. Атрибут name определяет имя элемента. Если указан атрибут multiple, то браузер позволяет выбрать более одного элемента списка.

Рассмотрим файл `test.php`. После того как пользователь ввел данные в форму, указанным переменным присваиваются эти значения. Как только пользователь нажмет на кнопку «Submit», начнется передача этих значений переменных в наш PHP-скрипт для последующей реализации. Например, в строках Name и Email пользователь ввел следующие данные: Name - Sasha, Email - `sashatest@mydomain.ru`. Из первых двух строчек программы, написанной на PHP:

```
echo "Имя: ".$personal["name"];
echo "Email: ".$personal["email"];
```

видно, что при выполнении данного скрипта одним из результатов проделанной работы будет вывод таких фраз «`Имя:` Sasha, Email: `sashatest@domain.ru`».

Это очень простая программа, но она показывает, что как только вы передали переменные, полученные в форме, в сам скрипт, то сразу можете без каких-либо про-

блем выполнять любые операции с этими переменными, конечно, при разумных условиях. Также обратите внимание на то, что в данном примере работа ведется не с отдельной переменной, а с одним массивом, что облегчает задачу, хотя на первый взгляд все выглядит очень сложно. Поверьте, это временное чувство. Уверены, после того как вы напишете самостоятельно пару раз этот скрипт, не заглядывая в книгу, все ваши волнения перестанут быть вам помехой в дальнейшем. Что касается строк;

```
for ($i=0;$i<sizeof($beer);$i++)
echo $beer[$i]."<br>";
```

можно сказать одно: данная функция является функцией цикла, в котором происходит прибавление по единице к переменной `$i`, при этом функция echo () выводит полученные результаты. Как таковая программа несложная, и сама работа с переменными также не представляет никаких трудностей. Самое главное запомнить основные нормы и правила при написании таких программ.

9.7. Передача значений переменных, соответствующих кнопкам формы

В PHP можно передавать значение переменных из форм соответствующим кнопкам формы. В скрипт передаются все именованые элементы форм. Таким образом, если у кнопки указан атрибут `name="button_name"`, то будет установлена соответствующая переменная в запросе, а ее значение будет равно атрибуту value (заметьте, что это — текст на кнопке). Если у кнопки опущен элемент `name`, то соответствующая переменная не будет установлена в запросе. Обратите внимание: чтобы эти переменные автоматически создавались, необходимо выполнение следующих параметров конфигурации (табл. 9.1).

Таблица 9.1. Параметры конфигурации

Параметры	Значение
egister_globals boolean	По умолчанию включено. Если выключено, то переменные не задаются
gpc_order string	По умолчанию GPC. Определяет, нужно ли принимать Get, Post и Cookie. Если оставить, например, GC, то данные, переданные по Post, не будут определены
track_vars boolean	По умолчанию включено. Определяет, создавать ли глобальные массивы `$HTTP_ENV_VARS`, `$HTTP_GET_VARS`, `$HTTP_POST_VARS`, `$HTTP_COOKIE_VARS` и `$HTTP_SERVER_VARS`

Кроме этого, в форме в качестве кнопки «Submit» можно применять картинку, например:

```
<input type=image src="image.gif" name="sub">.
```

Когда пользователь нажимает на какую-нибудь часть картинки, сопроводительная форма будет передана на станцию с двумя дополнительными переменными `sub_x` и `sub_y`. Они содержат координаты щелчка пользователя в пределах изображения. Опытный пользователь может заметить, что фактические имена переменной, посланные программой просмотра, содержат скорее точку, чем символ подчеркивания, но PHP преобразовывает эту точку автоматически в символ подчеркивания.

9.8. HTTP-Cookies

PHP поддерживает Cookies HTTP, как определено в соответствии с спецификацией Netscape. Cookies — механизм для накапливания данных в отдаленной программе просмотра, помогающий производить идентификацию возвращающихся пользователей. Вы можете устанавливать Cookies при помощи функции `setcookie()`. Cookies — часть HTTP, поэтому функция setcookie должна быть вызвана прежде, чем любой запрос послан программе просмотра (браузеру). Такое же ограничение касается функции `header()`. Любые Cookies, посланные вам от клиента, будут автоматически преобразовываться в PHP-переменную точно так же, как данные Get и Post метода.

Передавать данные скрипту можно не только посредством параметров в URL или в полях формы. Практически все современные браузеры понимают Cookies. В PHP вы можете пользоваться ими совершенно свободно. Когда браузер отдает Cookies вашему скрипту, вы просто получаете переменную с именем Cookies. Отдать же Cookies браузеру вы можете функцией

```
setcookie("имя","значение",[необязательные параметры]);
```

Помните, что эта функция должна выполняться для любого вывода текста страницы.

ВНИМАНИЕ

Если вы желаете назначить multiple-переменные к одиночному Cookies, просто добавьте [] к названию Cookies. Например:

```
setcookie ("MyCookie[]", "Testing", time()+3600);
```

СОВЕТ

Если у вас возникнет необходимость сделать так, чтобы Cookies сохранялся у клиента какое-то время, необходимо просто задать Cookies время действия (укажешь им январь 3000 года, и будут Cookies тысячу лет «болтаться»).

При помощи Cookies PHP позволяет хранить в браузере клиента информацию. Для этого реализован механизм Cookies-массива. Например, после выполнения такого фрагмента кода:

```
<?
$myData['id']=2345;
$myData['name']='Петя';
$myDate['email']='mynew@domainname.ru';
setcookie('myData[]',$myData);
?>
```

Все ваши скрипты будут иметь в начале выполнения хэш-массив `$myDate` с точно тем же содержимым. Помните, что количество информации в Cookies ограничено, так что старайтесь ею не злоупотреблять.

СОВЕТ

Не храните в Cookies текст на русском языке или двоичную информацию, лучше предварительно закодируйте ее с помощью функции rawurlencode().

Если же вам очень хочется завести о пользователе целую уйму персональных данных, а делать это при помощи Cookies неудобно, то вам стоит выбрать одну из схем «ведения пользователя». Их существует множество — это и модули к Web-серверу, и отдельные библиотеки к PHP, и множество готовых функций или классов, написанных на самом PHP. Вы легко можете сделать и свою систему ведения. Приведем пример. Включив этот файл в начало любого скрипта, вы можете вести пользователей, сохраняя информацию о пользователе в течение года.

Пример 9.6. HTTP-Cookies

```
<?
Function userTrack() {
Global $mySesID;
$db = dbmopen('/путь/userTrack.db','c');
if (! (isset($mySesID)&&dbmfetch($mySesID)));
{
$mySesID = uniqid();
dbminsert($mySesID,userTrackNew());
setcookie( 'mySesID', $mySesID, time()+60*60*24*365);
}
dbmclose($db);
$tmp = spit(<<'\t>>, dbmfetch($mySesID));
$user = array();
```

```
for ($i=0; $i<count($tmp); $i++)
{
.list($x,$y)=split('=',$tmp[$i]);
$user[$x]=$y;
}
return .$user;
}
?>
```

Подробно опишем программу и поясним, для чего нужна та или иная строка. `function userTrack () {` — задает функцию с именем userTrack. Чтобы можно было использовать переменные в функции строкой Global `$mySesID;`, объявляется переменная, в которой будут храниться данные Cookies. Также требуется открыть хэш-базу, в нашей программе это делается строкой `$db = dbmopen('/путь/userTrack.db','c');`. Постарайтесь указать путь как можно точнее, неправильное задание пути — это одна из распространенных ошибок при программировании у начинающих. Далее идет оператор условия if — (см. П. 13.7) — строка `if (! (isset ($mySesID) &&dbmfetch ($mySesID) ) ) ; {` — позволяет определять присутствие Cookies в хэш-базе, т. е. если Cookies нет в базе, то вносятся новые. `$mySesID = uniqid() ;` — задается новый уникальный идентификатор. Он и данные по умолчанию добавляются при помощи операции: `dbminsert ($mySesID,user TrackNew() ) ;`. После этого функцией setcookie () устанавливаются Cookies на год: `setcookie ( 'mySesID', $mySesID, time ()+60*60*24*365) ;`, время как год устанавливается в секундах именно строкой `time ()+60*60*24*365`. Вы можете изменять время на необходимое вам. После этой операции, которая реализуется при положительном исходе оператора `if`, завершается работа с ним. В самой функции определяется пустой массив: `$user = array () ;`. Далее данные необходимо перевести в хэш:

```
for ($i=0; $i<count($tmp); $i++)
{
- list($x,$y)=split('=',$tmp[$i]);
$user[$x]=$y;
}
```

`return $user;` — возвращает значения.

Теперь вы в состоянии работать с Cookies на высоком уровне. Этот файл можно включить в самом начале, не задумываясь о его содержимом, так как значение по умолчанию можно задать своей функцией:

```
<?
Include('userTrack.inc');
```

```
Function userTrackNew()
{
Return "id=5\tname=Сергей\tcolor=white";
}
$userinfo = userTrack() ; // получаем данные о пользователе
?>
```

В результате получим хэш-массив со всеми заданными нами настройками посетителя. Базы данных dbm* работают очень быстро, не требуют больших ресурсов и пользоваться ими можно и нужно.

СОВЕТ

Для хранения данных можно использовать СУБД (см. гл. 32), но при небольшом объеме данных это нерационально. Применение SQL СУБД оправданно, если большая часть ваших страниц также формируется из базы данных . Сейчас очень модно применять SQL-серверы везде, где надо и где не надо. На SQL-базах делают даже чаты. При небольших объемах данных все же следует использовать простые средства: текстовые файлы, хэш-базы (dbm*) и т. д.

9.9. Системные переменные

PHP автоматически делает системные переменные доступными как самые обычные переменные PHP. Приведенный пример показывает работу с системной переменной. Из него видно, что она используется также, как обычная:

```
ECHO $HOME;
```

В данном случае системная переменная $HOME.

СОВЕТ

Информация, приходящая через механизмы Get/Post/Cookie, также автоматически создает PHP-переменные, поэтому лучше всего читать переменную из среды, чтобы удостовериться, что вы получаете правильную версию. Такие функции, как getenv() и putenv(), могут быть использованы также для установления системной переменной:

```
getenv ("UNIQID=$uniqid");
putenv("UNIQUID=$uniquid");
```

Чтобы увидеть список всех системных переменных, вам необходимо воспользоваться функцией phpinfo().

9.10. Точки в именах входящих переменных

Как правило, PHP не изменяет названия переменных, когда они передаются в сценарий программы. Однако заметьте, что точка (полная остановка) не допускается в имени переменной PHP:

```
$varname.ext; /* недопустимое имя переменной */
```

Это можно описать так: синтаксический анализатор видит переменную, названную $varname, сопровождаемую оператором конкатенации строк, за которым следует barestring (т. е. неупомянутая строка, которая не соответствует любой известной клавише или зарезервированным словам) ext. Очевидно, что это не ведет к необходимому результату.

ВНИМАНИЕ

Заметьте, что в этом случае PHP автоматически заменит любые точки во входящих именах переменной символами подчеркивания.

9.11. Как проверить, был ли отмечен checkbox в форме

Следующая программа, которая будет проверять, был ли отмечен checkbox (поле для галочки) в форме, поможет вам не только освоить работу с формами, но также изучить новые для вас функции. Прежде всего создадим файлы index.html Итest.php.

Пример 9.7. Проверка checkbox

```
<html>
<head>
<title>Программа</title>
</head>
<body>
<form action="test.php" method="POST">
<input type="checkbox" name="checkme" value="yo">
<input type="checkbox" name="checkme1" value="yo1">
<input type="checkbox" name="checkme2" value="yo11">
<input type="submit" value="ok">
</form>
</body>
</html>
```

Думаем, что в этом примере нет ничего сложного для вас, особенно если вы изучали ранее HTML. Единственное, на чем хотелось бы заострить внимание, — на строке <input type="checkbox" name="checkme" value="yo"> . Обратиться к этой части окна можно будет только в том случае, если задать ей имя, как в случае name ="checkme". Далее вся работа с формой будет вестись именно с учетом данной переменной "checkme". Способов обращения к value="yo" может быть несколько. В данном случае рассмотрим, как при помощи операций с переменными можно решить одну и ту же задачу разными путями. Создадим файл test.php. Код, который он должен содержать, приведен ниже:

```
<?
if (isset($checkme)or isset($checkme1) or isset($checkme2) )
                    // здесь "checkme" — имя checkbox
{
echo "Check sets"; // если checkbox отмечен,
// выполняем какие-то действия
}
?>
```

Рассмотрим строку:

```
if (isset($checkme)or isset($checkme1) or isset($checkme2) )
```

Это оператор условия. Здесь проверяется, установлена ли галочка в рамке или же она отсутствует. Этим занимается функция isset($checkme). Если галочка установлена, эта функция возвращает значение true, и выполнения условия начинается. Иначе же все наоборот (значение — false, и выполнение условия не начинается). В результате видно, что если рамка checkbox является занятой (т. е. галочка установлена), то на экране браузера выведется следующая фраза: «check sets», в противном случае ничего выводиться не будет.

Опишем еще несколько способов решения этой задачи. Обратите внимание на value="yo" в файле index.html. Рассмотренная выше программа теперь будет решаться с учетом функции strlen();. Эта функция будет проверять количество символов в строке, в нашем случае она будет проверять количество символов переменной name="checkme" value="yo". Как видно, значение самой переменной value ="yo", ичисло, которое будет передавать данная функция, должно быть больше нуля, для того чтобы оператор условия выполнился, т. е. программа перешла на следующий этап. Так как количество символов каждой из переменных в трех рамках превышает значение «нуль», то в момент, когда пользователь поставит в любую из них галочку, произойдет выполнения условия:

```
<?
if ((isset($checkme) && strlen($checkme)>0)or ( isset($checkme1) &&
strlen($checkme1)>0)or (isset($checkme2) && strlen($checkme2)>0) )
```

```
{
echo "Check is set";
}
?>
```

Существует еще один способ, один из самых простых и актуальных: в форме перед тегом <input type='checkbox'> поместить тег <input type='hidden'> с тем же самым значением параметра 'name'. Например:

```
<form ...>
...
<input type="hidden" name="checkme" value="0">
<input type="checkbox" name="checkme" value="1">
...
</form>
```

Теперь если checkbox отмечен, в переменной $checkme передастся «1», в противном случае «О».

СОВЕТ

Как вы убедились, для решения одной задачи существует не один способ. Поэтому не бойтесь экспериментировать. Ведь именно методом практических экспериментов можно познать все особенности языка.

9.12. Определение типов переменных

Поскольку PHP определяет типы переменных и конвертирует их, как необходимо, не всегда очевидно, какой тип будет иметь данная переменная в любой момент. Для определения конкретных типов переменных, чтобы не совершать ошибки при работе с ними, PHP включает несколько функций, которые выясняют, какого типа та иди иная используемая переменная. Это функции gettype(), is_bool(), is_int(), is_string(), is_array() и is_object().

gettype()

возвращает тип переменной:

```
string gettype(mixed var)
```

Как видно отсюда, если мы запишем следующий пример:

```
$name = 5;
$String = gettype($name);
```

переменная $String становится автоматически строковой переменной, а переменная $name, как видно из примера, является типом integer. Следовательно, переменной $String будет присвоено значение 'integer', т. е., $String = 'integer';. Таким образом, при выполнении программы:

```
<?
$name = 5;
$String = gettype($name);
echo $String;
?>
```

в окно браузера выведется название типа переменной, которое обозначается как $name: "integer".

Ниже приведен список типов, которые данная функция определяет:

- boolean,
- integer,
- double,
- string,
- array,
- object,
- null,
- unknown type (неизвестный тип).

```
is_long()
```

Определяет, переменная имеет тип integer или другой тип. Описание:

```
bool is_long(mixed var);
```

Принцип работы данной функции заключается в следующем. Если тип переменной var integer (целое), то функция принимает значение true, иначе false. Пример программы работы данной функции:

```
<?
$name = 5;
$secname = 'Sasha';
$String = is_long($name);
$Secstring = is_long($secname);
?>
```

При выполнении данного скрипта переменной $string будет присвоено значение true, а переменной $Secstring - false (так как тип переменной `$secname` не является integer).

Функция `is_double()` аналогична `is_long()`, только она ориентирована на переменную типа double, а не на integer, как функция `is_long()`.

Функция `is_string()` работает с переменными типа string, принцип работы такой же, как и у изложенных выше функций. Описание:

```
bool is_string(mixed var)
```

Функция `is_array()` определяет переменную типа array. Принципы работы аналогичные изложенным выше. Описание:

```
bool is_array (mixed var)
```

Функция is_object () идентифицирует переменную типа object. Описание:

```
bool is_object(mixed var)
```

Употребление таких функций помогает решать задачи, прежде всего имеющие узкую спецификацию. Также это помогает точно определять тип самой переменной и при этом работать с ним как со значением переменной.

Заключение

В данной главе вы познакомились с переменными PHP. Как было ранее сказано, PHP позволяет использовать переменные. При этом их не нужно описывать так, как это делается в Visual Basic или Pascal. Вы просто вводите необходимую переменную там, где вам нужно, и тогда, когда вам это нужно. Имена переменных начинаются с символа $. Переменные могут быть трех типов: целые, с плавающей запятой и символьные строки. Хотя разделение на типы условное, каждая функция стремится использовать правильный тип автоматически.

С помощью PHP очень просто обрабатывать данные, полученные из форм, так как для каждого поля ввода в вашей форме автоматически создается переменная.

PHP поддерживает HTTP-Cookies. Cookies — это механизм для сохранения данных в браузере. Таким образом можно отслеживать или идентифицировать пользователя. Чтобы сохранить переменную на компьютере посетителя вашей странички, используется функция setcookie (). Любое значение Cookies, сохраненное таким образом, автоматически превращается в переменную, так же как и в случае с формами.

Глава 10
Предопределенные константы и их использование

В PHP определено несколько констант и имеется механизм для определения других констант во время выполнения программы. Следует также отметить тот факт, что сами константы очень схожи с переменными, но они определяются с помощью функции `define()`, и после этого им уже нельзя присвоить какое-либо другое значение. В этой главе вы сможете познакомиться более детально со всеми аспектами констант и изучить предопределенные константы и их использование.

Список предопределенных констант (доступны всегда):

`__FILE__`

имя файла, который анализируется в данный момент скрипта. При использовании во включенном (included) или необходимом (required) файле дается имя включенного файла, но не имя родительского.

`__LINE__`

номер строки в текущем файле скрипта, анализируемой в данный момент. При использовании во включенном (included) или необходимом (required) файле дается позиция во включенном файле, но не в родительском.

`PHP_VERSION`

строковое представление версии PHP, используемой в настоящий момент, например `'3.0.8-dev'`.

`PHP_OS`

имя операционной системы, в которой работает синтаксический анализатор PHP, например `'Linux'`.

`true`

значение «истина».

`false`

значение «ложь».

`E_ERROR`

указывает на ошибку, отличную от ошибки анализатора, которую невозможно устранить.

E_WARNING

указывает места, где имеется что-то ошибочное, но работа все равно продолжится; их может «перехватывать» сам скрипт. Примером может быть ошибочное регулярное выражение В ereg().

E_PARSE

в файле скрипта обнаружен недопустимый синтаксис. Восстановление невозможно.

E_NOTICE

что-то произошло, возможно ошибка. Выполнение продолжается. Примером может служить использование строки без кавычек в качестве хэш-индекса или обращение к переменной, которой не присвоено значение.

E_ALL

равна сумме всех констант E_*. Использование этой константы в функции error_reporting() приведет к выдаче отчета по всем проблемам, обнаруженным PHP.

В PHP константы E_* обычно используются с функцией error_reporting(), чтобы установить уровень выдачи сообщений об ошибках (см. гл. 16).

ВНИМАНИЕ

С помощью функции define() можно определить дополнительные константы.

Обратите внимание, что это константы, а не макросы в стиле языка C. Константа может представлять только допустимые скалярные данные.

Пример 10.1. Определение констант

```
<?php
define("CONSTANT", "Helloworld.");
echo CONSTANT; // выводит "Hello world."
?>
```

Пример 10.2. Использование констант __FILE__ и __LINE__

```
<?php
function report_error($file, $line, $message) {
echo "Ошибка в файле $file в строке $line: $message";
}
report_error(__FILE__, __LINE__, "Что-то не так!");
?>
```

Результат выполнения программы представлен на рис. 10.1.

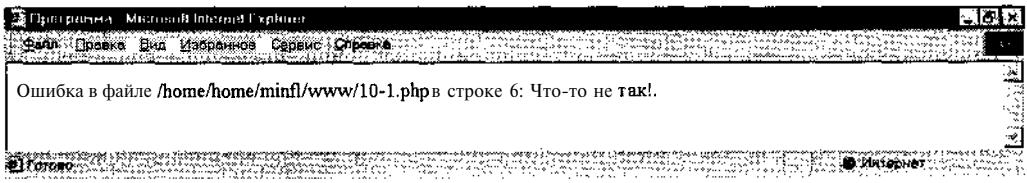


Рис. 10.1. Результат выполнения скрипта

Заключение

Исходя из этой небольшой главы, хотелось бы отметить, что наиболее важными аспектами в использовании констант являются:

- в PHP определено несколько констант и имеется механизм для определения других констант во время выполнения программы;
- сами константы очень схожи с переменными, но они определяются с помощью функции `define()`;
- константы E_* обычно используются с функцией error_reporting(), чтобы установить уровень выдачи сообщений об ошибках;
- константа может представлять только допустимые скалярные данные.

Глава 11
Выражения

В PHP почти все является выражениями. Простейший и наиболее точный способ определить выражение — это «что-то, имеющее значение», например константы и переменные. Когда вы набираете $a = 5, вы присваиваете значение 5 переменной $a.

После этого, если вы напишете $b = $a, вы будете ожидать того же, как если бы вы написали $b = 5. Другими словами, $b это также выражение со значением 5. Если все написано правильно, то именно так оно и случится. Как таковые выражения встречаются практически в каждой строке документированного кода. В данной главе рассмотрим основные аспекты использования выражений.

Выражения могут быть так различны, что иногда это может вас запутать. Есть много способов, которыми можно описать то или иное действие. И так, как вам это удастся сделать, будет работать и ваш скрипт. Например, чтобы заменить в текстовом поле $text нажатие клавиши «Enter» на тег
, а главное, чтобы все было в одну строчку, необходимо воспользоваться следующей строкой:

```
$text=ereg_replace ("\n", "<br>", $text);
```

И для этого совсем не нужно пользоваться функцией nl2br.

Е! этой главе вы все это подробно изучите, а также мы приведем несколько примеров, направленных на конкретную поставленную задачу:

- примеры выраженийфункций;
- скалярные и нескалярные выражения;
- регулярные выраженияPHP/FI2 и выражения присваивания;
- выражения сравнения;
- совмещенныевыражения;
- выражения условных операторов;
- логические значениявыражений;
- пример счетчика посещений на PHP.

11.1. Примеры выражений функций

Представьте себе программу, состоящую из большого числа подпрограмм. Чтобы выполнить подпрограмму в самой программе, надо при этом возвратить значение переменной, т. е. результат самой работы функции. Функции являются более сложными примерами выражений. Например:

```
Function mynew()
{
return 5;
}
```

Предположим, что вы знакомы с концепциями функции (см. ч. 4), вы считаете, что $c = mynew() практически то же самое, что $c = 5, и вы правы. Функции — это выражения с тем значением, которое они возвращают. Так как mynew() возвращает 5, значение выражения mynew() — 5.

Приведем пример, показывающий работу функции.

Пример 11.1. Функция return

```
<?
$a = 5;
Function mynew()
{
$name = 5;
$a = $name + 5;
return $a;
}
```

```
$a = mynew();
echo $a;
?>
```

Функция mynew() работает с переменными $a, $name. Что конкретно выполняет данная функция, можно понять по строчке $a = $name + 5; — именно в этой строчке значение переменной $a будет равно 10. После этого идет оператор return $a;, который возвращает значение, равное переменной $a = 10. После того как функция была написана, необходимо во время программы вызвать ее. Для этого предназначена строка $a= mynew();. Именно в этой строке переменной $a, которой вне функции (вначале программы) было задано значение 5, функцией задается новое значение. Убедиться в этом помогает строка echo $a;, которая выводит на экран значение переменной. В нашем случае число, которое будет выведено на экран, равно 10.

При работе массивов с функциями специально объявлять массив или хэш не надо. Но в некоторых случаях это бывает полезным. Например, если ваша функция возвращает массив, то она должна вернуть именно массив, а не неопределенное значение.

Пример 11.2. Функции работы с массивами

```
<?
Function setuparray($num)
{
if ($num>0) return $a[$num] = $num;
return $a;
}
?>
```

Эта функция возвращает массив, если $num больше нуля, и неопределенное значение — в противном случае, что затрудняет работу с результатом ее выполнения:

```
<?
...
$foo = SetArray(0);
$bar = $foo[0];
...
?>
```

Здесь вам поможет функция array(), возвращающая массив, в том числе и пустой. В другом случае стоит сразу инициализировать переменную $a = array() или вернуть пустой массив как return array(). Первый вариант предпочтительнее.

В PHP есть даже некоторая объектность, т. е. вы можете создавать классы, методы и т. д. Как правило, классы хороши в больших приложениях или как средство ограничение области видимости переменной.

СОВЕТ

Несмотря на то что объектный подход сейчас пользуется большой популярность, мы не рекомендуем очень уж активно им пользоваться.

Реализация классов в PHP 4 намного эффективнее, чем в PHP 3.

11.2. Скалярные и нескалярные выражения

PHP поддерживает три скалярных типа значений: целое, число с плавающей точкой и строки (скалярные выражения нельзя «разбить» на более маленькие части, как, к примеру, массивы). PHP поддерживает два нескалярных (составных) типа: массивы и объекты. Каждое из таких значений может быть присвоено переменной или возвращено функцией.

Пример 11.3. Массивы и объекты

```
<?
$a = 5; // целое число
$a = '5' ;  // строка
$a = 5.5;  // число с плавающей запятой
$a[0] = 5;  // массив
$a['пять'] = 5; // хэш-массив
?>
```

Последняя строка показывает очень популярный и удобный тип данных — хэш-массив (hash-array). В принципе это обычный массив, только индексом у него выступает строка, а не целое число. PHP позволяет создавать также многомерные структуры вида массив хэш-массивов, как например:

```
$name[5]['ой'][6] = 'не может быть';
```

В принципе массив — это тоже хэш, только в нем индексами являются строки, соответствующие числам. Например, фраза `$name[5] = 0;` создает массив `$name` из шести элементов (индексы считаются от нуля, как в C), но только один элемент реально существует, остальные просто не определены. На практике это не сильно мешает, помните только, что это не C и не Паскаль.

11.3. Регулярные выражения PHP/FI 2 и выражения присваивания

Регулярные выражения используются для сложного манипулирования строками в PHP/FI. Поддержка интерфейса между скриптом и регулярными выражениями осуществляется через следующие функции: reg_match (), reg_replace ()

и reg_search(). Первый аргумент всех трех функций — это строка, задающая регулярное выражение и состоящая из регулярных и специальных символов. Регулярные символы имеют то же значение, что и в других командах Unix, а специальные символы имеют специальное значение (табл. 11.1).

Таблица 11.1. Регулярные выражения PHP/FI 2.x выражения присваивания

Символ	Значение
.	Специальный символ, который соответствует любому символу, за исключением символа новой строки. Используя его, можно задавать регулярные выражения, подобные a.b, которое соответствует любой трехсимвольной строке, начинающейся с a и заканчивающейся b
*	Это не конструкция, а суффикс, который означает, что предшествующее регулярное выражение может быть повторено определенное количество раз. В строке fo* символ * применяется к символу o, так fo* задает f с последующим любым количеством символов o. В случае нулевого количества символов o строка fo* будет также соответствовать f. Символ * всегда применяет к наименьшему возможному предшествующему выражению. Таким образом, fo* задает повторение o, а не повторение fo. Процесс сравнения обрабатывает конструкцию *, пытаясь выполнить настолько много повторений, насколько много их может быть найдено. Затем он продолжает обработку остальной части шаблона. Если впоследствии появится несоответствие с шаблоном, происходит возврат путем отбрасывания некоторых повторений *, если это делает возможным совпадение остальной части шаблона. Например, шаблон c[ad]*ar для строки caddaar, [ad]* сначала совпадает с addaa, но это не позволяет совпасть следующему символу a в шаблоне. Так, последнее совпадение [ad] отменяется, и следующий символ a пробуется вновь. Теперь шаблон соответствует
+	Подобен *, за исключением того, что требуется по крайней мере одно соответствие для предшествующего образца. Таким образом, c[ad]+r не совпадает с cr, но совпадет с чем-либо еще, что может быть задано шаблоном c[ad]*r
?	Подобен *, за исключением того, что позволяет задать нуль или более соответствий для заданного шаблона. Таким образом, шаблон c[ad]?r будет задавать строки cr или car, или cdr, и ничего больше
[...]	[начинает «множество символов», которое завершается символом]. В самом простом случае символы между этими двумя скобками формируют множество. Таким образом, [ad] задает символы a или d, и [ad]* задает любую последовательность символов a и d (включая и пустую строку), из чего следует, что шаблон c[ad]*r задает car, и т. д. Диапазон символов также может быть включен в множество символов, с помощью символа -, помещенного между двумя другими.

Продолжение табл. 11.1

Символ	Значение
	Таким образом, шаблон [a-z] задает любой символ нижнего регистра. Диапазоны могут свободно перемежаться одиночными символами, как в шаблоне [a-z$%.], который задает любой символ нижнего регистра или символы $, % либо точку. Обратите внимание, что символы, обычно являющиеся специальными, внутри множества символов не являются таковыми. Внутри множества символов существует полностью отличный набор специальных символов:], - и ^. Чтобы включить] во множество символов, нужно сделать его первым символом. Например, шаблон []a] задает символ] или a. Чтобы включить символ -, нужно использовать его в таком контексте, где он не может указывать диапазон, т. е. или первым символом, или сразу после диапазона
[^ . . .]	[^ начинает «исключающее множество символов», который задает любой символ, за исключением заданных. Таким образом, шаблон [^a-z0-9A-Z] задает любой символ, за исключением букв и цифр. ^ не является специальным символом в множестве, если только это не первый символ. Символ, следующий после ^, обрабатывается так, как будто он является первым (это может быть - или]). Является специальным символом, который задает начало строки
$	Подобен ^, но только задает конец строки
\	Имеет два значения: выводит на экран вышеперечисленные специальные символы (включая \) и задает дополнительные специальные конструкции. Так как \ экранирует специальные символы, \$ является регулярным выражением, задающим только символ $, а \[является регулярным выражением, задающим только [, и т. д. В основном \ с последующим любым символом соответствует только этому символу. Однако есть некоторые исключения: символы, в которых \ предшествует специальная конструкция
|	Задает альтернативу. Два регулярных выражения A и B с | между ними формируют выражение, которое задает что-либо, чему соответствует или A, или B. Так, выражение foo|bar выводит или foo, или bar, но никакую другую строку. | применяется к максимально большим окружающим выражениям. Только \(. . .\) вокруг выражений могут ограничивать количество символов |. Существует полная возможность перебора с возвратами, когда задано множество |
\b	Задает границы слова
\B	Задает отсутствие границ слова, т. е. указывает, что в данном месте не может быть границы слова
\w	Задает любой символ, являющийся составной частью слова

Окончание табл. 11.1

Символ	Значение
\W	Задает любой символ, который не является составной частью слова
\(... \)	Является конструкцией группировки, которая служит нескольким целям, например заключать в себя множество альтернатив | для других операций. Так, шаблон (foo|bar)x соответствует или foox или barx. Также служит для объединения нескольких элементов для операции над ними как элементом, например шаблон ba\(па\) * задает bananana с любым (нуль или более) количеством па

Обратимся к примеру, с которым мы уже разобрались, $a = 5. Легко заметить, что тут задействованы два значения: значение целой константы 5 и значение $a, которое также становится равным пяти. На самом деле здесь присутствует еще одно значение — присваивания. Само присваивание становится равным присваиваемому значению, в данном случае — 5. На практике это означает, что $a = 5, не обращая внимания на то, что оно равно выражению со значением пять, т. е. запись типа $Ь = ($a = 5) похожа на запись $a = 5; $b = 5; (точка с запятой отмечает конец выражения). Присваивания рассматриваются справа налево, поэтому вы также можете написать $Ь = $a = 5.

Еще один пример направления выражения — это предварительное и последующее увеличение и уменьшение. Пользователи PHP/FI 2 и многих других языков могут быть знакомы с записями variable++ и variable--. PHP расширяет возможность увеличения/уменьшения, делая их выражениями, как и в C. В PHP, подобно C, есть два типа инкремента и декремента — префиксный и постфиксный. При использовании префиксного инкремента/декремента значение переменной сначала участвует в выражении, а после этого изменяется значение самой переменной.

Пример 11.4. Инкремент/декремент переменной

```
<?
$a = 5;
$b = 2;
$a++;
$b--;
echo $a;
echo $b;
?>
```

При выполнении данной программы получим следующие результаты: переменная $a будет иметь значение 6, а $b — 1. Более подробно об этих операциях рассказано в главе 12.

Пример 11.5. Префиксный и постфиксный инкременты

```
<?
$a = 1;
$b = $a++;// b=1, a=2
echo $b;
$b=++$a; // b=3, a=3;
echo $b;
?>
```

11.4. Выражения сравнения

Можно сказать, что данные выражения являются одними из широко применяемых в программировании. Это касается как PHP, так и других языков программирования. Эти выражения имеют значение 0 или 1 (означает ложь или истину соответственно). PHP поддерживает знаки > (больше, чем), >= (больше или равно), = (равно), < (меньше, чем) и <= (меньше или равно). Эти выражения в основном используются внутри условий, например оператора if.

Пример 11.6. Выражения сравнения

```
<?
$a = 5;    // число
$b = '6b'; // строка
$c = $b.$a; // строка '6b5'
$d = "$a$b"; // строка '56b'
$e = $a +$b; // число 11
$f = $b + $a; // число 11
?>
```

Пример 11.7. Выражения сравнения с оператором условия

```
<?
$a = 5;
$b = 0;
if ($a>$b) echo "1";
if ($b>$a) echo "2";
?>
```

Интерпретатор произвольно приводит типы строка и число друг к другу. В примере 11.6 сравниваются числа 5 и 0, результат естественно положительный.

Что касается условных операторов, рекомендуем использовать операторы приведения такого типа, как в языке С:

```
<? If ((int)$a<(int)$b);?>
```

ИЛИ

```
<? If ((string)$a<(string)$b); ?>
```

11.5. Совмещенные выражения

Вы уже знаете, что для того чтобы увеличить значение $ана единицу, можно написать $a++ или ++$a. Если следует увеличить значение больше, чем на единицу, можно написать $a++несколько раз, но это не очень удобно и эффективно. Намного больше распространено написание $a = $a + 3. $a + 3 вычисляется, как значение $а плюс 3, а затем присваивается переменной $a, в результате чего значение $а увеличивается на 3. В PHP также, как и в ряде других языков типа С, можно записать это короче: $a+=3. Это значит следующее: возьми значение $a, добавь к нему 3 и присвой это обратно $a. Кроме того, что это понятнее, такой тип выражений быстрее исполняется. Значение $a+=3, как и значение обычного присваивания, — это присвоенное значение. Заметьте, что оно не равно 3, а является общим значением $аи 3. Любой бинарный (имеющий 2 операнда) оператор может быть записан таким методом, например: $a-=5 (вычесть *5* из значения $a), $b*=7 (умножить значение $bна 7) и т. д.

Пример 11.8. Совмещенные выражения

```
<?
$a = 8;
$b = 6;
$c = 4;
$d = 10;
$t = 12;
$e = 14;
$a* = 2; echo $a; // умножить и присвоить
$Ь/ = 3; echo $b; // разделить и присвоить
$c+ = 2; echo $c; // сложить и присвоить
$d- = 3; echo $d; // вычесть и присвоить
$t<< = 1; echo $t; // побитовый сдвиг влево на 1 бит
```

```
$e>> = 2; echo $e; // побитовый сдвиг вправо на 2 бита
?>
```

Думаем, не стоит объяснять полученный результат. Далее вы сможете изучить принципы данных выражений более полно на основании примеров. Подобным образом можно комбинировать присваивание с любыми операциями, если результат присваивается одному из операндов. Делать такие подстановки совершенно не обязательно — язык это позволяет, но не требует.

11.6. Выражения условных операторов

Следующие выражения могут показаться вам незнакомыми, если вы не встречались с ними в других языках, — условный оператор с тремя операндами:

```
$first ? $second : $third
```

Если значение первого выражения истинно (не равно 0), то исполняется второе выражение, и это является результатом данного условного выражения. Иначе исполняется третий оператор.

Иногда очень удобно пользоваться конструкцией

```
<? if ($test !=''):?>
```

вместо правильной

```
<? if (isset($test)&&$test != ''): ?>
```

потому что первая запись короче. В первом случае если переменная $testНе будет определена, то ее значение будет пустой строкой. Логично считать, что «никакая» строка — это тоже пустая. Подобная фраза может вызвать предупреждение о синтаксической ошибке, но это бывает редко.

СОВЕТ

Если вам не лень, то пользуйтесь правильными конструкциями и проверяйте переменную перед обращением к ней. А если лень, то считайте неопределенную переменную пустой строки логической ложью.

Приведем пример, который будет охватывать все, изученное в этой главе. Он должен помочь вам лучше понять предварительное и последующее увеличение и другие выражения:

```
<?
function double($i) // функция удваивания переменной
{
return $i*2;
}
```

```
$Ь = $a = 5; // присваиваем значения переменным $a и $b
$c = $a++; // последующее увеличение, присваиваем $c
           // начальное значение $a (5)
$e = $d = ++$b; // предварительное увеличение, присваиваем $d и $e
                // увеличенное значение $b (6)
                // тут и $d, и $e равны 6
$f = double($d++); // присваиваем удвоенное значение $d
//до его увеличения,
// т. е. 2*6 =12, переменной $f
$g = double(++$e); // присваиваем удвоенное значение $e
// после его увеличения,
// т. е. 2*7 = 14, переменной $д
$h = $д += 10; // сначала увеличиваем значение $д на 10,
               // что дает в результате 24, а затем присваиваем
               // это значение переменной $h, что также дает 24
```

Выражения могут быть операторами. Впрочем, не каждое выражения является оператором. В данном примере оператор имеет форму выражение ;, т. е. выражение, за которым следует точка с запятой. В $Ь = $a = 5; $a = 5 — это правильное выражение, но само по себе оно не является оператором. А вот `$b = $a = 5;` является правильным оператором.

11.7. Логические значения выражений

Во многих случаях, в основном в условных операторах и операторах циклов, вы не заинтересованы в конкретных значениях выражений. Вам только нужно знать, являются ли их значения true или `false`. Логические значения вычисляются примерно также, как и в языке Perl. Любое не нулевое целое значение — это true, нуль — это `false`. Обратите внимание на то, что отрицательные значения — это не нуль, и поэтому они считаются равными true. Пустая строка и строка '0' — это `false`; все остальные строки — true. Что касается составных типов (массивы и объекты), то если значение такого типа не содержит элементов, оно считается равным false; иначе, подразумевается true.

Возникает вопрос, где и как используются данные значения выражений. Область применения их весьма обширна. Например, в выражениях условия (результат такого выражения может быть либо true либо `false`).

Пример 11.9. True/false

```
<?
$a = 5;
```

```
$b = 6;
if ($a>$b)
{
echo "Логический результат данного выражения true";
}
else
{
echo "Логический результат данного выражения false";
}
?>
```

В ходе выполнения данной программы вы получите следующий результат на экране: «Логический результат данного выражения false». Более подробно О функциях, в которых используются данные выражения, поговорим позже.

11.8. Счетчик посещений

Если вы внимательно ознакомились с ранее приведенным материалом, вы без труда получите стартовый капитал знаний, и он даст вам возможность самостоятельно в дальнейшем изучать язык PHP. Хотелось бы предложить вам в качестве примера по выражениям код скрипта, который позволит организовать на любой из страниц вашего сайта счетчик посещений. Этот счетчик не будет полнофункциональным, так как имеет достаточно много недостатков, но как пример применения PHP вполне годится. В любом месте вашей странички (но только там, где это нужно) вставьте следующий код:

```
<P>Посетителей странички -
</php
$filename = "counter.dat";
$fp = @fopen($filename, "r");
if ($fp) {
$counter = fgets($fp, 10);
fclose($fp); } else {
$counter = 0;
$counter++;
echo $counter;
$fp = @fopen($filename, "w");
```

```
if($fp) {
$counter = fputs($fp, $counter);
fclose($fp); }
?></P>
```

В том же каталоге, что и ваша страничка, создайте файл counter.dat, поместите его на сервер и с помощью FTP-менеджера измените атрибуты этого файла таким образом, чтобы он был доступен для записи. Обычно нужно установить галочки на всех атрибутах файла. Если вы этого не сделаете, скрипт будет постоянно выдавать ошибку при попытке записи в файл. Кстати, для того чтобы этого не произошло, стоит поставить перед командой записи и открытия файла символ @, он отменит вывод сообщения о возникнувшей ошибке на экран пользователя. Когда атрибуты изменены, обновите вашу страничку на сервере и обратитесь к ней по ее адресу в браузере. Вы увидите, что там, где вы вставили код PHP, появляется строка: «Посетителей странички- » и далее число, соответствующее количеству посещений. И никакого следа кода. Он был обработан на сервере в Интернете, а браузеру просто переведен результат этого исполнения.

Алгоритм этого скрипта очень прост. В первой строке присваивается выбранной переменной имя файла, в котором будет храниться число посещений. Во второй — открывается соединение с этим файлом для чтения. Дальше проверяется успешность соединения с этим файлом для чтения, и если файл существует и он доступен для чтения, из него считывается строка из 10 байтов, которой более чем достаточно для счетчика, и соединение с файлом закрывается. Показания счетчика увеличиваются на единицу и его новое значение выводится на экран. На следующем этапе нужно записать новое значение счетчика, и для этого снова открывается соединение (дескриптор) с файлом, но уже на запись с очисткой содержимого файла. Если оно успешно, туда записывается новое значение счетчика и дескриптор файла закрывается.

Заключение

Данная глава позволила вам овладеть основными аспектами выражений и понять принципы их построения. Необходимо запомнить следующее. Примеры выражений функций также необходимо иметь в виду при именовании функции, что само имя функции в выражении должно быть предельно кратким и понятным, в противном случае следует использовать комментарии. Скалярные выражения нельзя «разбить» на более маленькие части, как, к примеру, массивы. PHP поддерживает два составных (нескалярных) типа: массивы и объекты. Использование этих типов позволяет манипулировать переменными. Все строки в программе, которые используют переменную, являются выражением. Далее вы будете сталкиваться с выражениями практически в каждом примере.

Глава 12
Операции

Прежде чем перейти к подробному изучению новых для вас конструкций, а также их реализаций, опишем операции, используемые в PHP.

Такие операции, как + (сложение), — (вычитание), *(умножение) и / (деление), знакомы вам по математическим формулам еще со школьных времен. Посредством операций выражаются отношения между данными. Существует множество операций, которые вам знакомы, еще больше могут быть неизвестны, а есть немало таких, которые вы можете вообще не считать операциями.

ВНИМАНИЕ

Настоятельно рекомендуем: если вы не уверены в правильности применения операций (имеется в виду последовательности выполнения), уточняйте их расстановкой скобок.

СОВЕТ

В этой главе приведен полный список операций PHP. По конкретным вопросам назначения и использования отдельных операций обращайтесь к документации по PHP (www.php.net).

Поняв сам принцип употребления операций, вы сможете без проблем перейти к дальнейшему изучению языка PHP. В данной главе будет уделено внимание следующим темам:

- приоритет операций;
- одноместные операции;
- двухместные операции;
- арифметические операции;
- операции назначения;
- поразрядные операции;
- операции сравнения;
- операции контроля ошибок;
- логические операции;
- строковые операции.

12.1. Приоритет операций

Приоритет определяет, к какому операнду относится операция и порядок вычисления операций. Некоторые операции в разных контекстах имеют различный смысл, например операция () (скобки) может обозначать как вызов функции, так и приведение типа. Конкретный контекст определяет, в частности, приоритет операций.

Некоторые операции вам хорошо знакомы и вычисляются в привычном порядке. Например, во фрагменте:

```
$b+$c*$d
```

согласно правилам вычисления арифметических выражений, сначала производится умножение $c*$d, а потом прибавляется $b:

```
<?
$a=3;
$b=2;
$c=4;
$a+$b*$c;
?>
```

результат будет 11, но ни в коем случае не 20, так как оператор «*» имеет более высокий приоритет, чем оператор «+».

Если порядок вычисления последовательности операций не вполне понятен, его можно прояснить расстановкой скобок. Например, предыдущий фрагмент примет вид:

```
($b+($c*$d))
```

Здесь порядок операций однозначно ясен, хотя для компилятора обе записи совершенно равноценны. Что касается скоростей обработки этих двух выражений, то они приблизительно равны.

СОВЕТ

Операции - это фундамент выражений, поэтому с теми, которые вам незнакомы, следует экспериментировать.

В табл. 12.1 приведен список операций, применяемых в PHP. В ней операции приведены в порядке убывания их приоритета, т. е. элементы, имеющие более высокий приоритет, выполняются всегда первыми, имеющие более низкий приоритет — вторыми и т. д.

Таблица 12.1. Приоритет операций

Операции	Последовательность выполнения
New	—
[	Слева направо
! ~ ++ — (int) (double) (string) (array) (object) @	Слева направо
* / %	Слева направо
+ — .	Слева направо
<< >>	Слева направо
< <= > >=	—
== != === !==	—
&	Слева направо
^	Слева направо
|	Слева направо
&&	Слева направо
||	Слева направо
? :	Слева направо
= += -= *= /= .= %= &= |= ^= ~= <<= >>=	Слева направо
print	Слева направо
AND	Слева направо
XOR	Слева направо
OR	Слева направо
,	Слева направо

12.2. Одноместные операции

Термин «одноместный», или «унарный», явно подразумевает, что число операндов (переменных), над которыми операция производит свое действие, равно единице.

Первая популярная одноместная операция, которая используется для проверки условий, — это операция ! (не, not). Она производит отрицание операнда. Отрицание значения true есть false, и наоборот, не false — есть true. Если переменная имеет нулевое значение указателя, то результат false, если же не нулевое — true.

Общая форма записи одноместной операции выглядит так:

```
одноместная_операция операнд
```

Применим эту форму записи к операции !:

```
$a = 5;
if(!$a) // проверка на истинность
```

Вы видите, что операцию ! можно использовать в условных выражениях. Также в них можно использовать логические (булевы) значения, целые числа и различные выражения. То, что принимает нулевое значение, интерпретируется как ложное условие, а все, что не нуль, считается истиной. При подстановке вместо $a его логического эквивалента «ложь», вышеприведенное выражение примет вид: «если (не ложь)»; не ложно есть истинно.

```
<?
$p=5;
if(!$p) // условие ложно
?>
```

На протяжении этой книги вы постоянно будете сталкиваться с унарными операциями. Так, например, очень часто встречающейся их разновидностью являются префиксный и постфиксный инкременты и декременты (т. е. ++ и --) целочисленных значений. Например:

```
$number = 5;
$number++; // то же, что $number = $number +1 или $number += 1
--$number; // то же, что $number = $number -1 или $number -= 1
```

Префиксные операции располагаются слева от операнда и производят действие над ним до того, как его значение будет использовано, а постфиксные операции выполняются после всех вычислений.

Одноместная операция отрицания обозначается тильдой (~) и производит побитовое отрицание (или дополнение) операнда. Дополнение к –1:

```
$a = -1;
$a = ~$a;
дает в результате 0 .
```

Операции +и -имеют как одноместную, так и двухместную форму. По умолчанию число считается положительным, поэтому указание плюса (+4) является избыточным, если не определено обратное (–4).

12.3. Двухместные операции

Термин «двухместные», или «бинарные» (binary), означает, что операции этого рода производят свои действия над двумя операндами. Одни из наиболее часто употребляемых двухместных операций — это +, -, / и *. Их названия и значения пришли неизменными из вычислительной математики.

В общем виде двухместные операции записываются таким образом:

```
операнд двухместная_операция операнд
```

При использовании операций в более сложных сочетаниях очень важны правила ассоциативности, дистрибутивности, транзистивности и симметрии.

ВНИМАНИЕ

Многие операции имеют не одно значение. Как было сказано выше, в этом случае значения операций выясняются из контекста.

В табл. 12.2 приведены операции с описанием их обозначений и синтаксиса, сгруппированные по убыванию приоритета.

Таблица 12.2. Двухместные операции

Обозначение	Название	Синтаксис
[]	Элемент массива	указатель [выражение]
()	Вызов функции	выражение (аргументы)
()	Задание значения	тип (аргументы)
++	Постфиксный инкремент	значение++
++	Префиксный инкремент	++значение
--	Постфиксный декремент	значение--
--	Префиксный декремент	--значение
~	Дополнение	~выражение
!	Отрицание	!выражение
-	Одноместный минус	-выражение
+	Одноместный плюс	+выражение
&	Получение адреса	&значение
new	Создать (разместить)	new тип
new[]	Создать массив	new тип[]
()	Приведение типа	(тип) выражение
*	Умножение	выражение * выражение
/	Деление	выражение / выражение
%	Модуль (остаток деления)	выражение % выражение

Окончание табл. 12.2

Обозначение	Название	Синтаксис
+	Сложение	выражение + выражение
-	Вычитание	выражение - выражение
<<	Сдвиг влево	выражение << выражение
>>	Сдвиг вправо	выражение >> выражение
<	Меньше, чем	выражение < выражение
<=	Меньше, чем или равно	выражение <= выражение
>	Больше, чем	выражение > выражение
>=	Больше, чем или равно	выражение >= выражение
==	Равно	выражение = = выражение
!=	Неравно	выражение != выражение
&	Побитовое И	выражение & выражение
^	Побитовое исключающее ИЛИ	выражение ^ выражение
|	Побитовое ИЛИ	выражение I выражение
&&	Логическое И	выражение && выражение
||	Логическое ИЛИ	выражение I I выражение
?:	Условное выражение	выражение ? выражение : выражение
=	Простое присваивание	значение = выражение
*=	Умножить и присвоить	значение *= выражение
/=	Разделить и присвоить	значение /= выражение
%=	Взять остаток и присвоить	значение %= выражение
+=	Сложить и присвоить	значение += выражение
-=	Вычесть и присвоить	значение -= выражение
<<=	Сдвиг влево и присвоить	значение <<= выражение
>>=	Сдвиг вправо и присвоить	значение >> = выражение
&=	И и присвоить	значение &= выражение
|=	ИЛИ и присвоить	значение I= выражение
^=	Исключающее ИЛИ и присвоить	значение ^= выражение
,	Запятая (последование)	выражение, выражение

Создавать выражения, используя двухместные операции, сравнительно просто Определив целые переменные:

```
$a = 5;
$b = 3;
```

можно написать выражения с использованием сложения, вычитания, умножения и деления:

```
$c = $a + $b;   // присвоить $c сумму $a и $b
$c - $a — $b;   // присвоить $c разность $a и $b
$c = $b % $a;   // присвоить $c остаток от деления $b на $a
$c = $b * $a;   // присвоить $c произведение $a и $Ь
```

Наверняка у вас есть практика в написании выражений с арифметическими операциями, а если вы изучали логику или дискретную математику, то вам приходилось иметь дело и с чуть более экзотическими логическими операциями.

Ваших знаний об операциях и опыта достаточно для написания наиболее простых выражений. Двухместные операции вместе с операциями присваивания (знак равенства) можно использовать в сочетании. Рассмотрим:

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

В этом выражении $a увеличивается на 5. Поскольку в нем присутствует только $a и 5, можно заменить его на

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

Подобным образом можно комбинировать присваивание с любыми двухместными операциями, если результат присваивается одному из операндов. Делать такие подстановки совершенно не обязательно — язык это позволяет, но не требует. Имея в виду все вышесказанное, вы, по всей вероятности, уже овладели искусством писать выражения, используя более чем половину указанных в таблице операций.

С таким багажом знаний вы уже можете реализовать многие из наиболее популярных алгоритмов. Поставьте в таблице галочки напротив тех операций, с которыми вы знаете, как обращаться. Для оставшихся вам желательно написать несколько программ и поэкспериментировать. Однако искусство владения всеми типами операций — это только полдела.

12.4. Арифметические операции

Арифметические операции вам должны быть знакомы больше всех. Покажем применение данного типа операций (табл. 12.3).

Таблица 12.3. Арифметические операции

Пример	Название	Результат	На практике	Результат
$a + $b	Сложение	Сумма $a и $b	5+2	7
$a - $b	Вычитание	Вычитает $b из $a	5-2	3
$a * $b	Умножение	Произведение $a и $b	5*2	10
$a / $b	Деление	Деление $a на $b	5/2	2.5
$a % $b	Остаток от деления	Остаток от деления $a на $b	5 % 2	1

12.5. Операции назначения

Основным оператором назначения является =. Можно подумать, что это «равно», но это не так. В действительности это означает, что левый операнд получает значение выражения в правой его части (собирательное присваивание).

Значением выражения назначения является присваиваемая величина. Например, если $a = 3, то значением выражения назначения является 3. Приведем еще пример:

```
- $a = ($b = 4) + 5; // теперь $a равно 9, а $b стало равным 4
```

В дополнение к основным операторам присваивания есть «комбинационные операторы» для всех арифметических и строковых операторов, что позволяет использовать значение в выражении и затем устанавливать свое значение в результате этого выражения. Например:

```
$a - 3;
$a+ = 5; // теперь $a равно 8, то есть: $a = $a + 5;
$b = "Hello ";
$b .= "There!"; // теперь $b равно "Hello There!", как если бы
мы написали $b = $b . "There!";
```

Обратите внимание, что если две переменные имеют одинаковое значение, то изменение одной из них не вызовет изменения другой. Это очень уместно, если вам необходимо сохранить идентичное значение большого массива внутри рабочего цикла. PHP 4 поддерживает назначение ссылки. Например:

```
$var = &$othervar;
```

но это невозможно в PHP 3.

Назначение при помощи ссылки обозначает, что обе переменные указывают на то же самое значение переменной. Более подробно вы ознакомитесь с ссылками немного позже (§ 13.21).

12.6. Поразрядные операции

Бинарные операторы позволяют изменять биты в целых числах (табл. 12.4).

Таблица 12.4. Поразрядные операции

Пример	Название	Результат	Пример использования
$a & $b	И	Будут установлены биты, которые были установлены и в $a, и в $b	`$a=5; /* 0101 */ $b=12; /* 1100 */ $c=$a & $b; /*$c будет равно 4 (0100) */`
$a | $b	ИЛИ	Будут установлены биты, установленные в $a или $b	`$a=5; /* 0101 */ $b=12; /* 1100 */ $c=$a|$b /* $c будет равно (1101) */`
$a	НЕ	Будут установлены биты, не присутствующие в $a	`$a=5; /* 0101 */ ~ $a /* $a будет равно x(1010) */`

12.7. Операции сравнения

Операции сравнения позволяют сравнивать различные выражения, значения переменных и т. д. Принцип реализации данных операций весьма прост (табл. 12.5).

Таблица 12.5. Операторы сравнения

Пример	Название	Результат
$a == $b	Равно	Истина, если $a эквивалентно $b
$a != $b	Не равно	Истина, если $a не эквивалентно $b
$a < $b	Меньше, чем	Истина, если $a меньше $b
$a > $b	Больше, чем	Истина, если $a больше $b
$a <= $b	Меньше или равно	Истина, если $a меньше или равно $b
$a >= $b	Больше или равно	Истина, если $a больше или равно $b

12.8. Операции контроля ошибок

PHP поддерживает один оператор контроля ошибок — знак @, который употребляется перед выражением. При этом любое сообщение об ошибке, которое могло бы сгенерироваться этим выражением, будет проигнорировано, т. е. будет продолжено выполнение программы, только данное выражение не будет принимать в нем участия.

Если параметр track_errors установлен как enabled, все сообщения об ошибках, появившиеся в результате действия данного выражения, будут сохранены в глобальной переменной $php_errormsg. Эта переменная будет изменяться при появлении каждой новой ошибки, поэтому прежде чем воспользоваться ею, вам необходимо проверить ее заранее. Например:

```
<?php
/* намеренная ошибка SQL (дополнительная кавычка) */
$res = @mysql_query ("select name, code from 'namelist") or
die ("Query failed: error was '$php_errormsg'");
?>
```

Результат выполнения программы представлен на рис. 12.1.

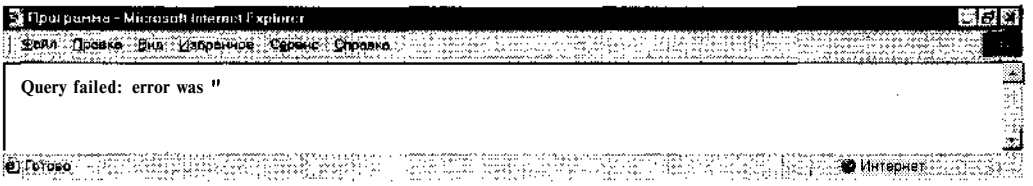


Рис. 12.1. Результат выполнения скрипта

ВНИМАНИЕ

Если вы используете @, чтобы подавить ошибки от некоторой функции, в таком случае сам скрипт прекратит свое выполнение без какого-либо объяснения, почему он это сделал.

12.9. Логические операции

PHP поддерживает следующие логические операции (табл. 12.6).

Таблица 12.6. Логические операции

Пример	Название	Результат
$a and $b	И	Истина, если истинны $a и $b
$a or $b	ИЛИ	Истина, если истинны $a или $b
$a xor $b	ИЛИ	Истина, если истинны $a или $ь, но не оба
! Sa	НЕ	Истина, если не истинно $a
$a && $b	И	Истина, если истинны и $a и $Ь
$a || $b	ИЛИ	Истина, если истинны $a или $b

Приоритет операций был описан выше.

12.10. Строковые операции

Существует только один оператор строк — оператор конкатенации (.). Покажем на примере:

```
$a = "Hello ";
$b = $a . "World!"; // теперь $b = "Hello World!"
```

Заключение

В этой главе вы получили представление об операциях и встроенных типах данных. Они выглядят именно так, как мы привыкли при вычислении на бумаге или с помощью калькулятора.

Из этой главы вы также узнали о средствах языка программирования PHP. В следующих главах вы изучите все этапы создания классов. А пока пусть вас утешает мысль, что хотя самые передовые стороны PHP для вас пока еще скрыты завесой таинственности, но они ждут своей очереди и скоро вы будете готовы встретиться с ними лицом к лицу. Побольше узнать о выразительной стороне операций и типов данных вы можете в следующих главах.

Глава 13
Структуры управления данными

В предыдущих главах мы рассмотрели числовые переменные и константы, познакомились с набором операций, с помощью которых написали небольшие программы. Эти программы имели линейную структуру (т. е. каждый оператор в них выполнялся один раз) и ориентировались в основном на арифметические вычисления. В данных программах отсутствовали какие-либо логические условия, касающиеся обработки отдельных переменных или отдельных операций. При каждом задании такой программы она выполнялась целиком, т. е. работали все ее операторы.

В этой главе обсуждается следующий, более высокий уровень, на котором формируются операторы. Тут будут рассмотрены управляющие конструкции языка. В PHP к ним относятся условные конструкции (ветвления) и циклические конструкции (циклы). Под ветвлением понимается ситуация, когда на основании проверки некоторых условий в программе реализуется один из несколько заранее предопределенных путей решения задачи. Используя циклические конструкции, программист может организовать многократное выполнение некоторой последовательности команд (фрагмента программы) без повторного описания этого фрагмента в программе. Прежде чем перейти к рассмотрению управляющих структур языка PHP, необходимо определить такое базовое понятие, как оператор.

Операторы являются наименьшими исполняемыми единицами программы. Они управляют процессом выполнения программы. Операторы в законченном блоке кода выполняются по порядку, каждый следующий только после окончания работы предыдущего, за исключением случаев, когда управляющий оператор заставляет переходить программу в другое место. Операторы можно разделить на четыре категории:

- последовательные операторы;
- операторы выбора;
- операторы цикла;
- операторы перехода.

Все они будут рассмотрены в следующих параграфах. Более подробное описание того, что вы сможете изучить в этой главе:

- последовательные операторы;
- операторы объявления;
- операторы выражения;
- пустые операторы;
- составные операторы;
- операторы выбора;
- конструкция if;
- конструкция if ... else;
- конструкция elseif;
- альтернативный синтаксис для управляющих структур;
- конструкция switch;
- операторы цикла;
- конструкция while;
- конструкция do...while;
- конструкция for;
- конструкция foreach;
- операторы перехода;
- оператор break;
- оператор continue;
- оператор возврата return;
- включение исходного кода текста, содержащегося в файле;
- оператор require();
- оператор include();
- оператор require_once();
- оператор include_once().

13.1. Последовательные операторы

Последовательными операторами называются такие операторы, которые не управляют непосредственно выполнением других операторов. Большинство операторов в программах являются последовательными. Всего существует четыре типа последовательных операторов:

- операторы объявления;
- операторы выражения;
- пустые операторы;
- составные операторы.

13.2. Операторы объявления

Операторы объявления — единственные операторы, которые могут располагаться вне функции. Они состоят из объявления переменной или функции и заканчиваются точкой с запятой (;). Объявление можно сочетать с присваиванием, как в следующем примере:

```
$a = 5;
$b = 2.6;
```

13.3. Операторы выражения

Операторы выражения — это выражения, за которыми следует точка с запятой. Это обычный вид операторов. Например:

```
$a++;
$a + $b;
--$c;
$a = goodway($y) + $s;
```

Заметьте, что строка $a + $b; хоть и будет компилироваться без ошибок, реального смысла не имеет.

13.4. Пустые операторы

Пустой оператор является одним из самых важных операторов:

```
;
```

Он состоит только из точки с запятой. Хотя такой оператор может показаться бесполезным, он удобен в тех случаях, когда присутствие оператора синтаксически необходимо, но совершенно неуместно. Иногда такое встречается в операторах while и for:

```
while (test_function($x));
```

13.5. Составные операторы

Составной оператор, или блок (block), представляет из себя несколько операторов, заключенных в фигурные скобки ({}). Составные операторы могут появляться везде, где допустимы простые операторы. Они позволяют выполнять последовательность операторов там, где по синтаксису подразумевается только один оператор. Например:

```
<?
$a = 2;
If ($a >= 0)
{
$d = 2;
$f = 3;
$n = 2;
$c = $a + $d;
$n += $c/$f;
}
$c++;
echo $c;
?>
```

При описании данного примера выделим кое-какие отличия языка программирования PHP от С. Дело в том, что в С переменные, объявленные внутри составных операторов, могут быть использованы только внутри блока. Таким образом, переменные в предыдущем примере вне скобок не определены. Это принцип работы языка С. Что касается PHP, то каких-либо строгих ограничений в этом направлении не существуют. Этот пример позволяет вам убедиться в том, что переменные, объявленные внутри составных операторов, могут быть использованы не только внутри блока, но также и за его пределами.

Тела функций также заключаются в скобки, следовательно, тело функции является составным оператором. Пустое тело функции — это один из немногих случаев, в которых имеет смысл пустой блок (т. е. составной пустой оператор). При разработке программ можно создавать заготовки функций — функции с пустыми телами — и заполнять их по ходу дела.

Точка с запятой после закрывающей скобки не обязательна, однако хуже не будет, если ее поставить.

13.6. Операторы выбора

Часто определенная часть программы может выполняться только при соблюдении некоторых условий. В качестве примера можно привести программу социологического опроса (заполнения анкеты), в которой задается вопрос о классе транспортных средств, которыми респондент имеет право управлять. Однако сначала необходимо спросить, имеются ли вообще у анкетируемого водительские права. Реализовать данную программу будет проще всего, применив именно операторы выбора.

Операторы выбора выполняют другие операторы в зависимости от условных значений. Это позволяет управлять программой на основе определенных критериев. Существует два типа таких операторов:

- оператор `if`;
- оператор switch.

13.7. Конструкция `if`

Оператор if проверяет выражение. Если значение выражения ненулевое, оно подчиняется значению true, в этом случае выполняется следующий за ним оператор; если же оно равно нулю, т. е. false, то следующий оператор не выполняется. Например:

```
If ($s)
$s++;
```

Скобки вокруг выражения $s являются обязательными. Выражение в скобках может быть совершенно произвольным, но с одним условием: оно должно возвращать скалярную величину, которую можно сравнивать с нулем.

ВНИМАНИЕ

Суть определения условия состоит в сравнении двух величин, поэтому в описании условий должен стоять простой символ равенства, соответствующий обозначению. Не путайте операцию присвоения значения (=) с операцией сравнения (==).

Возникает вопрос: когда же величина может быть равна нулю, т. е. в каком случае она считается строго равной нулю: `bool` (false), integer (0), floating point numbers (0 . 0), double (`0.0`).

Значения `float` и double бывает довольно сложно проверить на равенство нулю: могут сказываться ошибки округления в предшествующих вычислениях. Обычно величины этих типов не рекомендуется использовать в проверках на равенство. Тем не менее случается так, что надо подвергнуть проверке именно величины типа `floating` point или double. Допустим, что значение типа `float` (в данном слу-

чае — x) необходимо проверить на равенство (или приблизительное равенство) некой величине n. Также допустим, что требуемая точность сравнения лежит в пределах 0.00000001. Необходимо установить, находится ли проверяемая величина в пределах заданной точности от п:

```
$род = 0.00000001;
if ( ($x > $п - $род) && ($x < $п +$pog) )
echo "x достаточно точно равно", $n;
```

Часто проверяемое в операторе if выражение является логическим выражением или сравнением, например, в рассмотренном случае присутствует их комбинация.

В общем случае структура оператора if выглядит следующим образом:

```
If ( условие )
{
Оператор_1;
Оператор_2;
...
Оператор_последний;
}
```

Как уже говорилось ранее, составной оператор может быть помещен в любое место программы, в котором может находиться одиночный оператор. Стало быть, выражение, следующее после if, может являться составным оператором. Например:

```
If ( ($x > 5) && !$d)
{
$z = 5*$s;
$r+ = 45;
}
```

Часто по условию требуется исполнить более одного выражения. Конечно, не надо окружать каждое выражение конструкцией if. Вместо этого вы можете сгруппировать несколько выражений в блок выражений, как было показано в предыдущем примере. Не забывайте использовать фигурные скобки, иначе у вас получится:

```
If ( ($x > 5) && !$d)
$z = 5*$s;
$r+ = 45;
```

На первый взгляд, оба оператора после if при истинности условия выполняться. Но компилятор не станет смотреть на то, как вы расставили отступы: оператор

```
$r+ = 45;
```

выполнится даже в том случае, если значение выражения ложно. Другими словами можно сказать, что если после if не используются фигурные скобки и количество операторов больше, чем два, то весь принцип работы программы будет заключаться в следующем. Если условие истинно, то будет выполняться первый оператор, который идет после конструкции условия, и все остальные, следующие после этого оператора. В противном случае — ложно именно один первый оператор не выполнится, а все остальные, следующие после него, будут выполняться в заданной последовательности. Проиллюстрируем это на примере:

```
<?
$a = 3;
if ($a>5)
$b = $a + 2;
$b = $a + 1;
echo $b;
?>
```

В ходе выполнения этого примера переменная $b получит значение 4 и будет выведена. В свою очередь строка

```
$b=$a+2;
```

будет пропущена в ходе выполнения программы, так как значение выражения в операторе if будет false (ложно). Если бы переменная $a имела значение больше 5, конструкция if позволила бы выполнить в первую очередь выражение

```
$b = $a + 2;
```

а затем

```
$b = $a + 1;
```

СОВЕТ

Некоторые программисты в операторах цикла и в операторах выбора всегда используют составные операторы, даже если оператор в нем всего один. Если впоследствии понадобится вставить в блок еще один оператор, то можно будет не опасаться ошибок, вызванных отсутствием скобок. Подобная практика, как и расположение скобок, - дело вкуса. Вырабатывайте свой стиль и придерживайтесь его.

Выражение if может иметь неограниченную степень вложенности в другие выражения *if*, что позволяет эффективно использовать выполнение по условию различных частей программы. Например:

```
<?
$a = 5;
```

```
$b = 2;
$c = 1;
if ($a>3)
{
if ($b == 2)
{
if ((($c == 1) && ($b ==2))
{
$c = $b+ 5;
$s = 0;
}
$c++;
}
$c = $c*2;
}
echo $c;
?>
```

Результатом выполнения данной программы будет изменение значения переменной $cc $c = 1 на $c = 16;.

Обратите внимание, что если вы поставите точку с запятой непосредственно после самого оператора if:

```
if ($a >= 3);
```

то это не вызовет никакой ошибки при выполнении программы, как и при работе с каким-либо другим языком программирования. Но правила хорошего тона среди программистов состоят именно в том, чтобы создать как можно более понятный и документируемый код программы, а бесполезное использование точки с запятой после оператора if характеризует ваше отношение к создаваемой программе. Поэтому постарайтесь не допускать таких ошибок.

13.8. Конструкция if ... else

Оператору if может сопутствовать еще один оператор — else. Это выглядит так:

```
if ($s)
++$s;
```

```
else
echo "s = O";
```

Если выражение $s в операторе ifистинно (true), то выполняется первый оператор ++$s;. Если выражение $sложно (false), выполняется второй оператор: echo "s = 0";. Это позволяет в зависимости от значения выражения в условии исполнять одну из двух взаимоисключающих ветвей. Для увеличения количества возможных вариантов выбора пары if... elseможно объединить. Например, создадим форму, в которую будем вводить значение переменной $name — какое-нибудь имя. После выполнения скрипта на основе данных, введенных в форму, переменная будет перемещена в файл (например, test .php) для обработки. Там на основании значения этой переменной будет выполняться та или иная функция:

```
Файл test.php
<?
if ($name = = "Саша")
OneFunct();
else if ($name == "Таня")
OneFunct1();
else if ($name == "Миша")
OneFunct2();
else // по умолчанию использовать не обязательно
echo "ошибка ввода, такое имя не найдено";
```

Каждое условие будет проверяться в свою очередь. Оператор, следующий за первым if, условие которого истинно, будет исполнен. Все нижеследующие операторы ifбудут проигнорированы, не смотря на значения их условий. Если ни одно из проверяемых выражений не возвратит true, то выполнится оператор по умолчанию — следующий за else.

Общую форму оператора конструкции if, содержащего ветвь else, можно записать следующим образом:

```
if (условие)
{
Оператор_ветви_Если_То_1;
Оператор_ветви_Если_То_2;
...
Оператор_ветви_Если_То_последний;
}
```

```
else
{
Оператор_ветви_Если_Иначе_1;
Оператор_ветви_Если_Иначе_2;
...
Оператор_ветви_Если_Иначе_последний;
}
```

Если в ветви if или else находится только один оператор, то операторные скобки ({}) можно не открывать:

```
if (условие)
Единственный_оператор_ветви_Если_То
else
Единственный_оператор_ветви_Если_Иначе;
```

В последнем случае конструкция if ... else выглядит следующим образом (рис. 13.1).

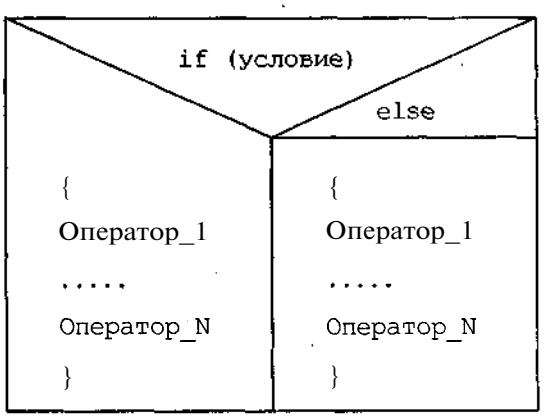


Рис. 13.1. Конструкция if...else

Допускается любое количество операторов, вложенных в if. Следите за тем, чтобы все операторы else соответствовали своим if. Приведем пример того, как не надо делать. Создадим два файла: один — index.html, другой — test.php. Пользователи будут вводить имя в указанную форму, после этого скрипт будет обрабатывать данную форму и выводить, есть такой пользователь в базе данных или нет. Количество пользователей сократим до двух, чтобы понятнее разъяснить саму суть выполнения программы:

```
<html>
<head>
<title>Программа</title>
</head>
<body>
<form action="test.php" method="POST">
```

```
Name: <input type="text" name="name">
<input type="submit">
</form>
</form>
</body>
</html>
```

Этот файл показывает содержимое формы, через которую задаются значения переменной `$name`. После того как значение будет задано, переходим к файлу `test.php`:

```
<?
if ($name != "Саша")
if ($name == "Таня")
echo "Вам придется зарегистрироваться!";
else
echo "Вы уже зарегистрированы у нас!";
?>
```

После того как будет введено значение переменной, обработка ее будет происходить в этом файле. Но в связи с тем что написание самого скрипта неверное, будет выдаваться неверное значение, т. е.: пользователь по имени Саша уже зарегистрирован, а пользователь по имени Таня — нет. Проанализируем программу. Если переменная приняла значение "Саша", то в первом условии false и в результате на экран браузера ничего выведено не будет, хотя по расположению оператора else кажется, что он относится именно к первому оператору `if`. Если значение переменной станет "Таня", то первое и второе условия станут true, и в результате выполниться строка:

```
echo "Вам придется зарегистрироваться!";
```

Все так и должно быть, но вот если значение нашей переменной станет, например, равной "Игорь", тут возникнут большие проблемы. Второе выражение в операторе условия примет значение false, и программа перейдет к строке:

```
echo "Вы уже зарегистрированы у нас!";
```

Хотя пользователь под этим именем не зарегистрирован. Дело в том, что PHP ставит оператор else (или else if) в соответствие последнему непарному оператору `if`. Поэтому последний оператор else относится ко второму оператору `if`, несмотря на то что, судя по отступам, имелось в виду совершенно другое. В такой ситуации выручают скобки:

```
<?
if ($name != "Саша")
```

```
{
if ($name == "Таня")
echo "Вам придется зарегистрироваться!";
}
else
{
echo "Вы уже зарегистрированы у нас!";
}
?>
```

13.9. Конструкция elseif

В PHP использование оператора условия очень популярно, и на основании этого было сделано много разнообразных модификаций по сравнению с C.

Оператор elseif, как и следует из его названия, является комбинацией if и else. Можно написать else if (два слова), что будет значить то же самое, что и elseif (одно слово). Elseif, как и else, позволяет выполнить выражение, если значение if равно false, но в отличие от else оно выполнится, только если выражение elseif равно true. Например, следующий код выведет «a is больше than b» если $a>$b, «a равно b» если $a==$b, и «a меньше b» если$a<$b:

```
if ($a > $b)
{
print "a больше b";
}
elseif ($a == $b)
{
print "a равно b";
}
else
{
print "a меньше b";
}
```

Внутри одного выражения if может быть несколько elseif. Первое выражение elseif, которое примет значение true, будет выполнено.

Выражение elseif будет выполнено, только если выражение if и все предыдущие elseif равны false, а данный elseif равен true.

13.10. Альтернативный синтаксис для управляющих структур

PHP позволяет использовать альтернативный синтаксис для управляющих структур. Основные структуры, для которых применяется этот синтаксис, следующие: if, while, for, foreach и switch. Если существует необходимость проверить ту или иную переменную и на основании этого вывести код, то именно в этом случае применяется альтернативный синтаксис:

```
<?php if ($a == 5): ?>
A равно 5
<?php endif; ?>
```

В данном примере используется структура if. Аналогичным образом могут быть описаны все вышеперечисленные операторы. Завершающим оператором для них будут соответственно следующие операторы: endif, endwhile, endfor, endforeach И endswitch.

Рассмотрим пример. Если значение переменной действительно будет равно 5, браузер помимо всего документируемого кода HTML выведет и строку «а равно 5».

Этот альтернативный синтаксис применим и к операторам else и elseif(). Вот пример подобной структуры:

```
if ($a == 5):
print "a равно 5";
print "...";
elseif ($a == 6) :
print "a эквивалентно 6";
print "!!!";
else:
print "a около 5 или 6";
endif;
```

13.11. Конструкция switch

Оператор switch позволяет задавать несколько вариантов действий практически так же, как и оператор elseif. На самом деле, используя elseif, можно написать эквивалент любого оператора switch, но иногда switch гораздо понятнее. Допустим, у вас есть следующий участок в программе:

```
if ($i == 0)
{
print "i равно 0";
}
else
if ($i == 1)
{
print "i равно 1";
}
else
if ($i == 2)
{
print "i равно 2";
}
else
print "ни одно условие не выполнилось";
```

Сэкономить время выполнения данной части, а также представить ее более логичным способом и поможет этот оператор.

Следующий пример совершает действия, аналогичные предыдущему, но в более красивой форме:

```
switch ($i)
{
case 0:
print "i равно 0";
break;
case 1:
print "i равно 1";
break;
case 2:
print "i равно 2";
break;
```

```
default: // необязателен
print " ни одно условие не выполнилось"
}
```

В таком представлении есть и еще одно преимущество. Если вы не поставите оператор break, например, перед case `1:`, то в случае, когда переменная $i будет равна нулю, после вывода на экран сообщения об этом программа пойдет дальше и выведет также сообщение о том, что переменная `$i` равна еще и 1, и только после, встретив break;, продолжит свое выполнение за пределами switch.

Опишем принцип работы данного оператора. Значение выражения в скобках после оператора switch в первой строке сравнивается со всеми значениями case. Оператор case — это метки. Каждая метка case должна быть целочисленной константой или приводимой к ней. Дополнительное требование к меткам состоит в том, что значение каждой из них должно быть уникальным. Значение выражения в операторе switch поочередно сравниваются с каждым значением case. Когда метка, равная значению выражения, найдена, выполняются все операторы после нее и до ближайшего оператора break или до конца оператора switch. Если значение переменной будет соответствовать 1, то вызывается функция print `"i равно 1";`. Если бы после print "i равно `1";` не было оператора `break`, то была бы также вызвана функция print "i равно `2";`, даже если значение переменной не было бы равно 2. Оператор break передает управление оператору, следующему за закрывающей скобкой оператора switch.

Иногда такое действие может оказаться полезным. Например, от пользователя требуется ввести a, `b` или `c` в нижнем регистре. Если регистр символа нам не важен, то надо разрешить ввод в любом регистре:

```
switch ($i) {
case 'A' :
case 'a' :
print "i равно A или a";
break;
case 'B':
case 'b';
print "i равно B или и";
break;
case 'C':
case 'c':
print "i равно C или c";
break;
```

```
default:
print "Ошибка ввода !";
}
```

Функция print "i равно A или a"; будет вызвана, когда переменная $i примет значение a или A. Метка default аналогична оператору else и так же, как и else, необязательна.

Изобразим структурограмму конструкции switch (рис. 13.2).

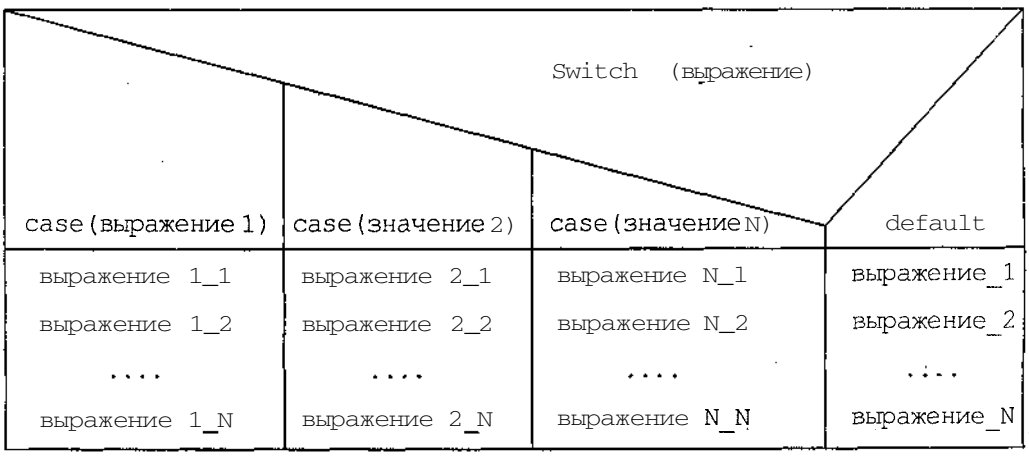


Рис. 13.2. Конструкция switch

Чтобы описать принцип работы данного скрипта, рассмотрим программу. Возьмем два файла: index.htm и test.php.

index.htm:

```
<html>
<head>
<title>Программа</title>
</head>
<body>
<form action="test.php" method="POST">
Введите ваше имя <input type="text" name="name"> <br>
<input type="submit">
</form>
</body>
</html>
```

Данный файл позволяет создать форму, через которую будем вводить значения нашей переменной. Далее эта форма вызывает выполнение программы, которая содержится в файле test .php:

```
<?
switch ($name) {
case 'Саша':
print "Пользователь под именем $name уже зарегистрирован";
break;
case 'Таня':
print "Пользователю под именем $name необходимо перерегистрироваться";
break;
case 'Игорь':
print "Пользователь по имени $name удален из нашей базы за несоблюдение правил работы";
break;
default:
print "Пользователь по имени $name у нас не зарегистрирован, пройдите, пожалуйста, регистрацию" ;
}
?>
```

Принцип выполнения программы заключается в следующем. В зависимости от значения нашей переменной будет выполняться тот или иной блок. Этот пример показывает, как можно работать с оператором switch. После того как вы изучите основные функции, вы сможет применять их в этих блоках.

13.12. Операторы цикла

Циклы в программировании — это повторяющиеся несколько раз операции. Начало (точка отсчета) указывается в начале цикла, а продолжительность его выполнения ограничивается каким-либо условием.

Операторы цикла (iteration statements) повторяют заданную последовательность операторов фиксированное число раз или до тех пор, пока не будет удовлетворено условие проверки. Существует три типа циклических операторов:

- while,
- do,
- for.

13.13. Конструкция while

Цикл while проверяет выражение, и в том случае, если оно равно true, выражение выполняет одиночный или составной оператор:

```
$i = 1;
while ($i <= 10)
{
print $i++;
}
```

Сначала проверяется выражение. Если $i меньше или равно 10, то выполняется оператор print $i++;. Если $i больше 10, то цикл передает управление следующему за ним оператору. Важной характеристикой цикла while является то, что оператор никогда не выполнится, если условие изначально было ложно.

ВНИМАНИЕ

Часто встречается ошибка, когда условие окончания цикла не удовлетворяется. Это может происходить потому, что управляющая переменная не изменяется в теле цикла, в результате цикл становится бесконечным. В таких случаях заметить ошибку довольно легко. При написании цикла убедитесь, что граничное значение достижимо.

Вместе с оператором цикла часто используется пустой оператор. В следующем примере происходит увеличение переменной $пате до значения, большего 10, т. е. в момент, когда значение переменной станет равной 11, цикл прекратит выполнение:

```
while ($name++ < 10)
;
```

Правила проверки контрольного выражения те же самые, что и для выражений в операторе if. Ниже изображена структурограмма цикла while (рис. 13.3).

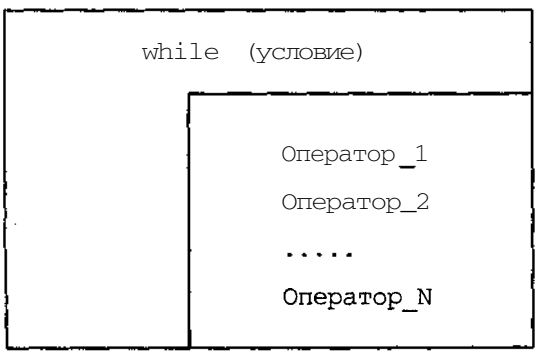


Рис. 13.3. Цикл while

Примером цикла может служить копирование нескольких файлов. Алгоритм выполнения этого задания можно описать так: установить счетчик скопированных файлов на нуль, скопировать файл, проверить, закончились файлы или нет: если нет — увеличить на счетчике количество скопированных файлов и вернуться к началу цикла (опять скопировать файл), если да — закончить цикл.

Рассмотрим элементарную программу сложения чисел от 0 до того значения, которое будет задано пользователем. Для этого необходимо задать файл index.htm, через который будет производиться вызов программы PHP и при этом задаваться значение переменной. Например, имя переменной, которая будет использоваться в программе — $name = 30;. Тогда сам скрипт будет выглядеть следующим образом:

```
<?
$name = 30;
$my =0;
while ($my < $name)
{
$my++;
}
echo $my;
?>
```

Значение переменной $name = 30; в начале программы задано намеренно, чтобы вы могли сразу реализовать данную программу. Реально ее там не будет, а само значение она будет получать из файла index.html. Результатом выполнения данной программы будет число, равное 30.

Иногда в конце конструкции while применяется оператор endwhile.

```
while(условие):
выражения
...
endwhile;
```

Например:

```
$i = 1;
while ($i <= 10):
print $i;
$i++;
endwhile;
```

Этот оператор не обязателен, его отсутствие в программе не вызовет никакой ошибки.

13.14. Конструкция do ... while

DO ... while — это тоже цикл, отличается от while тем, что значение логического выражения проверяется не до, а после окончания работы операторов, включенных в сам цикл. Таким образом, do ... while гарантированно будет выполнен хотя бы один раз, что в случае с while совсем не обязательно. Если условие ложно, управление сразу будет передано дальше. Например:

```
<?
$i =5;
$full = 0;
do
{
$full += $i;
}
while ($i-- >0);
print "Итого: $full ";
?>
```

Приведем структурограммуданного оператора (рис. 13.4).

Рассмотрим два простых цикла и принцип их работы:

```
<?
$num = 0;
print "счетчик цикла while:
";
while ($num++ < 10)
Print "$num";
$num = 0;
Print "счетчик цикла do: ";
do
print "$num";
while ($num++ <10);
?>
```

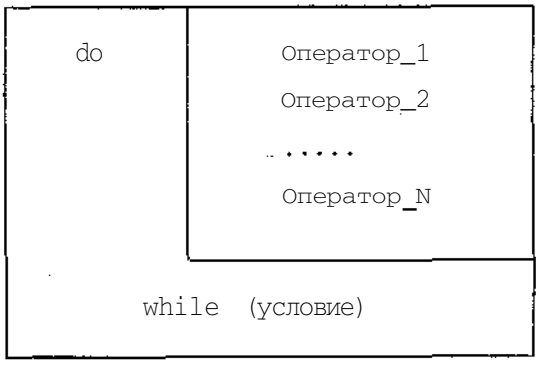


Рис. 13.4. Конструкция do ... while

В результате действия программы на экран будет выведено:

```
счетчик цикла while: 1 2 3 4 5 6 7 8 9 10
счетчик цикла do: 0 1 2 3 4 5 6 7 8 9 10
```

Отсюда можно сделать вывод, что цикл do выполняется 11 раз, а цикл while — 10. В обоих случаях конечное значение счетчика одинаково: 11.

Опытные программисты, использующие язык C, знакомы с иным использованием do ... while, позволяющим прекратить исполнение блока операторов в середине путем внедрения его в цикл do ... while (0) и использования оператора break. Следующий код демонстрирует такую возможность:

```
do
{
if ($i < 5)
{
print "i меньше требуемого значения";
break;
}
$i* = $factor;
if ($i < $minimum_limit)
{
break;
}
print "i значение выражения удовлетворяет потребности";
...
}
while(0);
```

ВНИМАНИЕ

Использование конструкции do ... while(0) приводит к выполнению цикла только один раз. В процессе выполнения оператора итерации будет проверено значение логического выражения, а оно равно false (0), и выполнение цикла завершится.

13.15. Конструкция for

При использовании оператора while цикл обычно выглядит следующим образом:

```
Начальное выражение;
while (условие)
{
```

```
// последовательность операторов
выражение_управления_циклом;
}
```

Последовательность операторов в теле цикла должна быть выполнена фиксированное число раз. Но в PHP, как и в C, предусмотрена конструкция, позволяющая делать то же самое более компактно — цикл for. Ниже приведен цикл for, эквивалентный предыдущему циклу while:

```
for (начальное_выражение; условие; выражение_управления_циклом)
{
// последовательность операторов.
}
```

Первое выражение начальное_выражение безусловно выполняется. В начале каждой итерации (проход цикла) вычисляется условие. Если оно равно true, то цикл продолжается и выполняется вложенный оператор или операторы. Если оно равно false, то цикл заканчивается. В конце каждой итерации исполняется выражение_управления_циклом. Каждое из этих выражений может быть пустым. Если условие пусто, то цикл продолжается бесконечно (PHP по умолчанию считает его равным true, как и в языке C). Это не так бесполезно, как может показаться, так как зачастую требуется закончить выполнение цикла, используя оператор break в сочетании с логическим условием вместо использования логического выражения в for. Если внутри цикла встречается оператор break, цикл прекращает выполнение итераций и управление передается следующей за циклом команде. Если встречается оператор continue, управление передается на начало следующего цикла.

Схематично выполнение цикла for можно представить так:

При выполнении цикла for происходит следующее:

1. Выполняется начальное_выражение. Это происходит только один раз при входе в цикл.
2. Проверяется условие. Если его значение равно true, то осуществляется переход к третьему пункту. Если же оно ложно, то управление передается оператору, следующему за блоком цикла for. Если это значение изначально было равно false, то операторы в теле цикла не выполняются ни разу.
3. Выполняется последовательность операторов.
4. **Вычисляется** выражение_управления_циклом.
5. Возврат в пункт 2.

Инструкции начальный_оператор, условие И выражение_управления_циклом ДОЛЖНЫ удовлетворять некоторым условиям. Обязательно должна присутствовать точка с запятой, т. е. корректная следующая запись:

```
for ( ;;)
{
// последовательность операторов
}
```

Это эквивалентно такому циклу:

```
while (1)
{
// последовательность операторов
}
```

Обычно начальное_выражение используется для инициализации и объявления переменной управления циклом. Здесь можно вставлять и инициализировать и другие переменные.

На рис. 13.5 приведена структурограмма циклической конструкции for.

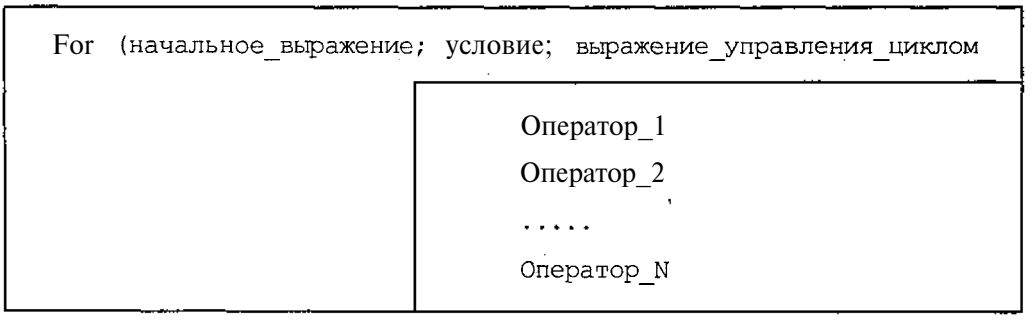


Рис. 13.5. Цикл for

Как несложно догадаться, условие чаще всего используется для проверки условия завершения цикла. Как правило, в нем происходит сравнение переменной управления циклом с некоторым значением.

Инструкция выражения_управления_циклом обычно используется для приращения счетчика цикла. В данном случае безразлично, использовать ли инкремент в префиксной или же постфиксной форме, поскольку на значение управляющего выражения это никак не влияет.

Рассмотрим следующие примеры. Все они выводят номера с 1 по 10:

Пример 13.1. Конструкция for (вариант 1)

```
for ($i = 1; $i <= 10; $i++)
{
```

```
print $i;
}
```

Пример 13.2. Конструкция for (вариант 2)

```
for ($i = 1;;$i++)
{
if ($i > 10)
{
break;
}
print $i;
}
```

Пример 13.3. Конструкция for (вариант 3)

```
$i = 1;
for (;;) {
if ($i > 10) {
break;
}
print $i;
$i++;
}
```

Пример 13.4. Конструкция for (вариант 4)

```
for ($i = 1; $i <= 10; print $i, $i++);
```

Конечно, первый вариант кажется лучшим, но возможность использования пустых выражений в цикле for зачастую является полезной.

PHP также поддерживает альтернативный синтаксис `for`:

```
for (expr1; expr2; expr3): выражение; ...; endfor;
```

ВНИМАНИЕ

Другие языки программирования используют оператор foreach для того, чтобы обрабатывать массивы или списки. PHP использует для этого оператор while и функции list() и each(). Для примера смотрите документацию по этим функциям (на сайте www.php.net).

Приведем пример, в котором будет представлена программа сортировки случайных целых чисел. Алгоритм сортировки сравнивает элементы, находящиеся в массиве далеко друг от друга, и при необходимости меняет их местами. Промежуток между сравниваемыми элементами постепенно уменьшается до единицы — тогда сравниваются соседние элементы и так при необходимости меняются местами. Таким образом, происходит сортировка массива:

```
<?
$size =10;
for ($a =0; $a<$size; $a++)
$aray[$a]= rand();
print "Выводим рандомные значения массива: <br>\n";
for ($a = 0; $a<$size; $a++)
print" $aray[$a] ";
for ($g = $size/2; $g>0; $g /=2)
for($a = $g; $a<$size; $a++) // средний цикл
// внутренний цикл:
for ($j = $a-$g; ($j>=0) && ($aray[$j]>$aray[$j+$g]); $j -= $g)
{
$temp = $aray[$j];
$aray[$j] = $aray[$j+$g];
$aray[$j+$g] = $temp;
}
print "<br> Сортированные значения массива: <br>\n";
for ($a = 0; $a<$size; $a++)
print" $aray[$a] ";
?>
```

В результате выполнения программы на экран будет выведено:

```
Выводим рандомные значения массива:
41 18467 6334 26500 19169 15724 11478 29358 26962 24464
Сортированные значения массива:
41 6334 11478 15724 18467 19169 24464 26500 26962 29358
```

Случайные числа, генерируемые вашим компьютером, могут отличаться от приведенных.

Сначала массив заполняется случайными числами, которые генерирует компьютер. Функция rand используется для генерации случайных чисел. Алгоритм сортировки состоит из трех вложенных циклов for. Внешний цикл устанавливает начальный интервал между сравниваемыми элементами в половину размера массива. При каждой итерации интервал уменьшается в два раза. Внешний цикл завершается тогда, когда размер интервала достигнет нуля. Может ли случиться так, что это условие не будет удовлетворено и цикл будет бесконечным? Может ли переменная $g (интервал) «застрять» в промежутке между 1 и 2? Нет. В конце концов значение $д достигнет 1, а при следующем делении на 2 уменьшится до 0,5, что при отбрасывании целой части (так как $д — целое) даст нуль.

Средний цикл сравнивает элементы массива, разделенные интервалом $д. Переменной цикла $a изначально присваивается значение $д. На каждом шаге цикла ее значение увеличивается на единицу, таким образом, условие окончания цикла всегда достижимо.

Внутренний цикл меняет местами элементы, если они не расположены по порядку. Переменной цикла $j изначально присваивается нулевое значение при проходе среднего цикла. При второй итерации оно становится равным 1 и т. д. Переменная $j уменьшается на величину $g при каждом переходе внутреннего цикла. Условие цикла состоит из двух выражений, причем для продолжения цикла необходимо, чтобы оба они были истинными. Легко заметить, что первое условие в конечном итоге прекратит выполнение цикла, поскольку переменная цикла примет нулевое значение.

13.16. Конструкция foreach

Данный оператор цикла является новым и используется только в PHP4. Аналогичный оператор можно найти в Perl. Он призван упростить последовательную обработку всех элементов массива. Существуют две системы синтаксиса для этого оператора:

```
Foreach (array_expression as $value) выражения
Foreach (array_expression as $key => $value) выражения
```

В первом случае значения переменных массива по очереди присваиваются переменной $value, над которой производятся действия. В итоге получается своеобразный указатель на то или иное значение переменной из необходимого массива.

Второй способ аналогичен первому, только в этом случае помимо значения переменной будет еще использоваться и ключ, т. е. обозначение этой переменной. Рассмотрим это на примерах.

Пример 13.5. Конструкция foreach (вариант 1)

```
$a = array (1, 2, 3, 17);
foreach ($a as $v) {
```

```
print "Current value of \$a: $v.\n";
}
```

Результат выполнения программы:

```
Current value of $a: 1.
Current value of $a: 2.
Current value of $a: 3.
Current value of $a: 17.
```

Ниже приведен аналогичный пример, только обратите внимание, что индекс массива всегда постоянный и равный 0. А значения передаваемой переменной изменяются. Это очень важно.

Пример 13.6. Конструкция foreach **(вариант** 2)

```
$a = array (1, 2, 3, 17);
$i = 0; /* для более подробного анализа*/
foreach($a as $v) {
print "\$a[$i] => $v.\n";
}
```

Результат этой программы:

```
$a[0] => 1.
$a[0] => 2.
$a[0] => 3.
$a[0] => 17.
```

В следующем примере показана работа оператора f oreach на основании массива значений. Помимо этих значений каждая переменная имеет свой собственный ключ.

Пример 13.7. Конструкция foreach **(вариант** 3)

```
$a = array (
"one" => 1,
"two" => 2,
"three" => 3,
"seventeen" => 17
);
foreach($a as $k => $v) {
```

```
print "\$a[$k] => $v.\n";
}
```

После выполнения этой программы на экран будет выведено:

```
$a[one] => 1.
$a[two] => 2.
$a[three] => 3.
$a[seventeen] => 17.
```

Думаем, на основании этих примеров вы смогли получить подробное описание работы данного оператора. Посмотрим, почему его использование намного удобнее, чем цикла while:

```
reset ($arr);
while (list(, $value) = each ($arr)) {
echo "Value: $value<br>\n";
}
foreach($arr as $value) {
echo "Value: $value<br>\n";
}
```

Это в случае второй конструкции (пример 13.6). А вот при использовании ключей (пример 13.7):

```
reset($arr);
while (list($key, $value) = each ($arr)) {
echo "Key: $key; Value: $value<br>\n";
}
foreach($arr as $key => $value) {
echo "Key: $key; Value: $value<br>\n";
}
```

Как видно из примеров, использование цикла while в этом случае намного сложнее для понимания, к тому же необходимо применение еще одной функции — reset().

ВНИМАНИЕ

Перед началом выполнения foreach внутренний указатель массива автоматически возвращается на первый элемент. Это означает, что вы не должны повторно вызывать функцию reset() перед циклом foreach.

Также заметьте, что foreach работает с копией указанного массива, поэтому указатель не изменяется с каждой конструкцией.

Рассмотрим пример, в котором число с плавающей запятой требуется округлить до определенного разряда. Проблема связана с теми же погрешностями представления, которые затрудняют сравнение чисел, а также возникает в ситуациях, когда точность ответа намеренно снижается для получения более наглядного результата:

```
<?
$a[0]=4.3;
$a[1]=4.5;
$a[2]=4.7;
$a[3]=-4.3;
print"Округляем с помощью round<br>\n";
foreach ($a as $b) {
$c = round ($b);
print " $c <br>\n ";
}
print"Округляем с помощью floor <br>\n";
foreach ($a as $b) {
$c = floor ($b);
print " $c <br>\n ";
}
print"Округляем с помощью ceil <br>\n";
foreach($a as $b) {
$c = ceil($b);
print " $c <br>\n ";
}
?>
```

Исходные значения:

```
$a[0]=4.3;
$a[1]=4.5;
$a[2]=4.7;
$a[3]=-4.3;
```

Результат будет следующий:

```
Округляем с помощью round
4
5
5
-4
Округляем с помощью floor
4.0
4.0
4.0
-5.0
Округляем с помощью ceil
5.0
5.0
5.0
-4.0
```

Округление серьезно отражается на работе некоторых алгоритмов, потому используемый метод должен быть точно указан. В особо важных приложениях (например, в финансовых вычислениях или системах наведения ракет) грамотный программист реализует свою собственную функцию, не полагаясь на встроенную логику языка (или ее отсутствие).

В PHP существуют следующие функции, предназначенные для округления чисел с плавающей запятой до целых: round(), ceil() и floor(). Встроенная функция round() возвращает целую часть числа с плавающей запятой. Функция floor() и ceil() округляет аргументы в большую или меньшую стороны, соответственно, до ближайшего целого.

13.17. Операторы перехода

Операторы перехода (jump statements) осуществляют безусловную передачу управления в определенное место программы. Мы рассмотрим следующие операторы перехода:

- break,
- continue.

13.18. Оператор break

Вы уже встречали оператор break в параграфе, посвященном оператору switch (см. п. 13.11). Когда оператор break употребляется в последовательности операторов case, управление программой передается оператору, следующему за блоком switch. Оператор break прерывает исполнение ближайшего внешнего оператора while, do, for или switch. Управление передается следующему за прерванным оператору. Например, использование оператора break в цикле while:

```
<?
$arr = array ('один', 'два', 'три', 'четыре', 'стоп', 'пять');
while (list (, $val) = each ($arr)) {
if ($val == 'стоп') {
break ; /* в этой строчке вместо break можно использовать
break 1 */
}
echo "$val<br>\n";
}
?>
```

Выполнение этой программы будет осуществляться до тех пор, пока значение переменной не станет равной стоп. После этого программа прерывается оператором break. В результате на экран браузера будет выведено:

```
один
два
три
четыре
```

Если бы оператора break не было, то к этим фразам бы добавилось еще две:

```
стоп
пять
```

При употреблении break можно также вместо него пользоваться таким синтаксическим обозначением break 1 — результат работы программы будет аналогичным.

Рассмотрим еще один пример:

```
<?
$i = 0;
while (++$i) {
```

```
switch($i) {
case 5:
echo "Программа прервалась на 5 <br>\n";
break 1;
case 10:
echo "Программа прервалась на 10, и произошла остановка выполнения. <br>\n";
break 2;
default:
break;
}
}
?>
```

Переменная $i в рассмотренной программе будет постепенно увеличивать. Как только значение ее станет равным 5, она автоматически по ссылке попадет на оператор case 5:, далее произойдет выполнения выражение echo "Программа прервалась на 5
\n";, что приведет к выводу строчки "Программа прервалась на 5", хотя, именно в этот момент программа еще не завершилась (это произойдет, как только выполнится оператор break 1;). Но даже в этом случае произойдет выход только из конструкции switch, но не из цикла. Дальше прибавляется по единице к имеющейся пятерке. Как только значение переменной станет равно 10, выполнится оператор break 2, за счет чего произойдет выход из оператора switch и из самого цикла while. Выполнение программы прервется окончательно. В результате получится следующее (рис. 13.6):

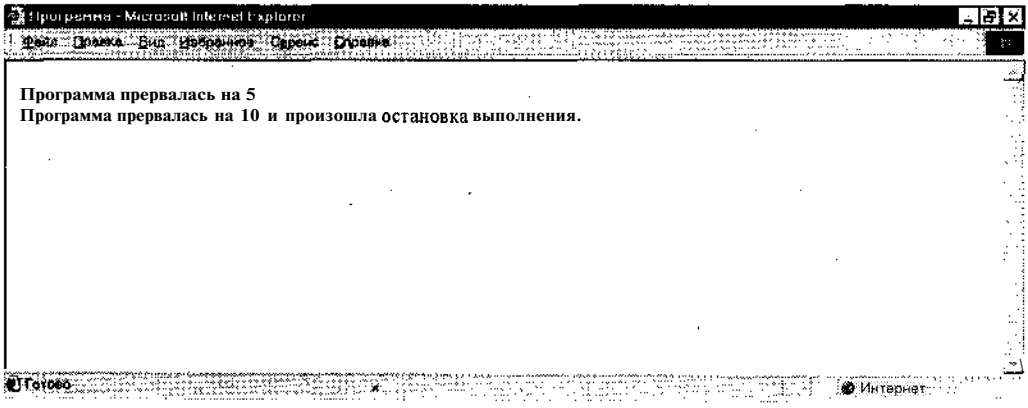


Рис. 13.6. Результат выполнения скрипта

13.19. Оператор continue

Оператор continue прекращает текущую итерацию в ближайшем внешнем цикле while, do или for. Управление передается проверочному выражению циклов while и do или выражению управления циклом в цикле for.

Пример 13.8. Оператор continue (вариант 1)

```
while (list ($key, $value) = each ($arr)) {
if (!($key % 2)) { // пропуск нечетных значений
continue;
}
do_something_odd($value);
}
```

Пример 13.9. Оператор continue (вариант 2)

```
$i = 0;
while ($i++ < 5) {
echo "Outer<br>\n";
while (1) {
echo " Middle<br>\n";
while (1) {
echo " Inner<br>\n";
continue 3;
}
echo "Это никогда не будет выведено.<br>\n";
}
echo "Никто не делает этого.<br>\n";
}
```

Эти два примера аналогичны описанным ранее при использовании оператора break Думаем, что понять его суть не составит труда. Если вдруг все-таки возникнут какие-либо вопросы, просто поэкспериментируйте самостоятельно.

13.20. Оператор возврата return

Оператор возврата return передает управление программой вызывающей функции. Обычно оператор return возвращает значение некоторого выражения, кото-

рое становится значением функции. Пустая функция — это функция, не возвращающая никакое значение. В этом случае с оператором return не связывается никакое выражение. Например:

```
return 1; // функция возвращает целое число
return $z/$e; // возвращает $z/$e
return; // пустая функция не возвращает значение
```

Оператор return может располагаться *в* любом месте функции. Функции и операторы возврата будут рассмотрены подробнее в гл. 14.

13.21. Включение исходного кода текста, содержащегося в файле

Каждый, сделавший хоть одну страничку в Сети, сталкивался с проблемой изменения тех или иных данных на ней. Конечно, это не сложно, когда страничек несколько, но если вы сделали большой сайт, дополнение (например, в меню) в сотни файлов может превратиться в настоящий кошмар. PHP быстро решает эту проблему, позволяя вкладывать одну страницу в другую. Достигается это с помощью операторов require и include. После них должен стоять путь к вкладываемому файлу. Например:

```
include ('test.phtml');
```

Различие между указанными операторами заключается в том, что require подменяется содержимым указанного файла и может быть использован только один раз, а include — вставляет и выполняет содержимое указанного файла, что позволяет применять его несколько раз в цикле. Вложения файлов могут происходить только внутри серверного пространства, доступного PHP. Другими словами, вы не можете использовать в имени файла `http://`.

Достаточно часто встречаются сайты, ссылки которых включают в себя специальные символы — &, ?, %, что указывает на работу програмы PHP. Дело в том, что если в конце ссылки добавить `?имя=значение`, это значение будет доступно под этим же именем в файле, куда указывает ссылка. Если необходимо добавить несколько имен, они могут быть разделены знаком &. Теперь мы можем сделать сайт, который будет доступен с помощью только одной странички, а всю остальную информацию эта страничка будет выводить на основании полученных по ссылке данных. Вид такой ссылки будет примерно таким:

```
http://my_domain_name/index.phtml?link=1
```

Единица в конце ссылки и есть наш параметр, который будет представляться в файле `index.phtm`. Например:

```
<html>
...
<?php
```

```
$name = "";
if ( $link = = 1 )
{
$name = "name1.phtml";
}
if ($link = = 2)
{
$name = "name2.phtml";
}
if ($link = = 3)
{
$name = "name3.phtml";
}
include ($name);
?>
...
</html>
```

Обратите внимание: написанный нами код учитывает ситуацию, когда посетитель по разным причинам указал неправильный параметр. В этом случае выводится заготовленное сообщение об ошибке. Если же параметр соответствует какому-либо из файлов сайта, он вкладывается в код файла index.phtml и исполняется. Таким образом, начало и конец остаются одинаковыми, а изменяется только середина. И какие-либо изменения уже не кажутся такими страшными, как раньше. Ведь сделать их надо только в одном файле, а отразится это на всем сайте.

Есть и другой путь. Его суть заключается в том, что у PHP есть доступ к так называемым переменным окружения сервера. Одна из этих переменных — запрашиваемый посетителем путь относительно адреса сайта. И этот путь становится нам доступен для использования. В этом случае ссылки будут такого вида:

```
http://my_domain_name/index.phtml?patch/name.phtml
```

Вторая часть ссылки — patch/name.html — будет нам доступна, если мы считаем параметр $query_string. Например:

```
$add = $query_string;
```

Теперь изменим наш головной файл index.phtml, чтобы все работало автоматически. А если запрашиваемый параметр не будет указан (будет равен пустой строке), чтобы что-то открыть, присвоим переменной $add имя файла, который дол-

жен быть открыт как главная страничка. Пусть это будет файл main.phtml. Тогда код будет выглядеть следующим образом:

```
<html>
...
<?php
$add = $query_string;
if ($add = = "")
{
$add = "main.phtml";
}
include($add);
?>
...
</html>
```

Как видите, еще проще.

ВНИМАНИЕ

Этот метод хоть и проще первого, но открывает путь к получению информации о сервере, где расположен сайт с такой организацией структуры. Злоумышленник или просто любопытный человек при наличии определенных обстоятельств и знаний сможет много узнать о вашем сервере, а это открывает прямой путь к взлому. Так что будьте осторожны.

13.22. Оператор require()

Принцип работы оператора require () подобен директиве препроцессора #include () в языке программирования C++. Если вы знакомы с этим языком, то освоить следующие операторы для вас не составит никакого труда, впрочем, даже если вы и не знакомы, мы постараемся более подробно описать принципы работы данных операторов. Оператор require () заменяет, вызывает содержимое файлов, путь и имя которого были указаны в блоке оператора:

```
require ('path/filename.html');
```

Если параметры URL fopenwrapperSB PHP установлены как enabled (допустимо) (как правило, они — default (недостаточная), то вы можете также производить определение файла в операторе require (), используя URL (Uniform Resource Locator — унифицированный указатель ресурса) вместо того, чтобы использовать локальный путь к необходимому файлу в операторе require (). Например:

```
$url = http://my_new_adress/index.phtml;
require ($url) ;
```

Оператор require () не является функцией. Скорее всего его можно назвать конструкцией языка. Почему мы не можем считать этот оператор функцией:

- require() не подчинен к каким-либо управляющим структурам;
- require() не возвращает какое-либо значение, что в свою очередь делает функция.

Попытка вызова какого-либо значения из оператора require () приведет к ошибке и прерыванию дальнейшего выполнения программы.

В отличие от include () require () всегда производит чтение строки адреса файла, даже в том случае, когда он не выполняется. Если вам необходимо условно включить файл, используйте оператор include (). Просто условные выражения не считаются эффективными при работе с оператором require (). Однако если строка, на которой расположен оператор require (), не будет выполнена, то тогда не будет выполнена ни одна строка кода в файле по этому адресу. Это логично, так как именно доступ к файлу в данном случае и осуществляется через оператор require ().

Структуры выполнения цикла не затрагивают режима работы оператора require (), хотя код, содержащийся в конечном файле, все еще подчинен циклу. Из этого можно сделать вывод, что оператор require () выполняется только один раз в отличии от include().

Следовательно/вы не можете помещать оператор require () со всеми прилагающимися к нему инструкциями в блок оператора цикла и ожидать при работе того самого цикла различного выполнения данного оператора на каждой итерации. Чтобы воспользоваться этим, предлагаем вам применить оператор include ().

Когда файл работает с оператором require (), содержащийся код наследует переменные возможности строки, на которой require () выполняется. Любые переменные, доступные на этой строке, будут доступны в пределах вызванного файла. Если оператор require () выполняется внутри функции в пределах вызывающего файла, то весь код, содержащийся в вызванном файле, будет вести себя так, как если бы он был определен внутри этой функции.

В том случае, если оператор require () работает с файлом, который вызван через HTTP и использует fopen wrappers, и если адрес станции интерпретирует конечный файл как PHP-код, то в этом случае переменные могут быть переданы в оператор require (), используя URL-запрос, как метод HTTP Get. Это не то же самое, что вызов файла оператора require (), так как скрипт практически продолжает выполнение сценария на удаленном сервере и результаты будут потом включены в локальный сценарий:

```
/* не будет работать, так как не был обработан сервером*/
require ("http://my_domain_name/file.txt?varone=1&vartwo=2");
/* не будет работать, так как имя файла 'file.php?varone=1&vartwo=2'
```

```
ориентировано на локальную файловую систему */
require ("file.php?varone=1&vartwo=2");
/* будет работать без ошибок */
require ("http://my_domain_name /test.php?varone=1&vartwo=2");
$varone = 1;
$vartwo = 2;
require ("file.txt");
require ("file.php");
```

Данные примеры помогут вам наглядно разобраться со способами применения оператора require (), а также не допустить ошибок при создании программ. Далее мы рассмотрим оператор include (), подобный оператору require (), и основные принципы его работы.

13.23. Оператор include()

Очень много было сказано об этом операторе (см. п. 13.21). Поэтому основное внимание уделим примерам его работы.

Очень часто используется внутри циклов, что отличает его от оператора require () и является преимуществом. Например:

```
$files = array ('first.inc', 'second.inc', 'third.inc');
for ($i = 0; $i < count($files); $i++)
{
include($files[$i]);
}
```

Include () отличается от require () также тем, что оператор include выполняется каждый раз при его встрече, а require () заменяется на содержимое указанного файла безотносительно, будет ли выполнено его содержимое или нет.

ВНИМАНИЕ

Так как include() — это специальный оператор, необходимо заключать его в фигурные скобки при использовании внутри условного оператора. Например:

```
/* это неправильно и не будет работать, как хотелось бы */
if ($condition)
include($file);
else
```

```
include($other);
/* а вот это — верно */
if ($condition) {
include($file);
} else {
include($other);
}
```

Когда файл исполняется, интерпретатор пребывает в «режиме HTML», т. е. будет выводить содержимое файла, пока не встретит первый стартовый тег PHP (<?).

13.24. Оператор require_once()

Оператор require_once() заменяет сам себя указанным файлом. Данный оператор работает наподобие директивы препроцессора #include в языке C и в этом отношении похож на принцип работы оператора require(). Основное отличие состоитвтом, что использование оператора require_once() обеспечит добавление кода к вашему сценарию только один раз и поможет избежать столкновений со значениями переменных или именами функций.

Продемонстрируем работу оператора require_once() на примере. Для этого сразу создадим пример с оператором require(), который будет выдавать ошибку, так как в данном примере мы хотели бы показать основные тонкости программирования с использованием оператора require_once(). Создадим два файла: test1.inc и test2.inc.

Пример 13.10. Файл test1.inc

```
<?php
echo "Работа файла test1.inc<br>\n";
function goodTea()
{
return "Проверка прошла успешно!";
}
?>
```

В данном примере вызывается файл test1.inc и сразу выполняется функция echo(), которая выводит строку «Работа файла test1.inc». Это сделано для того, чтобы вы смогли проследить, когда какой файл выполняется и какой из этого следует результат. При правильном написании дальнейшей программы выполняется функция goodTea(), которая возвращает результат — строку «Проверка прошла успешно».

Пример **13.11.** Файл **test2.inc**

```
<?php
require("test1.inc");
echo"Вызов файла test2.inc<br>\n";
?>
```

При вызове файла test2.inc сразу выполняется оператор require («test1.inc»);, что опять переводит выполнение программы в файл test1.inc. После этого из файла test2.inc при правильном написании программы должна выполняться функция echo О, которая выводит строку «Вызов файла test2.inc». Далее создаем файл test3.php.

Пример 13.12. Файл test3.php

```
<?php
require("test2.inc");
/* все, что написано далее, сгенерирует ошибку */
require("test1.inc");
echo "Произошло полное выполнение программы <br>\n";
echo "Вызов всех файлов проделан успешно <br>\n";
echo "Результат работы функции goodTea: " .goodTea(). "\n";
?>
```

Выполнение данного файла сразу переводит компилятор в файл test2.inc, после этого из файла test2.inc управление передается в файл test1.inc, в котором и произойдет ошибка. Ошибка будет выглядеть следующим образом:

```
Работа файла test1.inc
Вызов файла test2.inc
Работа файла test1.inc
Fatal error: Cannot redeclare goodtea() in z:\home\localhost\www\test1.inc
on line 3
```

В начале программа будет выполняться по установленному плану, но потом произойдет ошибка при обращении к функции goodTea. Данная ошибка говорит о том, что интерпретатор PHP не может повторно объявить функцию, которая была ранее объявлена. Для устранения данной ошибки предлагаем переписать код программы примера 13.11 следующим образом:

```
<?php
require_once("test1.inc");
```

```
echo"Вызов файла test2.inc<br>\n";
?>
```

А примера 13.12так:

```
<?php
require_once("test2.inc");
/* все, что написано далее, сгенерирует ошибку */
require_once("test1.inc");
echo "Произошло полное выполнение программы<br>\n";
echo "Вызов всех файлов проделан успешно <br>\n";
echo "Результат работы функции goodTea: ".goodTea()."\n";
?>
```

Изменению подверглись следующие операторы:

`require("test1.inc")` — `require_once("test1.inc")`

`require("test2.inc")` — `require_once("test2.inc")`

После выполнения программы с этими изменениями получим следующий результат:

```
Работа файла test1.inc
Вызов файла test2.inc
Произошло полное вьшолнение программы
Вызов всех файлов проделан успешно
Результат работы функции goodTea: Проверка прошла успешно
```

Как видно, ошибок нет.

СОВЕТ

Аналогично поведению директивы препроцессора flinclude языка C, require_once() работает в процессе компиляции, т. е. в момент времени, когда происходит анализ самого сценария, прежде чем произойдет его выполнение. Данный оператор не должен применяться для частей скрипта, которые должны вставляться динамически в течение его выполнения. Для этой операции лучше пользоваться такими операторами, как include_once() или include().

13.25. Оператор include_once()

Работа данного оператора подобна оператору include (), со значительным отличием, состоящим в том, что если код из файла уже был включен, то его нельзя включить повторно.

Оператор include_once() должен быть использован в случаях, в которых тот же самый файл мог бы быть включен и оценен больше, чем один раз, при частом выполнении сценария, при этом также следует убедиться в том, что код включен один раз, дабы избежать проблем с функциональными преобразованиями и назначенными значениями переменных.

Оператор include_once() был добавлен в PHP версии 4.0.1pl2.

ВНИМАНИЕ

Суффикс _once значит, что если файл уже подключен, то второй раз не подключается (очень полезно при подключении файлов с классами или библиотеками функций). Require от include отличается тем, что при require, если файл не найден, скрипт обрывается по Fatal Error, а при include – продолжает выполнение до конца с предупреждением о том, что нет файла для функции include().

Заключение

Операторы бывают четырех видов:

- последовательные операторы;
- операторы выбора;
- операторы цикла;
- операторы перехода.

Последовательные операторы — это все операторы, которые не управляют процессом выполнения программы. Сюда можно отнести операторы, не попадающие в другие категории.

Операторы выбора работают с другими операторами в зависимости от определенных критериев. К ним принадлежат операторы if и switch.

Операторы цикла (while и do) способны выполнять последовательности других операторов до тех пор, пока не будет выполнено определенное условие, оператор for делает это фиксированное число раз.

Операторы перехода безусловно переводят исполнение программы в определенное место. Оператор break прерывает ближайший внешний цикл или оператор switch. Оператор continue прекращает текущую итерацию ближайшего внешнего циклического оператора. Оператор return отдает использование вызывающей функции.

Также мы рассмотрели способы включения исходного кода текста, содержащегося в файле при помощи таких операторов:

- require(),
- include().

Глава 14
Базовые концепции функций

Программисты всегда стремились к простоте и понятности кода. Умение осуществлять разнообразные операции с функциями, а в частности понимать принципы и способы взаимодействия функций с переменными, позволит не только создавать лаконичные PHP-скрипты, но и получить максимально возможную производительность вашего скрипта. Это же в свою очередь приведет к быстрой обработке данных и выведению требуемого результата в окне браузера. Более подробно о функциях PHP вы узнаете из этой главы. Если вы уже знакомы с ними, то просто пропустите ее. В главе рассматриваются следующие вопросы:

- определяемые пользователем функции;
- переменныефункции;
- возвращениерезультатов;
- аргументыфункции;
- передача аргументов поссылке;
- значения переменных поумолчанию;
- оператор old_function;
- списки аргументов переменнойдлины.

14.1. Определяемые пользователем функции

Функция может быть объявлена так:

```
function foo ($arg_1, $arg_2, ..., $arg_n) {
echo "Пример функции.\n";
return $retval;
}
```

Соответственно, внутри функции может быть любой верный код PHP, возможно объявление другой функции или класса.

14.2. Переменные функции

PHP поддерживает так называемые переменные функции. Этот термин означает, что если вы добавите к своей функции имя переменной в скобках, то PHP будет искать функцию с именем, равным значению этой переменной, и предпримет попытку ее выполнить. Это можно использовать для реализации обратных вызовов, таблиц функций и т. д.

Пример 14.1. Переменная функция

```
<?php
function foo() {
echo "In foo()<br>\n";
}
function bar( $arg = '' ) {
echo "In bar(); argument was '$arg'.<br>\n";
}
$func = 'foo';
$func();
$func = 'bar';
$func( 'test' );
?>
```

Результат выполнения программы представлен на рис. 14.1.

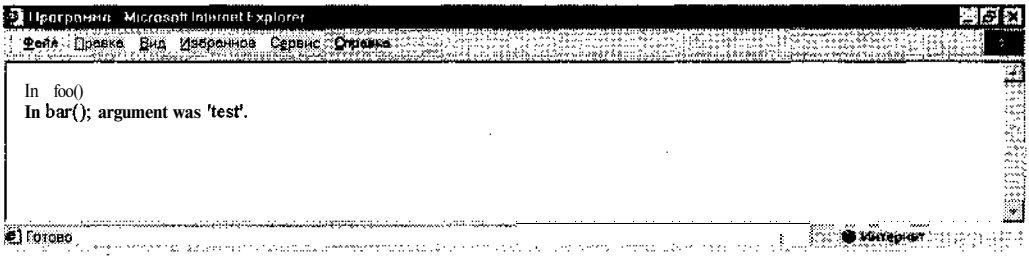


Рис. 14.1. Результат выполнения скрипта

14.3. Возвращение результатов

Результаты возвращаются через необязательный оператор return. Возвращаемый результат может быть любого типа, включая списки и объекты. Например:

```
function my_sqrt ($num) {
return $num * $num;
}
echo my_sqrt (4); // outputs '16'.
```

Массивы не могут быть возвращены в качестве результата, однако это можно реализовать другим способом, например:

```
function foo() {
return array (0, 1, 2);
}
list ($zero, $one, $two) = foo();
```

Чтобы функция возвращала ссылку, следует использовать оператор ссылки & в объявлении функции и при присвоении возвращаемого значения переменной, например:

```
function &returns_reference() {
return $someref;
}
$newref =&returns_reference();
```

14.4. Аргументы функции

Передавать информацию в функции можно с помощью списка аргументов, который представляет из себя разделенный запятыми список переменных и (или) констант.

В PHP можно передавать аргументы по ссылке, передача по значению осуществляется по умолчанию. Списки аргументов переменной длины поддерживаются только в PHP 4 и выше. Аналогичного эффекта можно достичь в PHP 3, передавая в функцию массив аргументов, например:

```
function takes_array($input) {
echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
```

14.5. Передача аргументов по ссылке

По умолчанию все аргументы функции передаются только по значению, но если вам понадобилось модифицировать аргументы в функции, то вы можете передать их по ссылке.

Если же вы хотите сделать так, чтобы аргумент всегда передавался по ссылке, то следует поставить знак & перед именем аргумента в объявлении функции. Например:

```
function foo( &$bar ) (
$bar .= 'and something extra.';
}
$str = 'This is a string, ';
```

```
foo ($str);
echo $str; // выведет: 'This is a string, and something extra.'
```

Если вы хотите передать аргумент по ссылке в случае, когда по умолчанию такого не делается, то добавьте & перед именем аргумента в вызове функции, например:

```
function foo($bar) {
$bar .= ' and something extra.';
}
$str = 'This is a string, ';
foo($str);
echo $str; // выведет 'This is a string, '
foo(&$str);
echo $str; // выведет 'This is a string, and something extra.'
```

14.6. Значения переменных по умолчанию

Функции могут определять значения по умолчанию для скалярных аргументов, как и в языке программирования C++. Например:

```
function makecoffee($type = "cappucino") {
echo "Making a cup of $type.\n";
}
echo makecoffee();
echo makecoffee("espresso");
```

В результате будет выведено:

```
Making a cup of cappucino.
Making a cup of espresso.
```

Значение по умолчанию должно быть константой, а не переменной или, к примеру, членом класса.

ВНИМАНИЕ

Аргументы, объявленные по умолчанию, обязательно должны быть расположены справа от всех аргументов, объявленных другим методом, в противном случае это не будет работать так, как задумано.

Рассмотрим пример:

```
function makeyogurt($type = "acidophilus", $flavour) {
```

```
return "Making a bowl of $type $flavour.\n";
}
echo makeyogurt("raspberry"); // не будет работать, как ожидалось
```

Этот пример выведет следующее:

```
Warning: Missing argument 2 in call to makeyogurt() in
/usr/local/etc/httpd/htdocs/php3test/functest.html on line 41
Making a bowl of raspberry.
```

А теперь сравните с этим:

```
function makeyogurt ($flavour, $type = "acidophilus") {
return "Making a bowl of $type $flavour.\n";
}
echo makeyogurt ("raspberry"); // а вот это работает
```

Результат программы:

```
Making a bowl of acidophilus raspberry.
```

14.7. Оператор old_function

Оператор old_function позволяет вам определять функцию, используя синтаксис PHP/FI2 (за исключением того, что вы должны заменить function на old_function).

Это свойство используется только для совместимости и лишь конверторами PHP/FI2 -> PHP 3.

Описанные таким образом функции никак не могут быть вызваны из служебного кода PHP. Это значит, что вы не можете использовать их в функциях типа usort(), array_walk() и register_shutdown_function(). Но вы можете обойти это путем введения специальной функции в терминах PHP 3, которая будет вызывать old_function.

14.8. Списки аргументов переменной длины

В PHP 4 имеется поддержка списков аргументов переменной длины в пользовательских функциях. С помощью функций func_num_args(), func_get_arg() и func_get_args() этого очень легко достичь. Не требуется никакого специального синтаксиса, списки аргументов могут по-прежнему явно задаваться в определении функции и будут нормально работать.

Заключение

Рассмотренная нами глава 14 «Базовые концепции функции» включает способы задания функций, а также принципы работы. Для систематизации материала выделим следующие концепции.

- Функции, определяемые пользователем. Пользователь, т. е. программист, задает непосредственно имя функции и назначает тот перечень операций, которые она должна выполнять. В свою очередь функция при правильном задании параметров осуществляет выполнение поставленной задачи и выводит результат пользователю. Помимо всего прочего в самой функции может быть объявлена еще одна функция.
- Переменные функции. Они, как и в первом случае, создаются пользователем, но несут другой смысл. Это значит, что если вы к фукнции добавите имя переменной в скобках, то PHP будет искать функцию с именем, равным значению этой переменной, и предпримет попытку ее выполнить.
- Возвращение результата. В зависимости от того, как вы производите задание функции и непосредственно операций, выполняемых в этой функции, вы будете получать необходимый вам результат. Будьте предельно внимательны, так как при работе с функциями, как и с классами, очень часто возникают ошибки по причине не донца осмысленного представления работы функции при возврате результата.

Глава 15
Классы и объекты

Наряду со ссылками и модулями в PHP появились классы и объекты. При написании программ необязательно пользоваться объектами, в отличие от языка Java, где программы представляют собой экземпляры объектов. Объект представляет собой переменную, принадлежащую к некоторому классу. Методами называться функции, ассоциируемые с классом или объектом.

PHP поддерживает несколько разных стилей. Благодаря этому программисты решают свои задачи так, как им нравится.

Эта глава является большим помощником для создателей больших программ. Излагаемая здесь информация является ключевой для понимания объектно-ориентированного подхода к программированию.

Если вы предпочтете оставить ознакомление с классами на потом и остаться верным структурному программированию, отложите изучение этой главы. Конечно, при этом ваше представление о средствах языка останется незавершенным, поэтому лучше прочтите ее и сохраните в уме до лучших времен, когда накопленный опыт и знание скрытых механизмов сделают излагаемый материал более доступ-

ным. Когда вы будете к этому готовы, данная глава станет вашим первым шагом к пониманию того, как с помощью PHP решать весьма сложные задачи.

В этой главе вы найдете следующую информациию о работе классов:

- обзор классов как типов данных;
- все ли можно считать классами;
- когда использовать классы;
- когда не использовать классы;
- синтаксис классов;
- данные класса;
- методы класса;
- задания значений изменяющимся переменным в классах;
- расширение классов;
- работа с переменными класса;
- манипуляция с именами;
- манипуляция уровнем сложности при работе с классами;
- указатель на самого себя $this;
- замечания по объектной терминологии;
- ссылки внутри конструктора.

15.1. Обзор классов как типов данных

Классы введены в язык как средство выражений взаимосвязей между данными и функциями путем связывания их в целую конструкцию. Цели создания подобных конструкций могут быть разнообразными, например теми же, которыми мы руководствуемся, используя простые определения для обозначения сложных понятий материального мира.

Например, фраза «О какой прекрасный бриллиант» несет в себе ту же информацию, что и фраза «Какой исключительно элегантный экземпляр ограненного кристаллического спрессованного углерода». Все это длинное определение умещается в слове «бриллиант».

Эта особенность человеческого мышления обеспечивает возможность оперирования сложными явлениями посредством простых понятий, что неимоверно упрощает как общение, так и программирование. Если удается сводить методы манипулирования классами к операциям и интуитивно понятным функциям, то сколь угодно сложные классы можно сделать практически столь же «ручными», как и базовые типы данных.

Достичь этого не всегда просто, но язык PHP, со своей стороны, обеспечивает программиста всем необходимым, позволяя выражать весьма сложные вещи в понятном и наглядном виде.

15.2. Все ли можно считать классами

Не переусердствуйте — не стоит пытаться все подряд представлять в виде классов. Во многих случаях классы являются не самым подходящим средством. Хотя это не мешает вам разобраться, насколько они подходят для реализации вашей концепции.

15.3. Когда использовать классы

Дадим несколько общих советов о том, как выяснить, когда стоит использовать классы и когда лучше обойтись без них (но не воспринимайте эти указания как догму):

- если в вашу абстракцию входит более одной-двух составных частей, независимо, данных или функций, то смело создавайте класс;
- старайтесь также использовать классы в тех случаях, когда следует обеспечить корректную инициализацию и деинициализацию ресурса. Это относится к таким вещам, как файлы, аппаратные устройства или процессы, требующие определенной серии шагов для инициализации и освобождения ресурса.

15.4. Когда не использовать классы

Обычно без классов можно обойтись:

- когда ваша конструкция содержит слишком мало данных-членов или функций-членов;
- одиночные функции также чаще всего бессмысленно оформлять в виде класса, примером этого могут служить алгоритмы сортировки или поиска (единственным переменным звеном в сортировке или поиске может являться тип данных).

Сложность программирования заключается помимо всего прочем еще и в том, что оценка качества решения в высшей степени субъективна. Если два радикально отличающихся решения обеспечивают правильный результат и примерно одинаковое время выполнения, то какая реализация лучше, а какая хуже — это в значительной степени дело вкуса. Сейчас, пока вы учитесь, просто стремитесь к правильности. Достигнув достаточного мастерства, делайте акцент в сторону выразительной простоты, и лишь потом уделяйте внимание скорости. Правильно — это всегда лучше, чем быстро и неверно, с простыми выражениями легче работать, чем с длинными.

15.5. Синтаксис классов

Как и полагается, начнем с простого — поговорим о синтаксисе.

Определение класса начинается с ключевого слова class. Независимо от назначения класса внешний синтаксис его остается неизменным.

Общая схема построения класса выглядит следующим образом:

```
Class NAME_CLASS
{
// блок операторов класса
}
```

В языке С после завершения тела класса, т. е. после последней закрывающей скобки, необходимо поставить точку с запятой. В PHP использование точки с запятой в конце класса не принципиально. Имеется в виду, что если вы напишите

```
class univer
{
...
};
```

это не вызовет ошибки в PHP, а более того, для определения класса в PHP чаще всего пользуются вышеприведенным синтаксисом, т. е. без точки с запятой.

Слово class — это ключевое слово языка PHP, name_class — уникальное имя. Давая классам имена, можно придерживаться распространенной традиции использовать символы только верхнего регистра:

```
class NEWSHOP
{
}
```

Вы также можете создавать имена такого типа:

```
class RuningShop
{
}
```

Это также не будет ошибкой, задание имени не является принципиальной особенностью в программировании классов. Тело класса — его определение — заключается междулевой и правой фигурными скобками, например:

```
class Demonstration
{
// тело класса
}
```

В приведенном примере представлен вид простого класса. Строка, содержащая ключевое слово class, может также нести некоторую дополнительную информацию в случаях, если класс является наследующим, классом-шаблоном или просто это объявление является опережающим.

Пример опережающего объявления

```
class Mynew {}; // объявление класса
```

т. е. без содержимого самого тела класса. Такие объявления классов иногда используются для более наглядного представления принципов работы программы.

15.6. Данные класса

Тело класса состоит из данных и функций.

Данные — это то, чем класс оперирует. Данные и функции классов называются членами (members). Данные могут быть произвольного типа.

В простейшем случае данные-члены — это переменные базового типа. Не существует никакого предписанного порядка расположения данных-членов в классе. Синтаксис определения данных-членов в точности совпадает с синтаксисом определения обычных переменных. Главное отличие заключается в том, что перед тем, как использовать какую-либо переменною в классе, ее необходимо определить. Синтаксис определения состоит в том, что перед переменной используется зарезервированное слово var (variables). Это значение показывает интерпретатору, что в классе задана переменная, с которой в последующем будет происходить работа.

ВНИМАНИЕ

Определение переменной происходит не сразу после var, а ниже в программе, т. е. интерпретатор PHP объявит ошибку, если произойдет присвоение значения переменной после слова var.

В следующем примере показано, как происходит определение переменных в классе:

```
class Cars
{
var $items; // простое целое данное-член
var $spirit;
var $money;
var $saturday;
}
```

Приведем пример, который показывает неправильную работу с данными:

```
class Cars
{
var $items = date("Y-m-d"); // простое целое данное-член
```

```
var $sat = $myfirstname;
t
```

Данный пример вызовет ошибку и приведет к некорректной работе с классом.

В PHP 4 задавать значение переменной после инициализации строкой var можно только в том случае, если задаваемое значение является константой. Приведенный ниже пример не вызовет ошибки:

```
<?
class Cart {
var $items = 'Sasha';
var $name = 'Igor Poleshchuk';
var $stat = 45.8;
}
?>
```

Итак, классы могут нести в себе какую-либо информацию, образно их можно назвать знаниями. Для хранения этих знаний и служат данные-члены.

ВНИМАНИЕ

Переменная, как и объект, — это имя некой конструкции конкретного типа. С помощью этих терминов можно ссылаться на конкретные экземпляры определенных типов.

15.7. Методы класса

Методы — это то, что класс «умеет». Данные-члены — это только половина класса. Вторую половину класса образуют функции-члены, которые иногда называются методами. Объекты реального мира обычно классифицируются по тому, что они из себя представляют и что могут делать. Моделирование объектов цифрового мира заключается в представлении их статической стороны посредством переменных, а динамической — посредством функций. Совокупность этих аспектов образует классы.

Объявления функций в классах ничем не отличаются от объявлений вне классов: возвращаемый тип, за ним следует имя функции, список аргументов и завершается вся эта конструкция закрывающейся фигурной скобкой. Например:

```
class Constr
{
var $integer; // простое целое данное-член
var $name;
```

```
function my_account() // функция-член
{
....
}
}
```

Данный пример определяет класс Constr с двумя данными-членами:

```
var $integer;
var $name;
```

и одной функцией-членом:

```
function my_account () {};
```

При разработке классов первым делом следует выявить те данные и функции, которые наиболее полно выражают концепцию вашей программы.

15.8. Задания значений изменяющимся переменным в классах

Как было сказано ранее, в классе можно манипулировать переменными так, как если бы вы работали в основном блоке программы. Рассмотрим, как можно вносить значения изменяющимся переменным, т. е. переменным, которые не являются константами. В начале класса задавать таким переменным значения не принято, поэтому, как правило, используют функцию work (см. пример 15.1.).

Приведем пример неправильного задания значений изменяющимся переменным:

```
class Stock {
var $todays_date = date("Y-m-d");
var $secondname = $myname;
var $owner = 'Sasha ' . 'Igor';
}
```

Задание переменных таким способом будет не совсем верным в PHP 4. Программа объявит ошибку.

Пример 15.1. Использование функции work

```
class Stock {
var $todays_date;
var $secondname;
var $owner;
```

```
function work() {
$this->todays_date = date("Y-m-d");
$this->secondname = $GLOBALS['myname'];
/* и т. д. */
}
}
```

Выполнение этого примера позволит манипулировать со значениями переменных и при необходимости использовать их в программе. Это наиболее рациональный способ задания значения при помощи функции work.

Для задания значения массиву применяется следующий синтаксис (пример 15.2).

Пример 15.2. Задание значения массиву

```
<?
class Stock
{
var $sums;
function add_item($new, $num) {
$this->sums[$new] = $num + 5;
}
}
$class = new Stock;
$s = 0;
for ($i=0; $i<10; $i++)
{
$class->add_item($s++,$i);
print $class->sums[$s-1];
}
for ($i=0; $i<10; $i++)
print $class->sums[$i];
?>
```

Данная программа изменяет значения массива. Сначала задается класс Stock, после него — переменные, с которыми будет работать класс. Далее задаются функции, через которые будет происходить внесение значений для переменных класса из

нашей главной программы. После этого происходит выполнение программы, т. е. вычисления. Функция

```
• function add_item ($new, $num) {
  $this->sums[$new] = $num + 5;
  }
```

позволяет производить добавление значений в массив. При этом к каждому реальному значению нашего массива будет прибавляться число 5.

В пределах функции класса переменная $this означает «этот объект». Следует использовать $this -> переменная, чтобы обратиться к любой переменной или функции, имеющей то или иное название в пределах вашего текущего объекта. В случае, когда происходит обращение к свойствам объекта, не нужно использовать знак $.

Строка $class = new stock; предназначена для того, чтобы создать объект по имени $class нашего класса stock. Синтаксис его состоит из использования оператора new, который показывает, что нами создан объект $class класса stock.

Присвоение переменной $s = 0; происходит для того, чтобы в дальнейшем можно было изменять ее значения, не опасаясь, что она может принять какое-либо произвольное значение. Это очень важно при работе с численными переменными. Если бы мы не назначили переменной $s значение, равное 0, интерпретатор бы присвоил ей значение, равное 5 или, например, 10. Далее идет обычный оператор цикла, задающий логическую последовательность внесения данных в массив при помощи функции `add_item($new, $num);`.

Чтобы получить доступ к функции класса, используем ранее объявленный нами класс $class. Операция обращения к функции add_item() заложена в строке `$class->add_item($s++,$i);`, при этом переменная $s++ — инкремент суммирования, т. е. от нуля до 10 будет происходить прибавление значения. Например, при `$s = 0` значение массива будет равно 5. Обратите внимание, что в самой функции также происходит прибавление 5. Например:

```
$sums[1] = 5;
```

После этого идет еще одна очень важная строка: print `$class->sums[$s-1];`, которая позволяет выводить все значения массива в логическом порядке. В строке `$class->add_item($s++, $i);` после того как было произведено внесение значений переменных $s и $i, сразу произошло прибавление единицы к значению переменной $s. Чтобы это ликвидировать, производим вычитание и получаем необходимое нам значение. В результате выполнения программы на экран будет выведено:

```
5 6 7 8 9 10 11 12 13 14
```

Происходит увеличение каждого следующего значения на единицу, как и следовало ожидать.

Последние строки кода:

```
for ($i=0; $i<10; $i++)
print $class->sums[$i];
```

показывают, что значения самого массива после его задания сохраняются и теперь можно пользоваться ими в течение всей программы.

После того как вы внесли данные и произвели задание значений, вы можете также создать любую другую функцию для произведения каких-либо операций, например:

```
function remove_item ($new, $num) {
if ($this->sums[$new] > $num) {
$this->sums[$new] -= $num;
return true;
} else {
return false;
}
}
```

15.9. Расширение классов

В PHP предусмотрена такая важная вещь, как расширение классов, т. е. у вас есть уже один класс, который отвечает за те или иные действия, и вам срочно необходимо внести в него добавления. Для этого можно использовать возможности других классов.

Расширенный, или производный, класс имеет возможность доступа ко всем переменным и функциям основного класса плюс те данные, которые вы добавляете в новом классе. Синтаксис расширения основного класса заключается в присутствии ключевого слова extends. Обратите внимание, что множественное наследование (наследование от двух или более классов) не поддерживается. Приведем пример расширения классов:

```
class Add_Stock extends Stock {
var $ owner;
function set_owner ($name) {
$this->owner = $name;
}
}
```

Класс Add_stock имеет свое собственное имя, со своими переменными и функциями, но при этом он также имеет доступ к переменным класса Stock, определенно-

го нами ранее (см. пример 16.2), т. е. может использовать значения как из своего класса, так и из ранее определенного.

В PHP также существуют функции-конструкторы. функции-конструкторы в классе — это функции, которые автоматически объявляются, т. е. задаются определенными конкретными значениями, когда вы создаете новый образец класса. Функция становится конструктором, когда она имеет то же самое название, что и класс. Например:

```
class my_study extends Stock {
function my_study() {
$this->add_item("10", 1);
}
}
```

Данная функция имеет аналогичное название, что и класс. Это определяет класс my_study, который является расширением класса stock. В массив класса stock добавляется элемент под номером 10, равный 1, с учетом описанной нами ранее функции. Как правило, при использовании таких классов необходимо также пользоваться комментариями, так как это дает более полное понимание того, как работает этот класс.

Приведем еще способы представления классов-конструкторов:

```
class my_study extends Stock {
function my_study ($item= "10", $num = 1) {
$this->add_item($item, $num);
}
}
$default = new my_study;
$different = new my_study ("20", 17);
```

Как видите, в данном классе происходит задание значения в самой функции, что, конечно, очень удобно. Обратите внимание на строки:

```
1. $default = new my_study;
2. $different = new my_study ("40", 77);
```

Первая строка создает новый объект под названием $default, что ведет к обращению к классу с переменными, которые заданы в функции. Если же необходимость в использовании данных значений пропала и вам срочно необходимо внести свои, то можно это сделать следующей строкой, опять же при помощи объявления нового объекта — строки два. При этом переменным $item, $num будут присвоены значения 40 и 77 соответственно.

ВНИМАНИЕ

Для производных классов конструктор исходного класса автоматически не вызывается, если был вызван конструктор производного класса.

15.10. Работа с переменными класса

Рассмотрим способ задания переменных и ошибки, которые совершают при этом пользователи. Создадим класс:

```
class Named_univer extends Stock j
var $owner;
function set_owner($name) {
$this->owner = $name;
}
}
```

Данный класс является расширением класса Stock. Покажем, как происходит работа переменных, т. е. продемонстрируем синтаксис задания значений для этого класса:

```
$nstock = new Named_univer; // создание объекта с именем $nstock
$nstock->set_owner("kris"); // способ передачи значения массиву
// в данном классе
print $ncart->owner; // вывод значения переменной,
//·которая находится в самой функции // класса
$nstock->add_item("10", 1); // вызов конкретного значения класса Stock,
```

Данный пример показывает, каким образом происходит задание и получение значений переменных в класс и из класса.

Рассмотрим основные особенности при работе с переменными:

- чтобы производить обращение к переменнойкласса, например при задании ей значения, не нужно использовать знак доллара $:

```
$nstick->owner = "chris";
```

- если вы используете знак доллара, то это вызовет ошибку и будет подобно следующему:

```
$nstock->$owner = "chris";
```

Равносильно строке $nstick->$owner = $nstick->"", что в свою очередь и вызывает ошибку.

- если создается новая переменная с каким-то значением, а затем присваивается переменной объекта к другому значению переменной, то это также вызывает ошибку:

```
$myvar = 'owner';
$nstock->$myvar = "chris";
```

Данная операция оценивается интерпретатором как:

```
$nstock->$myvar = $nstock->owner
```

что также вызывает ошибку и является некорректным заданием значения переменной.

15.11. Манипуляция с именами

Имея дело с большими программами, легко запутаться в изобилии имен. Но несметного количества глобальных имен и функций можно просто избежать.

Благодаря маскировке деталей программист избавляется от необходимости непосредственно «общаться» с каждой переменной и функцией. Для маскировки деталей ипользуют инкапсуляцию — деление программы на классы, объединяющие данные и процедуры их обработки. Внутренние даные классы могут быть обработаны только предусмотренными для этого процедурами. Каждый такой класс имеет внутреннюю часть, называемую реализацией (или представлением), и внешнюю часть, называемую интерфейсом. Доступ к реализации возможен только через интерфейс. Таким образом, релазиация класса как бы заключена в капсулу и скрыта.

Используя инкапсуляцию функций и данных в классе, разработчик дает пользователю (под пользователями понимаются программисты, использующие этот класс) понять, что ему незачем знакомиться со всеми подробностями устройства класса, а достаточно только тех его частей, которые образуют открытый интерфейс.

О любой вещи можно составить логическое представление на разных уровнях понимания. Возьмите автомобиль: дети используют автомобиль как пассажиры, взрослые склонны рассматривать его как средство передвижения, автомобилестроители — как источник существования, а механики — тоже как источник существования, но только после поломки.

Уровень понимания автомобиля у всех этих категорий пользователей различен. Очевидно, что он зависит не от модели автомобиля, а от субъекта. Точно так же обстоит дело и с классами. Разработчик класса обязан знать его во всех подробностях, но пользователям класса достаточно понимания только тех аспектов, которые относятся непосредственно к управлению им. Если же вы являетесь потребителем класса, то для вас имеют значение не детали, а те его свойства, которые помогут вам в решении собственных задач. Например, конструкторы автомобилей предусматривают тормоза, поскольку знают, что иногда приходится останавливаться. Однако решать, когда жать на тормоз, приходится не им, а водителю.

Такое разделение полномочий, являющееся неотъемлемой стороной объектно-ориентированной философии, позволяет существенно сократить сложность проекта в целом.

15.12. Манипуляция уровнем сложности при работе с классами

Инкапсуляция небольших частей, вместе образующих цельную конструкцию, поможет вам как программисту держать под контролем уровень сложности проектов. С продуманными классами можно оперировать так же, как с элементарными типами данных посредствам тех же операций и столь же простых обозначений.

15.13. Указатель на самого себя $this

У каждого объекта есть физическое местоположение. Оперировать с местоположениями можно посредствам указателя. Как известно, в указателях хранятся адреса. Обращаясь к указателю, мы в действительности обращаемся к объекту, расположенному по содержащемуся в этом указателе адресу.

Каждый объект класса имеет свой физический адрес, его можно извлечь из указателя $this. Внутренний указатель $this есть у каждого класса. На самом деле мы всегда явно используем указатель $this, когда обращаемся к членам внутри области видимости функции-члена класса. Например:

```
class newclass
{
var $name;
function sum()
{
$this->name = date("Y-m-d");
}
}
```

СОВЕТ

Конечно, нет надобности использовать этот указатель для обращения к членам класса изнутри класса, но это единственное средство сослаться из объекта на объект *в* целом. Указатель $this иногда бывает крайне полезен.

15.14. Замечания по объектной терминологии

Как правило, в каждой отдельной специализированной отрасли существует своя специфическая терминология: у программистов своя, путешественников — другая, у физиков — третья. Так и в объектно-ориентированном мире одни и те же концепции часто описываются разными словами. Что касается терминологии PHP, то ее можно сравнить с терминологией языка Perl и C++.

Например, объекты часто называются экземплярами классов, а методы этих объектов — методами экземпляров. Поля данных, относящихся к каждому объекту, часто называются данными экземпляров или атрибутами объектов, а поля данных, общие для всех членов класса, — данными класса, атрибутами класса или статическими переменными класса.

Термины «базовый класс» и «суперкласс» описывают одно и то же понятие (родитель или другой предок в иерархии наследования), а термины «производный класс» и «субкласс» описывают противоположное отношение (непосредственный или отдаленный потомок в иерархии наследования).

Программисты на C++ используют статические методы, виртуальные методы и методы экземпляров, но PHP поддерживает только методы классов и методы объектов. В действительности в PHP существует только общее понятие «метод». Принадлежность метода к классу или объекту определяется исключительно контекстом использования. Метод класса (со строковым аргументом) можно вызвать для объекта (с аргументом-ссылкой), но вряд ли это приведет к разумному результату.

Программисты C++ привыкли к глобальным (т. е. существующим на уровне класса) конструкторам (constructor) и деструкторам (destructor). Конструкторы — это члены классов, используемые для создания объектов-экземпляров классов. Есть несколько разновидностей конструкторов, в их числе есть довольно своеобразные, но основное их назначение в любом случае одно и то же: обеспечение удобного спроса создания объекта-экземпляра класса. Деструкторы выполняют работу, обратную той, что проделывают конструкторы. Хотя класс может иметь несколько конструкторов, но деструктор может быть только один. В PHP конструкторы и деструкторы подчиняются тем же правилам.

С позиции C++ все методы PHP являются виртуальными. По этой причине их аргументы никогда не проверяются на соответствие прототипам функций, как это можно сделать для встроенных и пользовательских функций. Прототипы проверяются компилятором во время компиляции. Функция, вызванная методом, определятся лишь во время выполнения.

В своих объектно-ориентированных аспектах PHP предоставляет полную свободу выбора: возможность делать одни и те же вещи несколькими способами (приведение позволяет создать объект из данных любого типа), возможности модификации классов, написанных другими (добавление функций в их пакеты), а также полная возможность превратить отладку программы в сущий ад — если вам это сильно захочется.

В менее гибких языках программирования обычно устанавливаются более жесткие ограничения. Многие языки с фанатичным упорством отстаивают закрытость данных, проверку типов на стадии компиляции, сложные сигнатуры функций и другие возможности. Все эти возможности отсутствуют в объектах PHP, поскольку они вообще не поддерживаются PHP. Помните об этом, если объектно-ориентированные аспекты PHP покажутся вам странными. Все странности происходят лишь оттого, что вы привыкли к другим языкам. Объектно-ориентированная сторона PHP абсолютно разумна, если мыслить категориями PHP.

15.15. Ссылки внутри конструктора

Для более корректной работы с классами, т. е. с его данными класса, создатель PHP ввел такую привлекательную вещь, как ссылки внутри конструктора. Собственно говоря, не зная данных возможностей и не изучая этот параграф, вы не создадите себе какой-либо пробел в знаниях. Все, что можно сделать при помощи ссылок внутри конструктора, аналогично тем операциям, которые вы умеете делать без ссылок. Ссылки, как правило, позволяют манипулировать с самим адресом. Это очень важно понять, так как иногда происходят ошибки при выполнении программы, если разработчик не знает этого. Использование ссылок внутри конструктора может привести к запутанному результату. Это также одна из причин, почему они не получают особого распространения, хотя иногда применяются. Приведенный пример познакомит вас с синтаксисом применения ссылок:

```
<?
class simp {
function foo($name) {
// создание ссылки внутри global массива $globalref
global $globalref;
$globalref[] = &$this;
// назначение name, чтобы передать переменную
$this->setName($name);
// и вызов ее обратно
$this->echoName();
}
function echoName() {
echo "<br>",$this->Name;
}
function setName($name) {
$this->Name = $name;
}
}
?>
```

Данная программа весьма проста. Обратите внимание, что ссылка обозначается символом &. Приведем пример, который покажет принципы различия применения переменных и ссылок:

```
$num = new foo('constructor');
```

```
$num->echoName();
$globalref[0]->echoName();
```

В данном примере приведены переменные и способы обращения их к классу. Результат работы программы будет следующий:

```
constructor
constructor
constructor
```

Теперь создадим аналогичный фрагмент скрипта, но с использованием ссылки:

```
$num2 =& new foo('constructor');
$num2->echoName();
$globalref[1]->echoName();
```

Результат работы программы будет следующий:

```
constructor
constructor
constructor
```

Заметьте, что переменная $num и массив $globalref [0], вызываемый как переменная _NOT_, являются не одной и той же переменной. Это потому, что оператор «new» по умолчанию возвращает не ссылку, а копию.

ВНИМАНИЕ

В PHP 4 не существует никаких принципиально важных различий использования копирования переменных вместо ссылок (т. е. своего рода операции присвоения). Однако в большинстве случаев лучше работать с копиями переменных, чем со ссылками. Дело в том, что для того чтобы создать переменную, необходимо затратить некоторое время при обработке кода. А на копию переменных не затрачивается какое бы то ни было время. Исключение составляет работа с массивами или большими объектами при условии, что происходит изменение массива, а в последствии самого процесса и объекта. В этом случае самым рациональным решением является использование ссылок, что поможет производить изменения значений одновременно.

Приведем пример, иллюстрирующий все вышесказанное:

```
// сейчас мы изменим name и вы сможете увидеть,
// что $num и $globalref[0] изменили свои значения
$num->setName('set from outside');
// как было упомянуто ранее, дело обстоит не так
```

```
$num->echoName();
$globalref[0]->echoName();
/* результат :
set on object creation
set from outside */
// посмотрите на различия между $num2 и $globalref[1]
$num2->setName('set from outside');
// тут $num2->Name, а массив $globalref[1]->Name аналогично
$num2->echoName();
$globalref[1]->echoName();
```

В результате выполнения программы на экран будет выведено:

```
set from outside
set from outside •
```

15.16. Демонстрационная программа

Чтобы более подробно ознакомиться на практике с программированием классов, приведем пример работы с классами, а также полное описание последовательности выполнения:

```
<?
class a {
function a ($i) {
$this->value = $i;
$this->b = new b($this);
}
function createRef() {
$this->c = new b($this);
}
function echoValue() {
echo "<br>","class ",get_class($this), ': ',$this->value;
}
}
```

```
class b {
function b(&$a) {
$this->a = &$a;
}
function echoValue() {
echo "<br>","class",get_class($this), ' : ' ,$this->a->value;
}
}
$a =& new a(10);
$'a->createRef();
$a->echoValue();
$a->b->echoValue();
$a->c->echoValue();
$a->value = 11;
$a->echoValue();
$a->b->echoValue();
$a->c->echoValue();
$a->value = 14;
$a->echoValue();
$a->b->echoValue();
$a->c->echoValue();
?>
```

Данный скрипт состоит из двух **классов, класса** a и класса b. Для более детального ознакомления с программой рассмотрим каждую строчку класса a:

```
1: class a {
2: function a($i) {
3: $this->value = $i;
4: $this->b = new b($this);
5: }
6: function createRef() {
7: $this->c = new b($this);
```

```
8: }
9: function echoValue() {
10: echo "<br>","class ",get_class($this),': ',$this->value;
11: }
12:}
```

Первая строка необходима для создания самого класса, где и происходит задание имени класса и открытие основного контейнера (блока) класса. Вторая строка используется для задания функции в этом классе, также там задается то, что в эту функцию через переменную `$i` будет вводиться некоторое значение. Третья строка характеризует способ задания переменной value значения, переданного в функцию непосредственно при вызове последней, т. е. значение переменной `$i`. Для чего это нужно, вы увидите при выведении результата значений. Четвертая строка создает объект, который ссылается на класс по имени Ь. При этом тут идет процесс обработки самого адреса, не касаясь непосредственно значений. Шестая строка характеризует способ задания второй функции в этом классе, в которой будет создаваться также объект с, переводящий управление в класс ь. В девятой и десятой строках задается функция, которая позволяет выводить значения наших объектов и принадлежащие им переменные. Обратите внимание, в этой функции используется функция get_class (), не изученная нами ранее. Она позволяет возвращать имя класса используемого объекта, и тем самым вы сможете пронаблюдать, из какого класса были вызваны те или иные переменные.

Теперь рассмотрим класс по имени b:

```
1:class b {
2: function b(&$a) {
3: $this->a = &$a;
4: }
5: function echoValue() {
6: echo "<br>","class ",get_class($this),': ',$this->a->value;
7: }
8:}
```

Данный класс имеет имя b, о чем, собственно, и говорит первая строка. Вторая и третья строки задают функцию, которая необходима для того, чтобы манипулировать ссылками. Переменная, которая приходит в данную функцию, сразу ссылается на ее адрес, после чего и происходят дальнейшие операции выполнения данной программы. Пятая и шестая строки задают функцию, которая позволяет выводить значения переменных, с которыми мы работаем. Обратите внимание, что эта функция и функция, приведенная в классе а, имеют свои специфические отличия. Для чего они необходимы, вы увидите на примере выполнения данной программы.

Далее идет основной код программы, описание которого вы без какого-либо труда разберете сами, так как основные концепции работы с классами вам были изложены ранее.

Результатом данной программы будет следующий столбец данных:

```
class a: 10
class b: 10
class b: 10
class a: 11
class b: 11
class b: 11
class a: 14
class b: 14
class b: 14
```

Заключение

Эта глава познакомила вас с классами. Те, кто говорит, что для программирования на PHP не обязательно знать классы, будут правы, но вы тем самым обделите себя, лишившись массы мощных и выразительных возможностей. Из этой главы вы вынесли много информации: от основных аспектов до весьма тонких и неочевидных. В изучении материала вам должны помочь приведенные примеры, особенно последний, разберите его как следует, а если что-то будет непонятно, обратитесь к предыдующим главам.

Часть III

Особенности реализации языка

Глава 16

Обработка ошибок

Данная глава написана для того, чтобы вы смогли понять систему определения ошибок в PHP. Прежде всего познакомимся с основными типами ошибок, а также как с ними бороться. Не зная эту главу, вы без труда сможете освоить язык, но в случае возникновения каких-либо неприятностей обращайтесь к этой главе. В ней рассматриваются следующие темы:

- типы ошибок и предупреждений об ошибках;
- подавление ошибок при обращении к функциям.

16.1. Типы ошибок и предупреждений об ошибках

В PHP имеется несколько типов ошибок и предупреждений об ошибках (табл. 16.1).

Таблица 16.1. Типы ошибок PHP

Значение	Константа	Описание	Примечание
1	E_ERROR	Неустранимые ошибки на стадии выполнения	
2	E_WARNING	Предупреждения на стадии выполнения (несущественные ошибки)	
4	E_PARSE	Ошибки анализа на стадии компиляции	

Окончание табл. 16.1

Значение	Константа	Описание	Примечание
8	E_NOTICE	Замечания на стадии выполнения (менее серьезные, чем предупреждения)	
16	E_CORE_ERROR	Неустранимые ошибки, происходящие во время начального запуска PHP	Только в PHP 4
	E_CORE_WARNING	Предупреждения (несущественные ошибки), происходящие во время начального запуска PHP	То же
64	E_COMPILE_ERROR	Неустранимые ошибки на стадии компиляции	«
	E_COMPILE WARNING	Предупреждения на стадии компиляции (несущественные ошибки)	«
256	E_USER_ERROR	Сообщение об ошибке, генерируемое пользователем	«
512	E_USER_WARNING	Предупреждение, генерируемое пользователем	«
1024	E_USER_NOTICE	Замечание, генерируемое пользователем	«
	E_ALL	Все вышеупомянутое в зависимости от поддержки	

Перечисленные выше значения (числовые или символьные) используются для построения битовой маски, определяющей ошибку, о которой выдается сообщение. Для комбинации значений или определенных типов ошибок можно использовать побитовые операторы. Но обратите внимание на то, что в php.ini будут восприниматься только |, ~, ! и &, а в php3.ini побитовые операторы восприниматься не будут.

В PHP 4 значением по умолчанию для параметра error_reporting является E_ALL & ~E_NOTICE, что означает отображение всех сообщений об ошибках и предупреждений, не имеющих уровня E_NOTICE. В PHP 3 по умолчанию используется значение E_ERROR | E_WARNING | E_PARSE, что означает то же самое. Однако обратите внимание, что, поскольку в php3.ini в PHP 3 константы не поддерживаются, установка error_reporting должна быть числовой (например, 7).

Начальную установку можно изменить в файле ini с помощью директивы error_reporting в файле httpd.conf Apache с помощью директивы php_error_reporting (php3_error_reporting для PHP 3), атакже во время выполнения в скрипте с помощью функции error_reporting().

ВНИМАНИЕ

При обновлении кода или серверов с PHP 3 на PHP 4 следует проверить эти установки и вызовы error_reporting(), иначе можно отключить сообщение об ошибках новых типов, особенно E_COMPILE_ERROR. Это может привести к выдаче пустых документов вместо сообщения о происходящем или информации о том, где нужно искать проблему.

Все выражения PHP могут также вызываться с префиксом @, отключающим сообщения об ошибках для данного конкретного выражения. Если к такому выражении происходит ошибка, и функция track_errors активна, сообщение об ошибке можно будет найти в глобальной переменной $php_errormsg.

Префикс @ оператора контроля ошибок не отключает сообщения, возникающие в результате синтаксического анализа.

ВНИМАНИЕ

Префикс оператора контроля ошибок @ отключит даже сообщения о критических ошибках, прерывающих выполнение скрипта. Помимо всего прочего это означает, что если вы используете оператор @ для отключения сообщений об ошибках в определенной функции, а функция недоступна или набрана с ошибкой, скрипт сразу же закончит работу, не указав причины остановки.

Рассмотрим пример использования возможностей обработки ошибок в PHP. Определим функцию для обработки ошибок, которая будет протоколировать информацию в файл (используя формат XML) и отправлять его разработчику по электронной почте в случае критической ошибки в логике.

Пример 16.1. Обработка ошибок в скрипте

```
<?php
// будем сами обрабатывать ошибки
error_reporting(0);
// определенная пользователем функция обработки ошибок
function userErrorHandler($errno, $errmsg, $filename, $linenum,
$vars) {
// дата и время записи об ошибке
$dt = date("Y-m-d H:i:s (T)");
// определение ассоциативного массива строк ошибок
// на самом деле следует рассматривать
// только элементы 2, 8, 256, 512 и 1024
$errortype = array (
```

```
    1 => "Ошибка",
    2 => "Предупреждение",
    4 => "Ошибка синтаксического анализа",
    8 => "Замечание",
    16 => "Ошибка ядра",
    32 => "Предупреждение ядра",
    64 => "Ошибка компиляции",
    128 => "Предупреждение компиляции",
    256 => "Ошибка пользователя",
    512 => "Предупреждение пользователя",
    1024=> "Замечание пользователя"
);
// набор ошибок, для которого будут сохраняться значения переменных
$user_errors = array(E_USER_ERROR, E_USER_WARNING, E_USER_NOTICE);
$err = "<errorentry>\n";
$err .= "\t<datetime>".$dt."</datetime>\n";
$err .= "\t<errornum>".$errno."</errnumber>\n";
$err .= "\t<errortype>".$errortype[$errno]."</errortype>\n";
$err .= "\t<errormsg>".$errmsg."</errormsg>\n";
$err .= "\t<scriptname>".$filename."</scriptname>\n";
$err .= "\t<scriptlinenum>".$linenum."</scriptlinenum>\n";
if (in_array($errno, $user_errors))
$err .= "\t<vartrace>".wddx_serialize_value($vars,"Variables")."</
vartrace>\n";
$err .= "</errorentry>\n\n";
// для тестирования
// echo $err;
// сохранить протокол ошибок и отправить его мне,
// если есть критические ошибки
error_log($err, 3, "/usr/local/php4/error.log");
if ($errno == E_USER_ERROR)
mail("phpdev@mydomain.com", "Critical User Error",$err);
```

```
}
function distance($vect1, $vect2) {
if (!is_array($vect1) || !is_array($vect2)) {
trigger_error("Incorrect parameters, arrays expected", E_USER_ERROR);
return NULL;
}
if (count($vect1) != count($vect2)) {
trigger_error("Vectors need to be of the same size", E_USER_ERROR);
return NULL;
}
for ($i=0; $i<count($vect1); $i++) {
$c1 = $vect1[$i]; $c2 = $vect2[$i];
$d = 0.0;
if (!is_numeric($c1)) {
trigger_error("Coordinate $i in vector 1 is not a number, using
zero",
E_USER_WARNING);
$c1 = 0.0;
}
if (!is_numeric($c2)) {
trigger_error("Coordinate $i in vector 2 is not a number, using
zero",
E_USER_WARNING);
$c2 = 0.0;
}
$d += $c2*$c2 - $c1*$c1;
}
return sqrt($d);
}
$old_error_handler = set_error_handler("userErrorHandler");
// неопределенная константа генерирует предупреждение
$t = I_AM_NOT_DEFINED;
```

```
// определение некоторых "векторов"
$a = array(2,3,"foo");
$b = array(5.5, 4.3, -1.6);
$c = array (1,-3);
// генерация пользовательской ошибки
$t1 = distance($c,$b)."\n";
// генерация другой пользовательской ошибки
$t2 = distance($b,"i am not an array")."\n";
// генерация предупреждения
$t3 = distance($a,$b)."\n";
?>
```

Это всего лишь простой пример, демонстрирующий использование функций обработки и протоколирования ошибок.

16.2. Подавление ошибок при обращении к функциям

Иногда желательно игнорировать фатальные ошибки, о которых могут сообщать специфические функции PHP. Например, вы захотите игнорировать ошибки от вызова функции dbmopen() и просто проверять возвращаемое значение обращения без того, чтобы сообщение об ошибке появлялось на экране. Это можно сделать, помещая символ @ перед именем функции.

```
$err_code = @dbmopen($filename,"w");
```

Реальное сообщение об ошибке, которое должно было быть выведено, может быть проверено во внутренней переменной PHP $phperrmsg.

Более общий подход для подавления вывода сообщений об ошибках — это использование функции SetErrorReporting(). С помощью этой функции вывод сообщений об ошибках может быть заблокирован для всех частей программы, например:

```
SetErrorReporting(0);
```

Это выключает все сообщения об ошибках. Их можно затем разрешить снова с помощью вызова:

```
SetErrorReporting(1);
```

Заключение

Если возникает ошибка, интерпретатор PHP автоматически выводит место с кратким описанием и указанием причины ее возникновения. Как правило, ошибки бывают разного плана, но наиболее частые мы постараемся выделять в наших следующих главах.

Глава 17
Идентификация в PHP

С появлением любой программы сразу возникает множество проблем, в том числе и связанных с авторизацией, т. е. правилами доступа для того или иного лица. Для чего это создается и что преследуют программисты и разработчики разных способов защиты, вам объяснять не надо. Каждая система имеет множество способов защиты и позволяет ограничить доступ нежелательных лиц. Во всех языках программирования эта проблема является неизбежной, и язык займет лидирующее место только в том случае, если именно у него будут присутствовать функции работы с ограничением доступа пользователей. Что касается языков программирования для Сети, то тут комментарии вообще являются излишними. Рассматриваемый нами язык программирования PHP имеет определенный перечень способов авторизации (регистрации в базе данных) пользователей. Это нужно для:

- ограничения доступа к вашему сайту;
- возможности вести учет посещаемости тем или иным пользователем;
- качественной регистрации;
- способа организации доступа на чаты, форумы и т. п.

Этот список может еще продолжаться и продолжаться, потому что задача, для которой применяют авторизацию посетителя, отвечает запросам самого программиста. Поэтому наша цель в этой главе показать, как более профессионально можно осуществить процесс авторизации посетителя, а в каком месте ее применить, вы разберетесь сами. В этой главе вы изучите следующие вопросы:

- функции HTTP (HyperText Transfer Protocol);
- основные концепции при программировании авторизации;
- авторизация посетителя сайта.

17.1. Функции HTTP (HyperText Transfer Protocol)

Функции HTTP необходимы для работы с HTTP-протоколом. Эти функции позволяют управлять выходом, посланным назад удаленному браузеру вплоть до уровня протоколов HTTP. Как правило, эти функции применяются при авторизации посетителей. Как это осуществляется, вы узнаете немного позже, сейчас рассмотрим процесс работы данных функций.

ВНИМАНИЕ

Данные функции будут работать только в том случае, если ваш PHP установлен как модуль. В противном случае могут возникнуть ошибки и процесс выполнения вашей программы будет некорректен.

Функции HTTP включают в себя:

- header(),
- headers_send(),
- setcookie().

В данном параграфе мы рассмотрим только две первые функции, так как setcookie() была рассмотрена нами ранее (см. п. 9.8). Наиболее распространенной из них является header().

Функция header() используется в начале HTML-файла, это необходимо учесть, так как данная функция посылает необработанные строки, возглавляемые протоколом HTTP. В случае возникновения какой-либо необходимости во время ошибки или же в процессе работы скрипта может произойти выполнение данной функции. Для более подробного ознакомления с необработанными HTTP-описаниями можно познакомиться, изучив спецификацию HTTP 1.1 по адресу www.w3.org. Мы же покажем вам способы применения данной функции, чтобы в последующем вам было намного легче ознакомиться с процессом программирования скриптов, выполняющих процесс авторизации посетителя. Синтаксис данной функции следующий:

```
int header(string string)
```

Она возвращает целое число. В поля string string заносятся определенные значения, зависящие прежде всего от поставленной задачи. Данная функция используется в PHP 3 и PHP 4.

В основном существует два запроса возглавляемого вызова, первый из которых Location. Этот запрос позволяет посылать заглавие назад браузеру, а также возвращает переназначающийся код состояния Apache. С точки зрения записи сценария это не важно, но программистам, которые разбираются во внутренней организации Apache, необходимо это знать. Ниже приведен пример кода:

```
header("Location: http://www.any_domain_name.com");
```

Второй случай — любой подзаголовок, который начинается со строки HTTP (регистр не существен). Например, если бы у вас была ошибка в документе следующего содержания: ErrorDocument 404 Apache, было бы неплохо удостовериться, что ваш сценарий PHP фактически генерирует ошибку 404. Для этого первая операция, которую вам необходимо сделать, должна быть такой:

```
header("HTTP/1.0 404 Not Found");
```

Таким образом, можно работать со всеми возможными запросами данной функции. В этом вам и поможет спецификация HTTP 1.1.

Часто PHP-скрипты создают динамический HTML, который не должен кэшироваться браузером клиента или какими-либо прокси-кэшами между браузером клиента и сервером. Чтобы прокси-сервер и клиенты не квалифицировали HTML-страницы, можно воспользоваться одним из следующих примеров:

```
1. header("Expires: Mon, 28 Jul 1995 08:00:00 GMT");
2. header("Last-Modified: ". gmdate("D, d M Y H:i:s"). " GMT");
3. header("Cache-Control: no-cache, must-revalidate");
4. header("Pragma: no-cache");
```

Функция под номером 1 — дата в прошлом, 2 — производит модификацию даты, 3 — используется в HTTP 1.1 и 4 — применяется в HTTP 1.0.

Чтобы более полно ознакомиться с принципами работы данных функций, попробуйте поэкспериментировать самостоятельно и посмотреть, что из этого получится.

Помните, что функция header () должна быть вызвана раньше, чем какие-либо данные нашего скрипта будут посланы, при условии, что не используется буферизация. Это необходимо, чтобы можно было дать понять нашему интерпретатору, что существует еще одна операция, которую ему необходимо будет выполнить в случае некорректного выполнения скрипта. Данное действие — одна из распространенных ошибок.

В ходе выполнения такого сценария, как только указатель дойдет до строки с функцией require (), произойдет переадресация и процесс выполнения программы дальше будет прерван. Наиболее верно будет следующее расположение:

```
<?php require("logins.inc") ?>
<?php header("Content-Type: audio/x-pn-realaudio"); ?>
```

Также данная функция поддерживает способы авторизации:

```
header("www - Authenticate: Basic realm = \"$realm"\"");
```

где $realm — область действия авторизации. Более подробно с принципами работы данной функции в области авторизации вы познакомитесь немного позже.

Также обратите внимание на функцию headers_sent (). Она является очень простой в обращении и имеет следующий синтаксис:

```
boolean headers_sent (void)
```

Как видно, функция возвращает значение типа boolean и используется в следующих версиях PHP:

- PHP 3.0.8 и выше;
- PHP 4.0b2 и выше.

Данная функция позволяет контролировать процесс функции header (), т. е. в случае, когда происходит посылка главных значений функции header (), функция headers_sent () возвращает значение true, в противном случае — false.

Ответ на вопрос, чем полезны данные функции, каждый сможет дать себе сам, как только разберется с принципами их работы. Мы же в свою очередь перейдем к более обширному и систематизированному применению данных функций, в частности при авторизации пользователей.

17.2. Основные концепции при программировании авторизации

Прежде всего при осуществлении авторизации необходимо, чтобы PHP был установлен как модуль Apache, а не CGI или IIS. Дело в том, что только в этом случае происходит доступ к константам $PHP_AUTH_USER, $PHP_AUTH_PW и $PHP_AUTH_TYPE, которые используются в процессе авторизации (при их помощи получают значения логина, пароля и типа). Функция header() производит вызов окна, в которое и будет заноситься (вводиться) наш пароль и логин. После того как вы введете эти значения и нажмете на кнопку «OK», ваши данные будут переданы следующим переменным. Значение, которое вы ввели в поле пароля, будет передано зарезервированной константе $PHP_AUTH_PW, и соответственно значение, которое вы ввели в поле логина, будет передано константе $PHP_AUTH_USER. Далее весь процесс осуществления авторизации вы можете вести, выполняя элементарные действия программирования, вплоть до вывода имеющегося пароля. В частности, можно сравнить введенные данные с имеющимися, и если они верны, разрешить доступ к скрытым данным вашего сайта. В противном случае предложить пользователю либо зарегистрироваться, либо связаться с администратором для обсуждения условий регистрации.

Например, вам необходимо ограничить доступ к одной из страниц, т. е. вы бы хотели позволить получать данные этой страницы только избранным пользователям. В этом случае можно воспользоваться следующим кодом:

```
if ($PHP_AUTH_USER!=$name || $PHP_AUTH_PW!=$pass) {
header("WWW-Authenticate: Basic realm=\"secret area\"");
header("HTTP/1.0 401 Unauthorized");
echo "Authorization Required.";
exit;
}
```

Переменные $name и $pass и будут содержать именно то верное значение пароля, которое необходимо ввести пользователю при получении равноправного доступа к документу. Данный код должен идти до любого вывода данных в браузер, т. е. прежде чем вы начнете выполнять какие либо операции, вам необходимо вставить данный код в начало страницы. Что касается условия в самой первой строке, то оно может быть любым, вплоть до проверки существования такого пароля и имени в вашей базе данных. Если у вас появилась необходимость ограничить доступ по паролю к нескольким страницам, просто можете вставить этот код в каждую страницу.

Приведем пример, позволяющий выводить ваш пароль и логин при регистрации и предоставлении всех имеющихся данных пользователю в момент проверки переданных данных:

```
<?php
if(!isset($PHP_AUTH_USER)) {
```

```
header ("WWW-Authenticate: Basic realm=\"$realm\"") ;
header ("HTTP/1.0 401 Unauthorized");
echo "Вам отказано в доступе\n";
exit;
} else {
echo "Доброго времени суток $PHP_AUTH_USER.<P>";
echo "Пароль, который вы ввели, является $PHP_AUTH_PW<P>";
}
?>
```

Данный пример работает очень просто. Если этот скрипт расположен в начале файла, происходит вызов функции header (), в ходе выполнения которой на экране вашего монитора появиться диалоговое окно, в котором вам будет предлагаться ввести пароль. Если вы нажмете на кнопку с надписью «Cancel», будет выполнена строка программы echo "Вам отказано в доступе\n";, что, конечно же, приведет к запрету в доступе к документу. В противном случае вы благополучно введете ваши логин и пароль, этим самым передадите после нажатия кнопки переменным $PHP_AUTH_USER $PHP_AUTH_PWзначения логина и пароля соответственно. Сам скрипт сможет производить работу с этими данными, в нашем же случае он просто произведет вывод данных на экран браузера. Для чего это может быть полезно, объяснять не стоит. Теперь вы без проблем сможете работать с паролем и логином.

Приведем код программы, который является рациональным в использовании PHP как модуля Apache, а также и в случае CGI. Текст скрипта содержится в файле testpwd.php:

```
<?
if (!session_is_registered("pwd"))session_register("pwd");
if (!session_is_registered("usr"))session_register("usr");
if (!session_is_registered("go")){session_register("go");}
if (php_sapi_name()== "cgi")
{
if ( ($usr_cgi!=$GLOBALS['user'] || $pwd_cgi!=$GLOBALS['password'])
&& $go!=789)
{
?>
<table><form method=GET>
<?if, (isset($usr_cgi) || isset($pwd_cgi)) echo "<tr><td colspan=2>
```

```
<font color=red>Введите правильный пароль/имя пользователя</
font></td></tr>"?>
- <tr><td>Username please</td><td><input name=usr_cgi size="7"><br></
tdx/tr>
<tr><td>Password please</td><td><input type =password name=pwd_cgi
size="7"><br></td></tr>
<tr><td><input type="submit" value=Ввести></td></tr>
</form>
</table>
<?;exit();
}
}
else
if(!isset($PHP_AUTH_USER)) { // эта переменная передается,
// только если введен
// пароль и имя в окошке, которое вызывается специальным
// заголовком. Эта часть передает специальный заголовок
Header("WWW-Authenticate: Basic realm=\"Введите ваш пароль\"");
Header("HTTP/1.0 401 Unauthorized");
$GLOBALS['title']="AutentificationError!";
require_once("./_top.php");
echo "
<p class=\"b\">Вы нажали Cancel! </p>";
require_once("./_bot.php");
exit;
} else { // здесь будет ваш код по проверке пароля...
if ($PHP_AUTH_USER!=$GLOBALS['user'] ||
$PHP_AUTH_PW!=$GLOBALS['password'])
{
Header("WWW-Authenticate: Basic realm=\"Введите ваш пароль\"");
$GLOBALS['title']="AutentificationError!";
require_once("./_top.php");
```

```
echo "<p class=\"b\">Error! </p>";
require_once("./_bot.php");
exit;
}
}
$go=789;
?>
```

Следующий пример иллюстрирует включение скрипта для введения системы аутентификации в вашу программу:

```
<?
// пароль и имя пользователя. Если у вас есть группа скриптов
// и вы хотите только один раз вводить пароль и имя пользователя
// для доступа к ним, то следующую строку запишите в отдельный
// файл, после чего включите этот файл во все ваши скрипты:
require_once("user.php");
$GLOBALS['user']="server";$GLOBALS['password']="pwd";
require_once("./testpwd.php"); //< --- аутентифицирующий скрипт
echo "Here is a secret information ";
?>
```

Обязательно разберитесь с принципами работы данного скрипта.

Вместо простого распечатывания `$PHP_AUTH_USER` и `$PHP_AUTH_PW` вы, вероятно, хотели бы проверить имя пользователя и пароль. Это можно сделать, посылая запрос к базе данных или ища пользователя в файле dbm.

Не перепутайте порядок функций `header()`. Наиболее правильный порядок формирования запроса функции `header()` таков: WWW-Authenticate раньше, чем `HTTP/1.0 4.01`, так как браузер Internet Explorer чувствителен к этому порядку.

Как Netscape Navigator, так и Internet Explorer способны производить очистку локального окна программы просмотра опознавательного кэша, т. е. другими словами, после того как интерпретатор PHP получит ответ 401, он произведет очистку кэша, где будут сохранены значения пароля и имени, введенные пользователем. Это помогает эффективно выходить из системы.

```
<?php
function authenticate() {
header( "WWW-authenticate: basic realm="$realm"") ;
```

```
header ( "HTTP/1.0 401 Unauthorized");
echo "Для того, чтобы получить доступ к данному ресурсу, вам
необходимо ввести логин и пароль\n";
exit;
}
if(!isset($PHP_AUTH_USER) || ($SeenBefore == 1 &&
!strcmp($OldAuth, $PHP_AUTH_USER)) ) {
authenticate();
}
else {
echo "Welcome: $PHP_AUTH_USER<br>";
echo "Old: $OldAuth";
echo "<form action=\"$PHP_SELF\" method=post>\n";
echo "<input type=hidden NAME=\"SeenBefore\" value=\"1\">\n";
echo "<INPUT TYPE=HIDDEN NAME=\"OldAuth\"
value=\"$PHP_AUTH_USER\">\n";
echo "<input type=Submit value=\"Re Authenticate\">\n";
echo "</form>\n";
}
?>
```

Испытание операционной системы Lynux показало, что Lynux не очищает данные при ответе станции 401. В этом случае пользователи могут нажимать клавишу «_», чтобы очистить их опознавательную информацию.

ВНИМАНИЕ

Описанное выше не работает при использовании IIS Microsoft Server и CGI-версии PHP из-за ограничения IIS.

17.3. Авторизация посетителей сайта

Как правило, авторизация применяется на сайте прежде всего для ограничения доступа пользователей. Это обычно происходит, если вы имеете какую-либо коммерческую информацию, за доступ к которой необходимо платить, или же у вас возникла острая необходимость в использовании пароля для ограничения доступа. В общем, причин для применения авторизации может быть множество, впрочем, как и способов реализации этой задачи. Вопрос авторизации в PHP решается

элементарно, как и было сказано ранее. Возможно, что PHP в этом отношении является наиболее удобным языком программирования для начинающих. При полученных знаниях вы также сможете реализовать любую поставленную задачу, связанную с авторизацией. Приведем фрагмент кода для реализации авторизации на практике. Данную программу после проверки на корректность работы вы спокойно можете внедрять в нужные вам файлы, дабы ограничить доступ к последним. Время использования этого кода может быть любым вплоть до года и более. Все зависит прежде всего от вас и программной необходимости. Например:

```
<?
If (!$auth_inc):
$auth_inc = 1;

function auth_Check($realm, $sorry = ''$tmp1= '',$tmp2 = '')
{
global $PHP_AUTH_USER;
global $PHP_AUTH_PW;
if (!isset($PHP_AUTH_USER))
{
Header("WWW - Authenticate: Basic realm = \"$realm\"");
Header("HTTP/1.0 401 Unauthorized");
if ($sorry = = '' )
echo"Вам отказано в доступе!";
else
{
include($sorry);
}
exit;
}
else
{
if (@authFunction($PHP_AUTH_USER, $PHP_AUTH_PW, $tmp1, $tmp2))
{
$0k = 1;
```

```
}
else
{
header("WWW-Authenticate: Basic realm = \"$real,\"");
header("HTTP/1.0 401 Unauthorized");
if ($sorry = = '')
echo "Вам отказано в доступе";
else
{
include($sorry);
}
exit;
}}}
endif;
?>
```

Функцию authFunction() пишет сам пользователь, она получает аргументами, соответственно, логин, пароль, введенные посетителем, и два необязательных аргумента по наследству от authFunction(), которая, например, будет проверять пользователя по базе данных и вызывать функцию authCheck() перед каким-либо выводом текста страницы, как было замечено нами в параграфе про функции setcookie() (см. п. 9.8). Написание разных authFunction() позволяет контролировать доступ посетителей из множества различных источников, таких как файлы паролей, **СУБД,** хэш-файлы, запросы к сетевым сервисам авторизации и др.

Такой тип авторизации самый надежный. При посещении закрытой страницы посетителю выдается диалоговое всплывающее окно с полями имени и пароля. В дальнейшем браузер запоминает $realm и при последующем запросе пароля будет автоматически отправлять серверу пару имя/пароль. Однако это не единственный способ авторизации. Описанную выше систему ведения пользователя вполне можно использоваться в работе, внеся небольшие модификации. Для регистрации создадим форму с полями login и password. Обработчик форм генерирует случайный уникальный ключ и записывает с ним дату/время регистрации в хэш-файл. Вместо даты и времени годится значение функции time(), возвращающей количество секунд с точки отсчета, равной 01.01.1970. Браузеру клиента выдается cookie, например xfile со значением только что сгенерированного ключа.

Каждая страница, требующая авторизации, должна проверить наличие ключа $xfile в хэш-файле и, если он есть, записать туда время обращения, не затрагивая

время регистрации. Для отказа от авторизации достаточно удалить ключ из базы. Внешняя программа может один раз в полчаса удалять ключи, выданные три часа назад, и ключи, к которым не было обращения в течение одного часа. Стойкость такой системы определяется трудностью подбора ключа cookies за ограниченное время, так как ключи существуют относительно не долго и они достаточно длинные. Ключи можно передавать не только через Cookies, но и через URL как параметр скрипта.

Автор сайта должен сам решить, какой алгоритм авторизации сайта он себе выберет.

Введение авторизации — это ключ к созданию электронного магазина, электронной почты, чатов, форумов и т. д.

Заключение

Данная глава позволила вам решать поставленные задачи в области авторизации, а также определить места, где используется авторизация. Также хотелось бы заметить, что в том случае, когда у вас есть выбор между сложностью описываемой задачи, стремитесь всегда ктому, чтобы программы, которые вы создаете, были как можно проще. Использование авторизации должно быть целенаправленным. При применении авторизации помните основные концепции, необходимые для полноценного выполнения скрипта:

- PHP должен быть установлен как модуль;
- должна быть соблюдена последовательность использования функцийheader():

```
header("WWW-Authenticate: Basic realm=\"$realm\"");
Header("HTTP/1.0 401 Unauthorized");
```

- переменные, которым передаются введенный вами пароль и логин, имеют следующие имена: $PHP_AUTH_PW, $PHP_AUTH_USER соответственно. Обработку полученных значений вы можете производить на свое усмотрение.

Как видно из всего вышеописанного, PHP является очень удобным языком программирования в плане авторизации и создания Web-приложений. Позже вы сможете ознакомиться с основными концепциями наиболее распространенных функцией. Данная глава ввела вас в проблему понимания задач программирования приложений и позволила рассмотреть их с точки зрения авторизации и возможностей ограничения доступа к той или иной странице нашего документа. Наиболее полно эту проблему вы сможете освоить только в том случае, если все вышеописанное вы проделаете на примерах. При этом недостаточно использования приведенных нами, важно ваше понимание и ориентация в самостоятельном программировании. Именно это позволит вам свободно овладеть данной областью. Не бойтесь экспериментировать.

Глава 18
Загрузка файлов по HTTP

Для чего вообще нужна загрузка файлов по HTTP? На самом деле, это очень удобно как для администратора, так и для пользователя. Дело в том, что не всегда есть возможность загружать файлы на сервер по протоколу FTP (как это делается в большинстве случаев). А вот послать через браузер по протоколу HTTP крайне удобно.

Данная глава содержит темы:

- пример формы ввода;
- скрипты для обработки принимаемых данных;
- возможные трудности.

18.1. Пример формы ввода

Представим, что у вас есть свой сайт, где вы бы хотели реализовать, например, загрузку фотографий посетителей вашего ресурса. В данном случае очень удобно воспользоваться именно технологией загрузки файлов по HTTP. Схема проста: посетитель выбирает нужный файл на своем жестком диске и нажимает кнопку «Отправить». Судите сами, что проще: использовать FTP или такую технологию. Нам кажется, выбор очевиден. Приведем пример такой формы:

```
<form method="post" enctype="multipart/form-data" action="">
<input size=40 type=file name=myfile1><br>
<input type=submit value="Отправить">
</form>
```

Результат выполнения программы представлен на рис. 18.1.

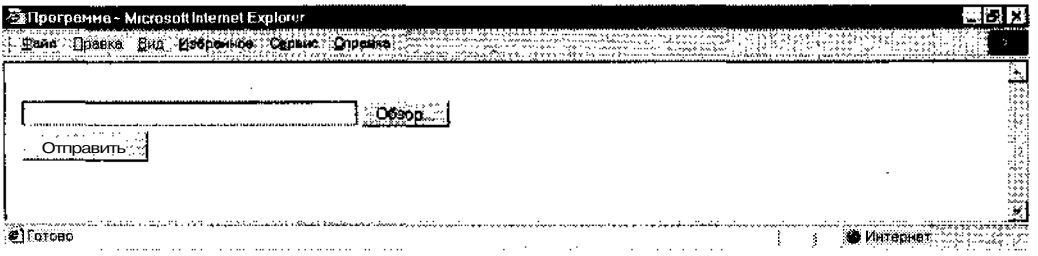


Рис. 18.1. Полученная форма

После того как посетитель нажмет кнопку «Отправить», файл передастся на сервер, где и считывается скриптом. После этого файл будет переименован (получит наиболее удобное для вас имя) и определен в тот каталог, который задаст непосредственно Web-мастер сайта. Что делать с полученным файлом — это уже ваше дело.

Теперь подробно разберемся, что это за форма ввода и какие у нее есть особенности. Обратим внимание на формат тега <form>. Это обычная форма, но существует несколько условий его использования:

1. Передача параметров скрипту должна осуществляться методом Post, а не методом Get, который используется по умолчанию в том случае, если метод передачи данных (параметр method) не был определен для формы Web-мастером.

2. Очень важно четко определить content-type, с которым данные и будут передаваться на сервер. Это делается с помощью параметра enctype, который должен иметь значение multipart/form-data.

ВНИМАНИЕ

Обычно значение enctype другое, но если в форме есть элемент ввода с типом file, нужно использовать именно этот тип данных.

Итак, мы уже знаем, как будет выглядеть наш тег <form>:

```
<form method="post" enctype="multipart/form-data" action="/cgi-
bin/upload.cgi">
```

В вышеприведенном теге присутствует привычный для большинства форм параметр action, который определяет, куда будут переданы результаты заполнения формы. В нашем случае это скрипт upload.cgi.

ВНИМАНИЕ

Имя скрипта и место его размещения может быть абсолютно произвольным. В данном случае расположение скрипта указано исключительно для примера.

Следующий шаг — это размещение стандартных элементов для нашей формы.

1. Элемент выбора файла:

```
<input type=file name=myfile1>
```

Ключ type с параметром file указывает на то, что в результирующем HTML-документе, который увидит посетитель сервера, будет форма ввода имени файла, а рядом кнопка, при нажатии на которую браузер откроет интерфейс выбора файла на жестком диске пользователя.

Параметр name=myfile1 указывает на то, что двоичные данные, из которых состоит файл, будут переданы скрипту в параметре myfile1. Именно этот параметр мы будем использовать, когда будем рассматривать процедуру принятия файла скриптом.

2. Кнопка.

Чтобы отобразить в HTML-документе кнопку, при нажатии на которую начнется передача данных из формы, нужно добавить в HTML-код такую строку:

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

После всего этого необходимо закрыть форму тегом </form>. Теперь наша форма готова к полноценному выполнению поставленной задачи.

18.2. Скрипты для обработки принимаемых данных

Итак, посетитель сайта выбрал файл на своем диске и нажал на кнопку «Отправить». Теперь наша задача — обработать данные, переданные формой, и сделать что-нибудь с передаваемым файлом. Для этого воспользуемся скриптом на PHP.

В PHP загрузка файлов осуществляется гораздо проще, нежели при помощи скриптов, написанных на другом языке, например на Perl. Все дело в том, что в PHP есть встроенный модуль, который и отвечает за загрузку. Конечно, можно поступить и по аналогии с Perl: открывать входной поток данных, вычислять имя, получать и писать данные в двоичном режиме и т. д. Однако гораздо удобнее воспользоваться встроенным механизмом.

Как вы помните, в созданной нами форме переменная, которая передает имя файла, названа `myfile1`. Если в качестве аргумента к параметру action в форме задан PHP-скрипт, то в самом скрипте после передачи в него методом Post данных из формы будут предопределены следующие переменные:

`$myfile1` — имя (полное, с путем) временного файла, под которым были сохранены загруженные на сервер данные;

`$myfile1_name` — имя оригинального файла, под которым данные были у пользователя на диске;

`$myfile1_size` — размер файла, который был загружен на сервер;

`$myfile1_type` — MIME-тип файла пользователя.

Имея такие переменные, можно написать скрипт, обрабатывающий загрузку файла:

```
<? move_uploaded_file($userfile, "/download/$userfile_name"); ?>
```

Если вы укажете такой скрипт, состоящий только из одной строчки, в action, он примет файл, поместит его во временный каталог, «вычислит» оригинальное имя этого файла и скопирует в каталог /download/.

Дополнительно можно показать сообщение о том, что файл был успешно загружен, а также проконтролировать, с тем ли расширением был загружен файл, имеет ли он тот `content-type`, не превышает ли он максимально допустимый размер и т. д.

`move_uploaded_file` — встроенная функция последних версий PHP. Обратите внимание, что файл, который мы загружаем во временный каталог, доступен исключительно во время запроса на загрузку. После этого PHP уничтожит данный файл. А это значит, что после загрузки файла необходимо обязательно куда-нибудь переместить полученный файл, иначе вы его потеряете.

18.3. Возможные трудности

При использовании этой технологии не забудьте предварительно убедиться, что функция загрузки файлов включена администратором сервера в число доступных. Если администрируете вы сами, то просто отредактируйте конфигурационный файл

PHP (php.ini): проверьте, установлена ли опция file_uploads в значение «On». Также в этом файле можно отрегулировать максимальный размер загружаемых файлов по HTTP. За это отвечает опция upload_max_filesize.

СОВЕТ

Проверяйте размеры всех файлов, которые вам присылают. После загрузки файла с помощью PHP убедитесь в том, что полученные данные не превышают нужного размера. Иначе, если вы платите за превышение выделенного вам места на диске хостинг-провайдера, злоумышленник, закачавший вам ненужные данные, может нанести вам убытки.

Заключение

Конечно, есть огромное количество готовых скриптов на самых разных языках программирования. Но чаще всего не хочется связываться с чужими скриптами, разбираться в них. Хочется просто скопировать в свой скрипт небольшой фрагмент и наслаждаться результатом. Надеемся, что вы смогли разобраться в сути процессов, происходящий при загрузке файлов, а также понять, как это работает.

Глава 19
Эффективная работа в PHP при сетевом соединении с Web-сервером

Данную главу можно было бы назвать «Корректная работа в PHP». Не достаточно знать сами принципы программирования для эффективного процесса выполнения написанного вами приложения на PHP, необходимо также учитывать все погрешности, т. е. возможные нюансы выполнения нашего скрипта, а в частности взаимодействия «клиент—сервер». Ведь неизвестно, как именно будет выполнять пользователь наш скрипт на сервере. Нет гарантии, что он же не прервет его в процессе передачи данных серверу и выполнения. Следует сделать так, чтобы работа скрипта не проделывалась впустую в случае разрыва соединения или же остановки выполнения загрузки данных в браузер пользователя. Это и будет рассмотрено нами в данной главе.

Представьте себе ситуацию, когда ваш скрипт обрабатывает сложный запрос, что-то читает из базы, пишет в файл, изменяет данные и т. д., а во время этого процесса клиент нажал кнопку «Стоп» и разорвал соединение. Работа вашей программы будет прервана в самый неожиданный момент, и обработка не будет завершена. Но частично-то работа проделана. Необходимо как-то вернуть назад внесенные изменения. Для этого в PHP встроен механизм контроля за соединением. Вы можете подготовить функ-

цию аварийной остановки с помощью register_shutdown_function(). Данная функция вызывается интерпретатором PHP при завершении работы скрипта. Сетевое соединение в PHP с Web-сервером может иметь один из следующих статусов:

```
0 - NORMAL,
1 - ABORTED,
2 - TIMEOUT.
```

Когда PHP работает нормально, статус соединений будет NORMAL. Когда клиент прерывает соединение, устанавливается флаг ABORT. Если для выполнения скрипта не хватило времени, то будет установлен флаг TIMEOUT. Обратите внимание на функцию set_time_limit(). Она позволяет ограничить максимальное время выполнения. Ее синтаксис:

```
void set_time_limit(int seconds)
```

Функция распространяется на 3 и 4 версии PHP. Функция set_time_limit() позволяет выполнять скрипт на заданное вами время. Если время, заданное на выполнение сценария, достигнуто, то наш скрипт возвратит фатальную ошибку. Предел установления — 30 секунд (установлен по умолчанию в файле конфигурации), но при желании вы можете его изменить. Это изменение производится в файле конфигурации. Строка max_execution_time отвечает за время установления. Когда вы устанавливаете секунды на нуль, никакой предел на выполнение скрипта не наложен. Обратите внимание, чем может быть полезна данная функция. Прежде всего она может позволить нам в случае какой-либо проблемы разрыва программы просто прекратить ее выполнение из-за истечения установленного времени.

В случае вызова функции set_time_limit() автоматически происходит перезапуск счетчика паузы с нуля. Например, если время, установленное по умолчанию, равно 30 секундам, выполнение самого скрипта длится 20 секунд, а время, установленное нами в функции set_time_limit (), равно 15:

```
set_time_limit (15);
```

то время выполнения нашего скрипта будет равно 35 секундам.

ВНИМАНИЕ

Функция set_time_limit () неэффективна, когда PHP выполняется в режиме safe mode (безопасном режиме). Чтобы она работала корректно, необходимо отключить режим safe mode или изменить предел времени в файле конфигурации.

Вы можете указать интерпретатору, будет или нет ваш скрипт прерван при разрыве соединения. Этот вариант можно выбрать функцией ignore_user_abort(). Другой вариант — это создание функции аварийного завершения при помощи функций connection_aborted();, connection_timeout() И connection_status(). Если вы использовали ignore_user_abort(), то функция аварийного завершения может получить 2 статуса: ABORT, если прервали соединение, и TIMEOUT, если при этом скрипт выполнялся слишком долго.

ВНИМАНИЕ

Оба статуса — и ABORTED и TIMEOUT могут быть активными в одно и то же время. Это возможно в случае, когда вы заставляете PHP игнорировать аварийное завершение, т. е. прекращение операции обращения к ресурсу пользователя. В эти моменты PHP еще будет обращать внимание на то, что пользователь, возможно, прервал соединение, но сам процесс выполнения будет продлен.

Как известно, для сервера самый страшный враг — это сам пользователь. А если таких много на одном сервере? И обязательно найдется тот, кто начнет либо искать спрятанную информацию, либо просто пакостить. В PHP можно ограничить возможность пользовательского скрипта, чтобы предотвратить утечку чужих данных. Провайдеры обычно разрешают пользователям выполнять скрипты в безопасном режиме. Сильно это ограничение не сковывает, но помнить об этом надо.

ВНИМАНИЕ

Все описанное в этой главе распространяется на версию 3.0.7 и старше.

Заключение

Данная глава описала некорректный разрыв соединения, а также способы эффективной нейтрализации последствий.

Часть IV

PHP-функции

Глава 20

Функции для работы с массивами

Данная глава подробно рассказывает о способах взаимодействия массивов, а также о том, как можно манипулировать значениями массивов. Как известно, чтобы создать массив, необходимо также умение работать с ним. В этой главе и будут приведены способы преобразования массивов, манипуляции значениями массива, а также все, что связано с профессиональной и корректной работой массива.

Функции, которые будут рассмотрены, позволяют взаимодействовать и управлять массивами в различных направлениях. Массивы являются незаменимыми атрибутами для сохранения, управления и действия над заданными переменными. Простые и многомерные массивы поддерживаются в PHP и могут быть или разработаны пользователем, или созданы другой функцией.

Данная глава содержат следующие темы:

- подсчет значений массива;
- вычисления матриц;
- функциивозвращения;
- применение вызовов;
- функции объединения;
- сортировка массивов;
- вытеснение элементов из массива;
- получение элементовсогласно внутреннемууказателю массива;
- функции среза элементов массива.

20.1. Подсчет значений массива

При работе с массивами у программиста возникают задачи, когда необходимо не только ввести и вывести значения массива, но и производить действия над массивами (например, количественная оценка значений переменных, суммирова-

ние и подсчет количества индексов массива). Все это выполняют функции, описанные ниже.

Чтобы определить количество подобных переменных, т. е. одинаковых значений, в массиве, необходимо использовать функцию

```
array array_count_values(array input)
```

Данная функция возвращает массив со значениями, равными количеству совпадающих значений в первоначальном массиве. Проиллюстрируем это на примере:

```
$a = array(1,2,4,5,"hello",1,1,1,"hello", "world", 4);
array_count_values($a);
```

Выполнив данную программу, вы получите следующий массив:

```
Array ([1] => 4 [2] => 1 [4] => 2 [5] => 1 [hello] => 2 [world] => 1)
```

Из результата работы данной функции можно сделать вывод, что рассматриваемая функция подсчитала количество одинаковых значений и создала новый массив, из которого видно, что единица в массиве встречается четыре раза, двойка — один, четверка — два и т. д. Данная функция работает в версиях PHP 4 и выше.

Чтобы сложить все значения массива, вам потребуется такая функция:

```
mixed array_sum (array)
```

Например, вы имеете массив значений 3, 2, 1, 4. Результатом работы данной функции будет число, равное 10. Эта функция применяется в PHP 4 и более старших версиях.

ВНИМАНИЕ

Данная функция возвращает фиксированное значение, т. е. результатом ее деятельности не может быть массив. Тип, который будет принимать переменная при выполнении данной функции, — это либо integer, либо float.

Как работает данная функция, можно понять из примеров, приведенных ниже.

Пример 20.1. Функция array-sum (вариант 1)

```
<?
$a = array(5,7,10,1);
echo "Сумма значений элементов массива будет целым числом =
".array_sum($a)."\n";
?>
```

Результат программы:

```
Сумма переменных массива будет целым числом = 23
```

Пример 20.2. Функция array-sum (вариант 2)

```
<?
$b = array("a"=>1.4,"b"=>3.3,"c"=>7.4, "d");
echo "Данная сумма будет числом с плавающей запятой =
".array_sum($b)."\n";
?>
```

Результатом выполнения будет следующая строка:

```
Данная сумма будет числом с плавающей запятой = 12.1
```

Обратите внимание: в данном примере создан массив следующего синтаксиса: `array("a"=>1.4,"b"=>3.3,"c"=>7.4, "d");`. Этот массив имеет переменную d, которая не ссылается на какое-либо число. Поэтому число, которое будет прибавлять функция `array_sum()` во время получения значения переменной d, будет равным 0, т. е. будет происходить следующее: 1,4 + 3,3 + 7,4 + 0= 12,1.

ВНИМАНИЕ

При желании можно присвоить это значение какой-либо переменной. Например:

```
$c = array_sum($a);
```

После этого можно производить работу с данной переменной как полноправной, не считая ее массивом.

Иногда бывает необходимо подсчитать количество элементов переменной или массива. Для этого применяют функцию count(). Элементарный синтаксис:

```
int count (mixed var)
```

Данная функция возвращает целое число. Как правило, она применяется для подсчета значений массива. Основная особенность данной функции состоит в том, что она возвращает значение, равное единице, если переменная не является массивом. Если же она является массивом и не определена, то она возвращает значение 0. Во всех других случаях — единицу.

Функция также возвращает значение, равное нулю, когда переменная инициализирована как пустой массив. Чтобы узнать, имеет наша переменная какое-либо значение или нет, и была создана функция isset(), которая определяет, была ли переменная инициализирована. Например:

```
$my = "newValue";
echo isset($my); // true
unset($my);
echo isset($my); // false
```

Данная функция возвращает значение true в случае, если переменная определена, и false, если нет. Обратите внимание: в нашем примере использована функция unset(). Это полная противоположность функции isset().

Приведем несколько примеров, чтобы объяснить все вышесказанное подробнее.

Пример 20.3. **Функция** count **(вариант 1)**

```
$var = 25;
$var2 = "hello";
$result = count($var);
$result2 = count($var2);
echo $result;
echo $result2;
```

Этот пример выведет результат, равный двум единицам (1 1). Это значит, что само выполнение функции не зависит от тех значений, которым равны переменные, главное, чтобы они хоть чему-нибудь были равны. Обратите внимание: в первом случае переменная является целочисленной, а во втором — строкой.

Пример 20.4. Функция count **(вариант 2)**

```
$time[0] = "Hello";
$time[1] = 3;
$time[2] = 5;
$time[3] = "World";
$result = count($time);
echo $result;
```

Результатом выполнения данной функции будет число, равное 4. Думаем, тут все предельно ясно и не потребует каких-либо толкований.

Пример 20.5. **Функция** count **(вариант** 3)

```
$foo[0] = 1;
$foo[1] = 3;
$foo[4] = 5;
$result = count($foo);
echo $result;
```

Тут специально пропущена последовательность индексов массива, чтобы показать, что произойдет и как изменится результат по сравнению с предыдущим. В итоге получится три.

20.2. Вычисления матриц

Бывают такие ситуации, когда существует примерно два-три массива со своими значениями, при этом половина значений присутствуют и в первом, и во втором, и в третьем, а вам при постановке задач необходимо создать такой массив, в котором эти значения встречались бы только один раз, т. е. не повторялись ни в первом, ни во втором, ни в третьем. Данная задача решается при помощи функции `array_diff()`, которая используется в версиях старше PHP 4. Синтаксис данной функции выглядит следующим образом:

```
array array_diff(array array1, array array2 [, array...])
```

Данная функция возвращает массив, состоящий из всех элементов массива array1, не встречающихся с другими элементами. Другими словами, сравниваются значения и выбираются те, которые ни разу не совпали. Пример реализации данной функции:

```
$array1 = array ("a" => "машина", "человек", "собака");
$array2 = array ("b" => "машина", "человек", "дом");
$result = array_diff ($array1, $array2);
```

Результатом выполнения данной программы будет массив array("собака"). А результатом примера

```
$array1 = array ("a" => "green", "red", "blue");
$array2 = array ("b" => "green", "yellow", "red");
$array3 = array ("c" => "green", "blue", "red");
$result = array_diff ($array1, $array2, $array3);
print_r($result);
```

будет пустой массив, так как все элементы массива $array1 встречаются в массиве $array2 и в массиве $array3.

Теперь рассмотрим другую функцию, которая выводит массив всех значений, совпавших со значениями массивов, включенных в эту функцию. Данная функция выглядит так:

```
array array_intersect(array array1, array array2 [, array...])
```

Она, как и предыдущая, возвращает массив значений. Применяется в PHP 4 и старше. Рассмотрим примеры, описываемые нами ранее:

```
$array1 = array("a" => "машина", "человек", "собака");
$array2 = array("b" => "машина", "человек", "дом");
$result = array_intersect($array1, $array2);
```

Массив, который возвратит данная функция, будет представлять из себя следующее:

```
array ("a" => "машина", "человек");
```

Как видно, те значения, которые встречаются в массиве `$array2`, будут входить в наш новый массив.

Приведем другой пример:

```
$array1 = array("a" => "green", "red", "blue");
$array2 = array("b" => "green", "yellow", "red");
$array3 = array("c" => "green", "blue", "red");
$result = array_intersect($array1, $array2, $array3);
print_r($result);
```

После выполнения данного кода будет создан массив, содержащий все элементы массива $array1, т. е. он каким был, таким и останется.

20.3. Функции возвращения

Для более удобного программирования принято использовать своего рода указатели в массивах, т. е. $a => "hello". Иногда необходимо осуществить работу именно с самими указателями. Получить доступ к ним помогает функция `array_keys()`. Эта функция возвращает массив значений с указателями, которые были применены в массивах, приведенных в данной функции. Ее синтаксис:

```
array array_keys(array input [, mixed search_value])
```

Данная функция введена в PHP 4.

Приведем примеры применения данной функции.

Пример 20.6. Функция `array_keys` (вариант 1)

```
$array = array (0 => 100, "color" => "red");
array_keys($array); // результат array (0, "color")
```

Данный пример возвращает массив array(0,"color").

Пример 20.7. Функция array_keys (вариант 2)

```
$array = array("машина", "дом", "компьютер", "машина",
"машина");
array_keys($array, "машина"); // результат array (0, 3, 4)
```

Когда указателей нет, то после определения массива, с которым должна работать функция, также указывается значение, на какое именно ей стоит обращать внима-

ние. В нашем случае это "машина". В результате данная функция вернет следующий массив:

```
Array(0,3,4);
```

Это номера индексов, под которыми значения массива соответствует искомому значению "машина".

Пример 20.8. Функция array_keys (вариант 3)

```
$array = array("color" => array("blue", "red", "green"), "size"
=> array("small", "medium", "large"));
array_keys ($array); // результат array("color", "size")
```

Даже если сами указатели работают непосредственно с массивами, функция array_keys все равно будет возвращать массив указателей заданного в ней массива. Это иногда очень полезно и помогает избавиться от ненужных громоздких выражений вычисления, которые необходимо проделать, чтобы получить массив со значениями.

ВНИМАНИЕ

Эти операции необходимо было проделать в PHP 3, так как функция array_keys включена только в PHP 4.

Приведем пример:

```
function array_keys($arr, $term="") {
$t = array();
while (list($k,$v) = each($arr)) {
if ($term && $v != $term)
continue;
$t[] = $k;
}
return $t;
}
```

Рассмотрим функцию array_values, похожую по принципу действия на array_key(). Она предназначена для получения массива значений из массивов, внесенных в нее. Другими словами, функция array_key находит ссылки и создает массив из них, а функция array_value работает по тому же принципу, только она находит все значения, не обращая внимания на ссылки (указатели в этих массивах), и также создает массив из найденных значений:

```
array array_values(array input)
```

Применяется в PHP 4 и старше.

ВНИМАНИЕ

В PHP 3 для этой цели использовали следующую функцию:

```
function array_values($arr) {
$t = array();
while (list($k, $v) = each ($arr)) {
$t[] = $v;
return $t;
}
```

Она помогает реализовать тот же принцип, но в данный момент уже устарела, поэтому лучше пользоваться array_values, так вы сможете сэкономить время и энергию.

Пример выполнения array_values ():

```
$array = array("размер" => "XXL", "марка" => "BMW");
array_values($array); // возвращается array ("XXL", "BMW")
```

Результатом выполнения данного кода будет массив, состоящий из значений «XXL», «BMW».

Чтобы поменять местами значения массива, существует много разных функций, все зависит от того, в каком направлении действовать. Если необходимо поменять элементы в обратном порядке, применяется функция array_revers ():

```
array array_reverse(array array [, bool preserve_keys])
```

Введена в версию PHP 4 и старше, чем 4.0b4.

Если preserve_keys будет указана как true, то значения переменных массива изменят свое местоположение, а индексы останутся на прежнем месте. Например:

```
$input = array("мама", 4.0, array ("папа", "сестра"));
$result = array_reverse($input); // первый случай
$result_keyed = array_reverse($input, true); // второй случай
```

В первом случае получим:

```
Array ([0] => Array ( [0] => папа [1] => сестра) [1] => 4 [2] => мама).
```

Во втором случае результат будет следующий:

```
Array ([2] => Array ([0] => папа [1] => сестра) [1] => 4 [0] => мама)
```

Обратите внимание, что значение "мама" в первом случае находится в массиве под индексом, равным 2, а во втором случае индекс своего значения не изменил, т. е. остался равным нулю.

Параметр функции array_reserve — preserve_keys добавлен в функцию в PHP 4.0.3.

Как правило, каждый массив имеет свой внутренний указатель на текущий элемент, который изначально стоит на первом элементе, вставленном в матрицу.

Функция current () просто возвращает элемент массива, на котором находится внутренний указатель. Это не перемещает указатель. Если внутренние точки указателя вне списка элементов, то функция current () возвращает false.

Приведем пример:

```
<?
$input = array("php", 4.0, array ("green", "red")) ;
$my = current ($input);
$result_keyed = array_reverse ($input, true);
$my1 = current ($result_keyed);
print_r($my);
print_r($my1);
?>
```

Результат выполнения данной программы:

```
php
Array ([0] => green [1] => red)
```

Это те самые первые значения, о которых было сказано выше.

ВНИМАНИЕ

Если матрица содержит пустые элементы (0 или "" — пустая строка), тогда эта функция возвратит false для этих элементов. При использовании функции current () могут возникнуть проблемы при определении, находится ли указатель действительно в конце списка такого массива или нет.

Чтобы более подробно узнать о свойствах массива, а в частности о том, может он состоять из пустого элемента или нет, используйте функцию each ().

Данная функция возвращает значение текущего указателя и непосредственно само значение элемента в массиве. Синтаксис:

```
array each(array array)
```

Функция возвращает значение индекса матрицы и его переменной (текущее) и передвигает курсор в массиве.

Эти значения возвращаются в четырехэлементной матрице со значениями 0, 1, указателем и значением. Элементы 0 и указатель содержат ключевое название элемента массива, а 1 и значение содержат данные.

Если внутренний указатель матричных точек проходит мимо всех значений, функция each () возвращает false. Приведем пример:

```
$new = array ("саша", "игорь", "таня", "валера", "катя", "надя");
$too = each ($new);
```

Результатом выполнения данной программы будет массив, состоящий из следующих строк:

```
0 => 0
1 => 'саша'
key => 0
value => 'саша'
```

Так как никаких манипуляций с указателем матрицы не было произведено, то key = 0;. Теперь произведем преобразования:

```
$new = array("Саша" => "Шурик", "Виталик" => "Виталь");
$bar = each($new);
```

В итоге получим:

```
0 => 'Саша'
1 => 'Шурик'
key => 'Саша'
value => 'Шурик'
```

Если необходимо проверить наличие какой-либо конкретной переменной в массиве, как правило, используют функцию in_array (). Функция введена в PHP 4. Она работает с условными операторами и возвращает булевую переменную:

```
bool in_array(mixed значение, array имя_массива, bool strict)
```

Как видно, данная функция производит поиск значения в массиве, в поле, где указана строка "имя_массива". Она возвращает true, если значение, которое ищется в данном массиве, было найдено. Если переменная не была найдена, то данная функция возвращает false. Если в третьей строке данной функции bool strict установлен атрибут true, то функция in_array () осуществляет проверку типа "значения" в нашем массиве.

```
<?
$new = array("Hello", "NTT", "World", "New");
if (in_array ("NTT", $new))
{
print "Значение NTT в массиве $new найдено \n";
```

```
print_r ($new);
}
?>
```

В первой строке данного примера задается массив $new со значениями: Hello, NTT, World, New. Далее, как и было сказано выше, используется оператор условия, в котором и происходит проверка. В нашем примере функция возвращает значение true, так как искомое значение существует в массиве. После этого выполняются функции print() и print_r(). Результат данной программы будет следующим:

```
Значение переменной NTT в массиве Array найдено
Array ([0] => Hello [1] => NTT [2] => World [3] => New)
```

Поиск можно осуществлять также в более строгой форме, т. е. сравнивая помимо значения еще и тип. Например:

```
<?
$new = array("Hello", "NTT", "World", "New");
if (in_array ("NTT", $new, true))
{
print "Значение найдено в строгом соответствии с заданным типом";
}
?>
```

Результатом данной программы будет:

```
Значение найдено в строгом соответствии с заданным типом
```

Рассмотрев функцию in_array(), следует уделить внимание и функции array_search(). Она имеет аналогичный принцип работы, единственное отличие — возвращаемое значение в переменную. Функция in_array() возвращает значение true (истина) или false, а эта функция — значения индекса переменной, которым обозначается данная переменная, т. е. s[1] = 34; индексом является значение, равное 1. Искомое число в нашем случае будет 34. Синтаксис этой функции:

```
mixed array_search(mixed needle, array haystack, bool strict)
```

Приведем пример, характеризующий принципы работы данной функции:

```
<?
$new = array("Hello", "NTT", "World", "New");
$a = array_search("Hello", $new,true);
print "Индекс значения Hello равен $a<br>";
$b = array_search("NTT", $new,true);
```

```
print "Индекс значения NTT равен $b<br>";
$c = array_search("World", $new,true);
print "Индекс значения World равен $c <br>";
$d = array_search("New", $new,true);
print "Индекс значения New равен $d <br>";
print_r ($new);
?>
```

Результат работы данной программы:

```
Индекс значения Hello равен 0
Индекс значения NTT равен 1
Индекс значения World равен 2
Индекс значения New равен 3
Array ([0] => Hello [1] => NTT [2] => World [3] => New)
```

Обратите внимание, что все значения, которые возвращает функция, строго совпадают с нумерацией индекса.

20.4. Применение вызовов

Данные функции необходимы для того, чтобы программист мог без проблем передавать переменные имеющегося массива в указанную функцию. После работы функции она возвращает новое значение, которое потом записывается в новый массив. Этим и занимается функция `array_map()`. Элементарный синтаксис данной функции:

```
array array_map(mixed callback, array arr1 [, array arr2...])
```

Как видно, она возвращает новый массив переменных. Более конкретно это выглядит следующим образом: строка mixed callback задает имя функции, через которую будут обрабатываться в последовательном порядке значения массива. Строки array arr1 [,array arr2...] задают именно те массивы, т. е. их имена, из которых и будут браться те самые переменные. Например, имеем массив, состоящий из двух элементов: 1, 2, и функцию, которая будет просто умножать эти значения на два, так вот массив, который вернет данная функция, будет иметь другое имя и следующие значения: 2, 4.

Рассмотрим программу, которая будет умножать на три каждый из элементов нашего массива и при этом создаст новый массив:

```
<?
function sum($n) {
```

```
return $n*3;
}
$a = array(1, 2, 3, 4, 5) ;
$b = array_map(sum,$a);
?>
```

В данном примере последовательно будут вноситься значения массива и умножаться на 3, далее будет создан сам массив `$b = array(3,6,9,12,15);`.

ВНИМАНИЕ

Эта функция работает в самых новых версиях PHP. Поэтому она не работает в таких, как PHP 3 или PHP 4 бета-версиях.

Чтобы совершить работу не с одним, а с двумя массивами, необходимо использовать следующие принципы построения кода. Пример поможет вам познакомиться с этими принципами:

```
function show_Spanish($n, $m) {
return "The number $n is called $m in Spanish";
}
function map_Spanish($n, $m) {
return array ($n => $m);
}
$a = array(1, 2, 3, 4, 5);
$b = array("uno", "dos", "tres", "cuatro", "cinco");
$c = array_map("show_Spanish", $a, $b);
print_r($c);
//результат данной работы данной программы приведен ниже:
// Array
// (
// [0] => The number 1 is called uno in Spanish
// [1] => The number 2 is called dos in Spanish
// [2] => The number 3 is called tres in Spanish
// [3] => The number 4 is called cuatro in Spanish
// [4] => The number 5 is called cinco in Spanish
// )
```

```
$d = array_map("map_Spanish", $a, $b);
print_r($d);
```

Результат работы:

```
Array([0] => Array([1] => uno) [1] => Array([2] => dos)
[2] => Array([3] => tres) [3] => Array([4] => cuatro)
[4] => Array([5] => cinco))
```

Обычно при использовании двух или более массивов они должны иметь равную длину, потому что функция повторного вызова применяется параллельно к соответствующим элементам. Если массивы имеют неравную длину, самый короткий будет продлен при помощи пустых элементов, т. е. будет заполнен пустыми элементами.

Следующий пример показывает принцип работы данной функции с тремя массивами:

```
$a = array(1, 2, 3, 4, 5);
$b = array("one", "two", "three", "four", "five");
$c = array("uno", "dos", "tres", "cuatro", "cinco");
$d = array_map(null, $a, $b, $c);
print_r($d);
```

Результат:

```
Array([0] => Array([0] => 1[1] => one[2] => uno) [1] => Array(
[0] => 2[1] => two[2] => dos) [2] => Array([0] => 3
[1] => three[2] => tres) [3] => Array([0] => 4[1] => four
[2] => cuatro) [4] => Array([0] => 5[1] => five[2] => cinco))
```

Существует еще две функции, аналогичные по принципу работы `array_map()`. Первая из них `array_filter`. Ее синтаксис:

```
array array_filter(array input [, mixed callback])
```

Данная функция возвращает массив элементов, т. е. новый массив, при условии, что каждый элемент был обращен к заданной функции. После этого над ним производятся операции и он либо входит в новый массив, либо нет. Например, нужно выбрать из массива элементы значений, остаток от деления на два которых равен единице, или что-нибудь в этом роде. Так происходит формирование массива. Пример работы данной функции:

```
function foo($var) {
return($var % 2 == 1);
}
```

```
function foos($var) {
return ($var % 2 == 0) ;
}
$array1 = array("a"=>1, "b"=>2, "c"=>3, "d"=>4, "e"=>5) ;
$array2 = array( 23, 24, 25, 26, 27, 28, 29, 30);
$odd = array_filter($array1, "foo");
$even = array_filter($array2, "foos");
```

Как видно из примера, в первом случае создается массив, содержащий элементы массива $array1, делящиеся на 2 с остатком, равным 1. И результат будет следующий:

```
$odd = array("a"=>1, "c"=>3, "e"=>5);
```

Во втором случае создается массив со значениями из массива $array2, все значения которого делятся на 2 без остатка, все остальные значения отбрасываются. Результат программы:

```
$even = array(24, 26, 28, 30)
```

ВНИМАНИЕ

Если работа осуществляется с ассоциативным массивом, то при работе функции array_filter() также возвращается массив значения с указателями.

Следующая функция, которую нам необходимо рассмотреть, — array_reduce(). Синтаксис:

```
mixed array_reduce (array input, mixed callback [, int initial])
```

Этот массив вызывает постоянное значение, т. е. просто число, в соответствии с условиями, заданными в параметре callback. Массив, с которого будет осуществляться процесс вызова значений, указывается в input. В строку int initial заносятся те значения, которые вы совместно со значениями массива хотели бы внести в функцию callback. Пример работы данной функции:

```
<?
function rsum($v, $w) {
$v+ = $w;
return $v;
}
function rmul($v, $w) {
$v* = $w;
```

```
return $v;
}
$a = array(1, 2, 3, 4, 5);
$x = array();
$b = array_reduce($a, "rsum");
$c = array_reduce($a, "rmul", 10);
$d = array_reduce($x, "rsum", 1);
?>
```

Результатом работы данного примера будут следующие значения переменных:

```
$b = 1+2+3+4+5 = 15
$c = 1*2*3*4*5*10 = 1200
$d = 1
```

Рассмотрим еще одну функцию — array_walk. Она производит добавление каждого значения массива в используемую функцию. Ее синтаксис:

```
int array_walk(array arr, string func, mixed userdata)
```

Данная функция работает с последовательностью значений указанного массива arr, в строке func задается сама функция и третий параметр, который будет вводиться в функцию для обработки данных (первые два — это указатель и само значение).

Если func требует больше, чем два или три параметра (в зависимости от userdata), всегда будет выдаваться ошибка при вызове функцией array_walk (), указанной func. Эти предупреждения могут быть подавлены знаком @, запросом к функции array_walk() или при помощи error_reporting ().

ВНИМАНИЕ

Если func должна работать с фактическими величинами матрицы, необходимо определить ее первый параметр при помощи func путем пропуска ссылкой. Тогда любые изменения элементов будут сделаны непосредственно в самом массиве.

Передачи указателя и userdata в func были добавлены в PHP 4.0.

Функция func работает в PHP 3 и выше.

20.5. Функции объединения

Данные функции необходимы для того, чтобы можно было объединять несколько массивов в один и производить дальнейшую работу с полученным массивом. Рассмотрим две функции: array_merge () и array_merge_recursive ().

Функция arrayjnerge () очень проста в использовании. Ее синтаксис:

```
array array_merge(array array1, array array2 [, array...])
```

Данная функция используется в PHP 4. Если при вызове arrayjnerge () у вас возникнут ошибки, непременно проверьте версию вашего интерпретатора. Функция вызывает массив значений, состоящий из элементов значений массивов array1, array2... Последовательность сортировки будет следующая: сначала будут следовать значения первого массива, далее второго и т. д., т. е. значения каждого последующего массива переходят в конец предыдущего. Это очень важно запомнить, так как иногда из-за непонимания этого возникают ошибки работы с данной функцией.

ПРИМЕЧАНИЕ

Если в массивах встречаются одинаковые значения, то сортировка происходит по тем же правилам, за исключением того, что одинаковое значение (например, указатель) будет использовано из предыдущего массива.

Приведенный пример поможет вам освоить принцип работы данной функции:

```
<?
$array1 = array("color" => "красный", 1.2, 1.4);
$array2 = array("a", "b", "color" => "синий", "shape" => "круглый", 2.4);
$array3 = array( 3.2, 3.5, "color" =>"зеленый", 3.8);
$array4 = array_merge($array1, $array2, $array3);
print_r($array4);
?>
```

Как видно из данного примера, все массивы имеют одинаковое имя указателя, хотя значения разные. В этом случае будет вставлен тот указатель, который находится в последнем массиве (в нашем случае "color" =>"зеленый"). Результат работы данной программы:

```
Array ([color] => зеленый [0] => 1.2 [1] => 1.4 [2] => a [3] =>
b [shape] => круглый [4] => 2.4 [5] => 3.2 [6] => 3.5 [7] =>
3.8)
```

Обратите внимание на последовательность вставляемых значений: сначала идут все значения из массива $array1, затем $array2, и т. д. В общем весь процесс подчиняется строгому логическому порядку, помнить о котором ваша задача.

Теперь рассмотрим функцию array_merge_recursive(). Принцип ее работы полностью аналогичен вышеописанной функции arrayjnerge (). Единственное отличие состоит в том, что функция arrayjnerge () просто производит объединение массивов, а array_merge_recursive() производит объединение массивов рекурсивно. Синтаксис данной функции следующий:

```
array array_merge_recursive (array array1, array array2 [, array...])
```

Она также возвращает массив значений. Применяется непосредственно в PHP старше 4-й версии. Все тонкости, которые касались работы функции array_merge(), относятся также и к этой функции.

Приведем пример:

```
<?
$arrray1 = array("color" => array("favorite"=> "красный"),
1.5);
$arrray2 = array(2.10, "color" => array("favorite" => "зеле-
ный", "синий"), 2.15 );
$result = array_merge_recursive($arrray1, $array2);
print_r($result);
?>
```

Результатом работы данной программы будет массив, содержащий следующие значения:

```
array ("color" => array("favorite" => array("красный", "зеле-
ный"), "синий"), 1.5, 2.10, 2.15);
```

Обратите внимание, как произошла обработка данных. Был создан массив array ("красный", "зеленый"), "синий"), при этом не произошло никакого удаления первого значения массива, а произошло последовательное подчинение значений в массиве. Если изменить ранее описанный пример, то программа будет работать следующим образом:

```
<?
$array1 = array("color" => "красный", 1.2, 1.4);
$array2 = array("a", "b", "color" => "синий", "shape" =>
"круглый", 2.4);
$array3 = array( 3.2, 3.5, "color" =>"зеленый", 3.8);
$array4 = array_merge_recursive($array1, $array2, $array3);
print_r($array4);
?>
```

Результат программы:

```
Array ([color] => Array ([0] => красный [1] => синий [2] => зеле-
ный) [0] => 1.2 [1] => 1.4 [2] => a [3] => b [shape] => круглая
[4] => 2.4 [5] => 3.2 [6] => 3.5 [7] => 3.8)
```

Из этой труднопонимаемой на первый взгляд строки можно сделать вывод и привести отличия функций array_merge() и array_merge_recursive(). Последняя функция, вместо того чтобы заменять значения, производит создание отдельного массива, подчиненного тому или иному указателю, встречающемуся повторно, и при этом сами значения формируются в последовательном порядке. А у функции array_merge() немного не такой принцип, т. е. она производит замену имеющегося значения указателя на встреченное ниже при условии, конечно, что указатели имеют одинаковое имя. Выбирать, какой функцией лучше пользоваться, нужно с учетом задачи, стоящей перед программистом. Думаем, что с этим вы справитесь без проблем.

20.6. Сортировка массивов

При программировании массивов иногда возникает острая необходимость применить их сортировку, при этом сортировка необходима в разнообразном порядке, т. е. по убыванию, по возрастанию, в алфавитном порядке и т. д. Решить данную задачу призвано множество функций в PHP.

Первая функция, которую мы рассмотрим, — array_miltisort(). Она позволяет сортировать данные массива в нужном нам направлении. Данная функция возвращает значение булевого типа (true или false). Часто программисту необходимо прямо в самом операторе условия и не только там пользоваться функцией array_multisort, чтобы сократить объем кода, а также время выполнения программы. Синтаксис данной функции:

```
bool array_multisort(array ar1 [, mixed arg [, mixed... [, array...]]])
```

Данная функция используется в таких версиях старше PHP 4. Программисты иногда пытаются использовать в функции array_multisort () несколько массивов одновременно, чтобы сортировать несколько матриц сразу или многомерную матрицу, содержащую одну или несколько составляющих. При сортировке поддерживается зависимость индексов массива. Данная ситуация довольно плачевно заканчивается.

ВНИМАНИЕ

В функции array_multisort() нельзя производить сортировку сразу нескольких массивов одновременно.

Сам процесс сортировки заключается непосредственно в составлении таблицы, каждая матрица занимает в ней свое место, затем и происходит эта самая сортировка данных последовательно по строкам таблицы. Данный процесс чем-то похож на функциональные возможности сортировки SQL ORDER BY.

Структура параметров функции array_multisortнемного необычна, но, с другой стороны, довольно-таки гибка. Самый первый параметр должен являться массивом. Далее, если вы сортируете не один массив, а несколько, предлагается вводить следующие массивы для сортировки, так как данная функция может работать дале-

ко не с одним массивом. После того как вы введете все необходимые вам массивы, для сортировки можно указать своеобразные ключи, конечно, при необходимости. Данную функцию можно использовать как с ключами, так и без.

Параметры, устанавливающие порядок сортировки:

SORT_ASC — осуществляет сортировку в порядке возрастания;

SORT_DESC — сортирует в порядке убывания.

Параметры, устанавливающие сортировку по типам:

SORT_REGULAR — осуществляет сортировку как строк, так и чисел;

SORT_NUMERIC — осуществляет сортировку чисел;

SORT_STRING — осуществляет сортировку строк.

Чтобы данная спецификация сортировки происходила (была применена к конкретному виду массива), необходимо после каждого массива в функции array_multisort() указывать нужные вам тип и способ сортировки. Если вы это сделаете только для одного массива, на другой она распространяться не будет. Приведем пример работы с массивами:

```
<?
$arl = array (10, 100, 5, a, 4, 7, 1, 3, 8, b, u, d);
$ar2 = array (1, 3, "2", "1", 10, 20, 10, 40, d, r, f, a, c);
print_r($ar1);
print_r($ar2);
array_multisort ($ar1,SORT_ASC,SORT_STRING);
array_multisort($ar2,SORT_ASC,SORT_NUMERIC);
print_r($ar1);
print_r($ar2);
?>
```

Этот пример показывает способ работы с так называемыми параметрами обработки массивов. В первом случае происходит сортировка только строк массива $arl в порядке возрастания, во втором случае — только числовых значений массива $ar2 в порядке возрастания. Результат данной программы:

```
Array ([0] => 10 [1] => 100 [2] => 5 [3] => a [4] => 4 [5] => 7 [б] => 1 [7] => 3 [8] => 8 [9] => b [10] => и [11] => d)
Array ([0] => 1 [1] => 3 [2] => 2 [3] => 1 [4] => 10 [5] => 20 [б] => 10 [7] => 40 [8] => d [9] => r [10] => f [11] => a [12] => c)
Array ([0] => 1 [1] => 10 [2] => 100 [3] => 3 [4] => 4 [5] => 5 [б] => 7 [7] => 8 [8] => a [9] => b [10] => d [11] => u)
```

```
Array ([0] => c [1] => f [2] => d [3] => r [4] =>. a [5] => 1 [6] =>
1 [7] => 2 [8] => 3 [9] => 10 [10] •=> 10 [11] => 20 [12] => 40)
```

Первые два массива — исходные, вторые два — сортированные. Существует также и другие способы сортировки массивов. Например:

```
<?
$mas = array(100, 10, 100, "v");
$mas2 = array(1, 3, 67, 30);
array_multisort($mas, $mas2);

print_r($mas);
print_r($mas2);
?>
```

Результатом сортировки при помощи данной функции будут следующие массивы:

```
Array ([0] => v [1] => 10 [2] => 100 [3] => 100)
Array ([0] => 30 [1] => 3 [2] => 1 [3] => 67)
```

Обратите внимание: так как конкретные параметры сортировки данной функции не были заданы, вы не смогли получить предсказуемый результат.

В следующем примере обратите внимание, каким способом осуществляется работа с массивами:

```
<?
$ar = array(array (45, 67, "s", 35, "a", "g") , array (1, 4, 8,
7, 1, 3»;
array_multisort($ar[0], SORT_ASC, SORT_STRING,
$ar[1], SORT_NUMERIC, SORT_DESC) ;
print_r($ar[0]);
echo "<br>";
print_r($ar[1]);
echo "<br>";

array_multisort($ar[0], SORT_ASC, SORT_STRING, $ar[1], SORT_ASC,
SORT_NUMERIC);

print_r($ar[0]);
echo "<br>";
```

```
print_r($ar[1]);
echo "<br>";

array_multisort($ar[1], SORT_ASC, SORT_NUMERIC, $ar[0], SORT_ASC,
SORT_STRING);

print_r($ar[1]);
echo "<br>";
print_r($ar[0]);
echo "<br>";

$ar = array(array (45, 67, "s", 35, "a", "g"), array (1, 4, 8,
7, 1, 3));
array_multisort($ar[0], SORT_ASC, SORT_STRING);
array_multisort($ar[1],SORT_NUMERIC, SORT_DESC);

print_r($ar[0]);
echo"<br>";
print_r($ar[1]);
?>
```

Результаты работы программы:

```
Array ([0] => 35 [1] => 45 [2] => 67 [3] => a [4] => g [5] => s)
Array ([0] => 7 [1] => 1 [2] => 4 [3] => 1 [4] => 3 [5] => 8)
Array ([0] => 35 [1] => 45 [2] => 67 [3] => a [4] => g [5] => s)
Array ([0] => 7 [1] => 1 [2] => 4 [3] => 1 [4] => 3 [5] => 8)
Array ([0] => 1 [1] => 1 [2] => 3 [3] => 4 [4] => 7 [5] => 8)
Array ([0] => 45 [1] => a [2] => g [3] => 67 [4] => 35 [5] => s)
Array ([0] => 35 [1] => 45 [2] => 67 [3] => a [4] => g [5] => s)
Array ([0] => 8 [1] => 7 [2] => 4 [3] => 3 [4] => 1 [5] => 1)
```

ВНИМАНИЕ

При перемене мест ключей или задании способа сортировки результат не изменяется. При использовании более одного массива в функции сортировки сортируется только первый, остальные какими были, такими и остаются.

Думаем, вы сможете при виде результатов понять принципы работы и правильного оформления данной функции.

Еще одна функция сортировки значений ассоциативных массивов — arsort(). Легко запоминается и имеет также легко используемый синтаксис:

```
void arsort(array array [, int sort_flags])
```

Применяется в PHP 3 и PHP 4. Данная функция применяется в ассоциативных массивах, т. е. она сортирует значения массива в обратном порядке. Эта сортировка применима как к строкам, так и к числам.

ВНИМАНИЕ

Сортировку указателей, используемых в массиве, данная функция не осуществляет.

Приведем пример применения функции arsort():

```
<?
$world = array("d" => "car", "a"=>"house", "b"=>"apple", "c"=>"table",
"e" => "quality");
arsort($world);
reset($world);
while (list ($key, $val) = each ($world))
{
echo "$key = $val\n";
}

echo "<br>";
$number = array ("d"=>56, "a"=>23, "b"=>89, "c"=>1, "e" => 30 ) ;
arsort($number);
reset($number);
while (list ($key, $val) = each ($number) )
{
echo "$key = $val\n";
}
?>
```

Результатом работы этой программы будут следующие строки:

```
c = table e = quality a = house d = car b = apple
b = 89 d = 56 e = 30 a = 23 c = 1
```

В первом случае происходит сортировка массива, состоящего из строк, во втором — из численных значений.

Способ сортировки данной функции можно изменять, основные параметры, предназначенные для этого, — такие же, как при использовании функции sort(). Данная функция производит сортировку массива в прямом порядке и имеет следующий синтаксис:

```
void sort(array array [, int sort_flags])
```

Он является аналогичным тому, который имеет функция arsort(). Рассмотрим применяемые в двух данных функциях флаги, т. е. переменные, необходимые для задания способа сортировки:

SORT_REGULAR — сортирует значения массива (как чисел, так и строк);

SORT_NUMERIC — сортирует только числа;

SORT_STRING — сортирует только строки.

Для более полного понимания проделайте следующую операцию. Возьмите функцию arsort() в приведенном выше примере и замените на функцию sort(). Сортировка значений переменных в этом случае будет осуществляться в прямом порядке и результат работы программы будет выглядеть следующим образом:

```
0 = apple 1 = car 2 = house 3 = quality 4 = table
0 = 1 1 = 23 2= 30 3= 56 4= 89
```

Применение флагов в данной функции было заложено только с версии PHP4.

ВНИМАНИЕ

Функция sort о не поддерживает индексную зависимость, т. е. систему указателей в массиве, в отличие от функции сортировки asort().

Функция asort() имеет те же свойства, как и ранее приведенные две функции arsort(), sort(). Приведем синтаксис, хотя он аналогичен предшествующим функциям:

```
void asort(array array [, int sort_flags])
```

Если же мы теперь изменим в приведенном ранее примере значения функции arsort() на asort(), то получим:

```
b = apple d = car a = house e = quality c = table
c = 1 a = 23 e = 30 d = 56 b = 89
```

Как видно, данная функция позволяет осуществлять сортировку с указателями массива, чего нельзя сделать при использовании функции sort().

Еще одна функция сортировки массивов в обратном порядке — rsort(), она является почти полной аналогией функции sort(), единственное отличие — первая сортирует в обратном порядке, а sort O — в прямом. При подстановке rsort() вместо sort() в вышеописанный пример результат будет следующий:

```
O = table 1 = quality 2 = house 3 = car 4 = apple
0= 89 1= 56 2= 30 3= 23 4=1
```

Функция ksort() сортирует непосредственно по указателю, а не по значению указателей:

```
a = house b = apple c = table d = car e = quality
a = 23 b = 89 c = 1 d = 56 e = 30
```

Функция krsort() сортирует сами указатели в обратном порядке:

```
e = quality d = car c = table b = apple a = house
e = 30 d = 56 c = 1 b = 89 a = 23
```

При этом значения нас не интересуют.

Данная функция применяется в PHP старше третьей версии.

Рассмотрим функцию, которая сортирует данные в естественном виде. При обычной сортировке сравнивается каждый последующий символ в алфавитном порядке. Например, нужно отсортировать значения max10, max11, max12, max1, max2, max21, max22. Сортировка, описанная ранее, осуществляется следующим образом: maxl, maxlO, maxll, max12, max2, max21, max22. Функция, которая будет рассмотрена, сортирует значения так: maxl, max2, maxlO, maxll, max12, max21, max22. Данный способ сортировки и называется натуральной сортировкой, и этим занимается функция natsort(). Функция имеет следующий синтаксис:

```
void natsort (array array)
```

Функция используется непосредственно в PHP 4 и старше 4.0RC2. Приведенный ниже пример поможет разобраться, чем же отличается стандартный способ сортировки от натурального:

```
<?
$ar1 = $ar2 = array ("max12","max10","max2","max1");
sort($ar1);
echo "<br>Стандартный порядок сортировки<br>\n";
print_r($ar1);
natsort($array2);
echo "\n<br>Натуральный порядок сортировки<br>\n";
```

```
print_r ($ar2) ;
?>
```

Как видно, в примере приводятся функции sort() и natsort(). Результат работы данной программы приведен ниже:

```
Стандартный порядок сортировки
Array ([0] => maxl [1] => max10 [2] => max12 [3] => max2)
Натуральный порядок сортировки
Array ([0] => max12 [1] => maxlO [2] => max2 [3] => maxl)
```

Если необходимо создать свой сценарий сортировки, чтобы провести сортировку по какому-либо специально установленному условию, то осуществить это в PHP не составит труда. Существуют функции, которые помогают производить сортировку переменных, определяемую непосредственно самим программистом. К такому типу относятся следующие функции:

- usort(),
- uksort(),
- uasort().

Функция usort() использует определяемые пользователем массивы и сортирует значения данного массива, работая непосредственно с указанной пользователем функцией. Пользователь создает функцию с определенными условиями и, ссылаясь на данную функцию, производит сортировку массива. Синтаксис ее следующий:

```
void usort(array array, string any_function)
```

Принцип ее работы заключается в следующем: она сортирует массив значений, используя определяемую пользователем функцию сравнения. Если массив, который необходимо сортировать, нуждается в использований каких-либо особых условий, вам следует воспользоваться данной функцией. Например:

```
<?
function des($a, $b) {
if ($a == $b) return 0;
return($a < $b);
}
$a = array(3, 2, 2, 5, 6, 1);
usort($a, "des");
while (list ($key, $value) = each ($a)) {
echo "$key: $value\n";
}
?>
```

В примере была задана функция, которая будет производить сначала сравнение двух переменных, взятых из массивов $a и $b. После этого, если они равны, функция возвращает значение, равное нулю. Иначе она возвращает значение $a<$b. Таким образом и осуществляется сортировка в обратном порядке (имеется в виду по убывающей). Результат работы данной функции:

```
0: б
1: 5
2: 3
3: 2
4: 2
5: 1
```

Обратите внимание, что она работает даже в случае применения повторяющихся значений.

Следующая функция, которую мы рассмотрим, — uksort(). Она осуществляет сортировку не самих значений массива, а именно указателей на них, т. е. работает с ассоциативными массивами. Сам процесс работы аналогичен приведенной выше функции. Ее синтаксис:

```
void uksort (array array, function name_function)
```

Если в рассмотренной выше программе произвести замену массива $a на новый:

```
$a = array (1 => "один", 2 => "два", 24 => "двадцать четыре", 15
=> "пятнадцать");,
```

а функцию usort() на uksort(), то результат программы будет следующий:

```
24: двадцать четыре
15: пятнадцать
2: два
1: один
```

Как видно, произошла сортировка не значений указателей, а самих указателей по убыванию. Чтобы сортировка происходила по возрастанию, необходимо вместо знака < поставить знак > в строке return ($a < $b);.

Функция uasort() похожа на ранее описанные функции. Единственное ее отличие заключается в том, что она осуществляет сортировку с поддержкой индексной зависимости массива. Приведем описанный ранее пример с измененными параметрами:

```
<?
function des($a, $b) {
```

```
if ($a == $b) return 0;
return ($a < $b);
}
$a = array ( 15, 5, 24, 1);
uasort($a, "des");
while (list ($key, $value) = each ($a)) {
echo "$key: $value\n";
}
?>
```

Результат работы данной функции:

```
2: 24
0: 15
1: 5
3: 1
```

Обратите внимание на индексную зависимость, о которой было сказано ранее. При пользовании функции `uasort()` индексная зависимость ни в коем случае не изменяется. Это и делает данную функцию отличной от приведенных выше.

20.7. Вытеснение элементов из массива

Иногда программисту необходимо вытеснить данный элемент, при этом не потерять его, а переместить в указанное место.

Функция `array_pop()` предназначена именно для такой цели. Она возвращает последний элемент массива, заданного ей. Ее синтаксис:

```
mixed array_pop(array array)
```

Если массив данной функции (array) является пустым либо ей задается просто какая-либо переменная, которая не является массивом, то данная функция возвращает значение, равное нулю.

Например:

```
<?
$stack = array ("муж", "жена", "любовник");
$myhus = array_pop ($stack);
print_r($stack);
```

```
echo "<br>";
print $myhus;
echo "<br>";

$free = array ();
$popfree = array_pop($free) ;
print_r ($free);
echo "<br>";
print $popfree;
echo "<br>";

$foo = 5;
$valfoo = array_pop($foo);
echo $valfoo;
?>
```

Данная программа состоит из трех блоков. Первый блок — с массивом и применяющимися к нему функциями. В нем произойдет вытеснение последнего значения массива, которое будет присвоено переменной `$myhus`. Далее выведутся результаты значений программы.

Второй блок показывает, что произойдет в случае работы функции с пустым массивом.

Третий блок характеризует принцип работы программы с переменной.

Приведем результаты работы данной программы:

(первый блок)

```
Array ([0] => муж [1] => жена)
любовник
```

(второй блок)

```
Array ( )
```

Как видите, третьего вообще нет, это говорит о том, что нуль не был выведен. Отсюда можно сделать вывод, что этот нуль подразумевается, но оперировать с ним как со значением не стоит. Лучше присвойте истинный нуль и смело работайте.

Если вам необходимо вставить некоторые значения в массив, применяется функция `array_push()`. Данная функция имеет следующий синтаксис:

```
int array_push(array array, mixed var [, mixed...])
```

Используется в PHP 4. Функция довольно проста в использовании и не создает труда при программировании. Приведем пример ее практического использования:

```
<?
$new_work = array(1,2,5,7);
array_push($new_work, "+", 8,9,10);
print_r($new_work);
?>
```

Результатом работы данного примера будет следующая строка:

```
Array ([0] => 1 [1] => 2 [2] => 5 [3] => 7 [4] => + [5] => 8 [6] =>
9 [7] => 10)
```

Таким способом вы можете добавлять любые значения, которые необходимы вам при работе.

Если нужно вытеснить элемент из начала массива, то вам необходимо воспользоваться функцией array_shift (). Она имеет следующий синтаксис:

```
mixed array_shift(array array)
```

Работает аналогично array_pop(). Приведем пример:

```
<?
$new_work = array(1, 2,5,7);
$foo = array_shift($new_work);
print_r($new_work);
echo"<br>";
echo $foo;
?>
```

Результат выполнения примера:

```
Array ([0] => 2 [1] => 5 [2] => 7)
1
```

Если вам необходимо вставить один или несколько элементов в начало массива, применяется функция array_unshift(). Эта функция аналогична функции array_push(). Приведем пример:

```
<?
$new_work = array(1,2,5,7);
array_unshift($new_work, 20,21,22,23);
print_r($new_work);
?>
```

Пример выведет следующую строку с добавленными элементами:

```
Array ([0] => 20 [1] => 21 [2] => 22 [3] => 23 [4] => 1 [5] => 2
[6] => 5 [7] => 7)
```

СОВЕТ

Если вам необходимо удалить из имеющегося массива все повторяющиеся элементы, воспользуйтесь функцией array_unique(). Например, если у вас есть массив, состоящий из элементов 1, 2, 3, 4, 2, 2, 4, то после применения данной функции будет возвращен новый массив, состоящий из элементов 1, 2, 3, 4. Все повторяющиеся элементы будут удалены. Данная функция работает в PHP 4.0.1.

20.8. Получение элементов согласно внутреннему указателю массива

Очень часто при работе с массивами возникает необходимость получить тот или иной элемент массива, переместить внутренний указатель. В каждом массиве существует свой внутренний указатель, работа которого и определяет элемент, с которым в данный момент должен работать массив. Решить проблему возврата значений и перемещения внутреннего указателя призваны следующие функции:

- pos(),
- end(),
- next(),
- prev(),
- reset().

Функция pos() имеет следующий синтаксис:

```
mixed pos (array array)
```

Она производит возврат элемента массива, но не просто неопределенного элемента, а именно того, на котором в данный момент программы находится внутренний указатель. Если он стоит на первом элементе, то будет возвращен первый элемент. Например:

```
<?
$new_work = array(1,2,5,7) ;
$a = pos ($new_work);
echo $a;
?>
```

На экран будет возвращено значение, равное единице, так как в начальный момент внутренний указатель находится именно в том месте. Данная функция используется в PHP 3 и PHP 4.

Если необходимо перевести внутренний указатель массива на последний элемент, используют функцию end(). Она имеет синтаксис, аналогичный pos(). Если изменить нашу программу и вместо функции pos() записать end(), то внутренний указатель переместится в конец нашего массива и вернет значение, равное семи.

Функция next() перемещает внутренний указатель массива на следующий элемент. В нашей программе при использовании функции next() получим результат, равный двум. Сначала внутренний указатель находился на первом элементе, рассматриваемая функция перевела его на второй и возвратила значение.

Функция prev() позволяет переместить указатель на предыдущий элемент. Рассмотрим программу

```
<?
$new_work = array(1,2,5,7);
$a = end($new_work);
$b = prev($new_work);
echo $b;
?>
```

Программа выведет значение, равное пяти. В первом случае указатель будет перемещен в конец массива, а затем при помощи функции prev() будет перемещен на предыдущий элемент.

Функция reset() перемещает внутренний указатель на первый элемент. Она н г аналогична работе функции pos(). Дело в том, что функция pos() устанавливает внутренний указатель на текущий элемент и возвращает это значение, а функция reset() — только на первый элемент массива и также производит возврат значения.

ВНИМАНИЕ

Все вышеописанные функции подчиняются одному правилу: если массив, к которому они производят обращение, является пустым, то каждая из этих функций возвратит значение, равное false (ложь).

20.9. Функции среза элементов массива

Данные функции позволяют получить доступ к любому элементу или последовательности элементов массива. К таким функциям относится функция array_slice(). Она имеет следующий синтаксис:

```
array array_slice(array array, int offset [, int length])
```

Применяется в PHP4. Array_slice() возвращает последовательность элементов массива с указанными параметрами длины и смещения.

После того как указывается массив, из которого необходимо производить срез, производится задание самих параметров специфики среза. Эти параметры указываются в цифрах. Цифры в свою очередь могут быть положительными или отрицательными. Если цифра является положительной, отсчет значений массива, т. е. порядка, начинается с первого элемента массива, если используется отрицательная цифра, отсчет производится с конца массива. После того как вы задали первые цифры, например 2, можно задать еще одну, чтобы сделать срез более точным. Например, из параметров 2, 2 следует, что отсчитываются два элемента с начала матрицы, после этого второй параметр соответствует тому, что следующие два значения этого массива будут возвращены (т. е. не весь массив, а только эти два значения). Более понятно вам станет после того, как вы изучите пример:

```
<?
$arr = array ("a", "b", "c", "d", "e", "g" ) ;
$out = array_slice ($arr, 4) ; // первый блок
print_r($out);
echo "<br>";
$out = array_slice ($arr, 2, -1); // второй блок
print_r($out);
echo "<br>";
$out = array_slice ($arr, -2, 1); // третий блок
print_r($out);
echo "<br>";
$out = array_slice ($arr, 0, 3) ; // четвертый блок
print_r($out);
echo "<br>";
?>
```

Каждый отдельный блок этой программы выводит на печать массив:

Первый блок:

```
Array ([0] => e [1] => g)
```

Второй блок:

```
Array ([0] => c [1] => d [2] => e)
```

Третий блок:

```
Array ([0] => e)
```

Четвертый блок:

```
Array ([0] => a [1] => b [2] => c)
```

Заключение

Все функции можно разделить на следующие группы:

- функции подсчета значений массива — позволяют производить подсчет значений и все, что с этим связано. К ним относятся : array_count_values(), array_sum(), count();
- операции вычисления матриц — позволяют производить вычисления. К ним относятся array_diff() , array_intersect (} И др.;
- функции возвращения: array_keys(), array_reverse(), array_value(), current(), each(), in_array(), array_search() И др.;
- применения вызовов: array_map (), array_wolk () и др.;
- функции объединения: array_merge(), array_merge_recursive() .;
- сортировка массивов: array_multisort(), arsort(), asort(), krsort(), ksort(), natsort(), rsort(), sort(), uasort(), uksort(), usort() и др.;
- вытеснение элементов из массива: array_pop(), array_shift(), array_push() и др.;
- получение элементов согласно внутреннему указателю массива: next(), pos(), reset(), prev();
- функции среза элементов массива: array_ slice().

Глава 21
Функции обнаружения орфографических ошибок

Во всех популярных текстовых редакторах встраиваются функции поиска и исправления ошибок. Уверены, вы уже успели ощутить данный сервис на себе — это очень приятно и удобно в использовании. Допустим, вам периодически необходимо обновлять Web-ресурс новостями, можно воспользоваться функциями проверки ошибок, просто подключив соответствующие библиотеки, словари и необходимые файлы для этого. Думаем, не стоит объяснять вам, на что это может повлиять. Прежде всего, если аудитория вашего ресурса состоит из серьезных людей, т. е. потенциальных клиентов вашей фирмы или же каких-либо научных сотрудников и т. д., то вопрос о грамотном написании новостей или каких-либо объявлений вообще должен отсутствовать. Серьезные фирмы — это прежде всего серьезное отношение к делу, а ваш Web-ресурс будет лицом вашего предприятия. Пожалуй, не каждому директору хотелось бы иметь Web-ресурс с ошибками. Для этого и были созданы в PHP функции контроля синтаксиса. В этой главе будут рассмотрены следующие вопросы:

- функции ispell, aspell и pspell;
- создание конфигурации;
- вызов файлов контроля синтаксиса;
- списки слов проверки орфографии и принципы работы с ними;
- функции непосредственной проверки орфографии;
- функция игнорирования слов определенной длины.

21.1. Функции ispell, aspell и pspell

Для проверки орфографии в PHP существуют функции ispell, aspell и pspell.

Функции ispell использовались на начальной стадии развития PHP.

Функции aspell были разработаны, чтобы в конечном счете полностью заменить ispell. aspell работают более быстро и имеют больше возможностей реализации необходимых задач. Недавние исследования показали, что эти функции намного эффективнее функций проверки орфографии редактора Microsoft World 97 и других распространенных редакторов. Функции также проверяют TeX и HTML-файлы и поддерживают проверку других языков во время выполнения.

Рекомендуется использовать aspell через библиотеку pspell, что позволяет работать с более новыми версиями aspell.

Сама по себе aspell работает с очень старыми библиотеками. Поэтому для более корректной работы предлагается использовать библиотеку функций pspell. Более подробно о принципах работы этих функции вы узнаете далее.

21.2. Краткий обзор концепций функций aspell

Полный обзор этих функции мы не будем делать, так как они уже устарели. В случае необходимости обратитесь к документации по этому вопросу. Для понимания отличия этих функции от pspell необходимо знать наиболее распространенные способы их реализации на практике.

Функции aspell призваны решать задачи поиска ошибок в словах и предложениях. Принцип работы данных функций очень прост. Прежде всего в коде необходимо указать, с каким словарем вы собираетесь работать, т. е. является ли он русским или английским и т. д. Также вам необходимо воспользоваться библиотекой, находящейся по адресу: `http://aspell.sourceforge.net/`. Когда все проблемы с библиотекой aspell решены, предлагаем перейти к изучению основных функций aspell:

- aspell_new,
- `aspell_check`,
- `aspell_check_raw`,
- `aspell_suggest`.

ВНИМАНИЕ

Если вам придется изучать какой-либо код программы, в которой используются функции проверки правописания, то, чтобы отличить функции aspell, просто посмотрите на первое слово. Как вы уже заметили, эти функции начинаются с одинакового слова aspell. Это делает его более систематизированным и понятным в процессе изучения того или иного кода программы.

Чтобы произвести вызов нового словаря, необходимого вам в работе, следует воспользоваться функцией aspell_new(). Ее синтаксис:

```
int aspell_new(string master, string personal)
```

Функция открывает указанный в ней словарь, а также производит возврат целого значения, равного идентификатору на ссылаемый словарь, для использования его в других функциях aspell. Библиотека может также быть создана самостоятельно, поэтому параметры функции aspell_new имеют две строки: string master, string personal. Обратите внимание, что объявление словаря производится в кавычках" ".

Функция aspell_new работает в PHP 3—3.0.7 и PHP 4.

Пример использования функции aspell_new():

```
<?
$ident = aspell_new("english");
?>
```

Функция вызывает английский словарь, при этом присваивает описанный выше идентификатор переменной $ident. После вызова словаря можно непосредственно приступать к проверке правильности написания слов и предложений. Для этого воспользуемся функцией aspell_check(). Функция имеет следующий синтаксис:

```
boolean aspell_check(int dictionary_link, string word)
```

Возвращает значение либо true, либо false. Как правило, используется в операторах условия. Значение true возвращается в том случае, если указанное нами слово, в данном случае за это отвечает параметр string word в функции aspell_check(), будет верным, т. е. не будет содержать ошибок. В противном случае получаем значение false. Например:

```
<?
$ident = aspell_new("english");
if (aspell_check($ident, "dogg"));
{
echo "Слово написано верно и не содержит никаких ошибок";
}
```

```
else
{
echo "Приведенное слово содержит ошибки";
}
?>
```

Результатом работы данной программы будет фраза, выведенная браузером:

Приведенное слово содержит ошибки

Аналогичную задачу выполняет функция `aspell_check_raw()`, единственное ее отличие от `aspell_check()` состоит в том, что `aspell_check_raw()` производит поиск ошибки без изменения регистра слова или же без попытки автоматически исправить ошибку. Функция так же, как и `aspell_check()`, возвращает true в случае правильного написания слова и `false` — неправильного, т. е. с ошибкой. Например:

```
<?
$ident = aspell_new("english");
if (aspell_check_raw($ident, "dog"));
{
echo "Слово написано верно и не содержит никаких ошибок";
}
else
{
echo "Приведенное слово содержит ошибки";
}
?>
```

Результатом работы данной функции будет предложение:

Слово написано верно и не содержит никаких ошибок.

Чтобы более подробно разобраться в различиях работы данных функций, поэкспериментируйте самостоятельно.

После того как проверена правильность слова, можно воспользоваться функцией, которая предложит нам массив значений правильного написания указанного нами слова. Именно за этот процесс отвечает функция `aspell_suggest()`. Ее синтаксис:

```
array aspell_suggest(int dictionary_link, string word)
```

Как видно из синтаксиса, функция возвращает массив значений используемого нами слова. Например:

```
<?
$ident = aspell_new("english");
if (aspell_check($ident, "test"»
{
echo "Слово написано без ошибки";
}
else
{
$correct = aspell_sugest ($ident, "test");
for ($i =0; $i<count($correct); $i++)
{
echo "Слова, предлагаемые для исправления:". $correct[$i]. "<br>";
}
}
?>
```

В ходе выполнения скрипта при неправильном написании слова будет выведен список возможных исправлений, в противном случае — фраза: «Слово написано без ошибки», что и произойдет в нашем случае. Попробуйте изменить скрипт так, чтобы он производил выполнение блока программы, который заключен между фигурными скобками оператора else. Действительно, один из способов будет верным, если вы просто вместо правильного слова напишите слово, содержащее ошибку, либо же просто произведете выполнение программы, изменив строку оператора. условия на строку, вызывающую вместо true false. Это делается введением в начало строки знака ! (отрицания):

```
if (!aspell_check($ident, "test"))
```

Проделайте эти операции самостоятельно и понаблюдайте, что в результате произойдет. На этом рассмотрение функция aspell закончено, перейдем к более распространенным функциям pspell.

21.3. Особенности функций pspell

Функции pspell являются широко распространенными и имеют намного более развитую систему применения на практике. То, что они включают возможность работы с функциями aspell, также влияет на их популярность В случае выбора между функциями aspell и pspell, как правило, используют последнюю.

Также это связано и с библиотекой, подключаемой для возможности работы с данными функциями. Ее можно найти по адресу `http://pspell.sourceforge.net/`. В процессе установки PHP необходимо также указать опцию — `with-pspell [=dir]`. В противном случае вам придется сделать это самостоятельно, отредактировав файл конфигурации.

Функции pspell получили распространение в PHP 4–4.0.2. Прежде всего это говорит о том, что данные функции сравнительно недавно введены для применения, что позволило им занять лидирующее место, так как именно при создании новых функций принято учитывать старые ошибки и нюансы при реализации поставленной задачи.

21.4. Создание конфигурации

В рассмотренной нами ранее функции aspell для подключения словаря используется функция `aspell_new()`. Количество ее параметров, указываемых при подключении, было минимально, чего нельзя сказать о новой функции `pspell_config_create()`. Данная функция в принципе аналогична aspell_new (), единственное, что ее отличает, — расширенная система задания параметров при вызове словаря. Это, конечно, и позволило завоевать популярность и получить значительное преимущество по отношению к функциям aspell. Возможно, что эти параметры могут и не использоваться при вызове функции, но в случае конкретного подхода описываемой задачи просто очень удобно манипулировать и использовать те параметры, которые вам необходимы. Функция pspell_config_create () имеет следующий синтаксис:

```
int pspell_config_create(string language [, string spelling [, string
jargon [, string encoding [, int mode]]]])
```

После создания необходимой конфигурации вы можете использовать любые функции `pspell_config_*` перед запросом `pspell_new_config()`. Это прежде всего позволит вам воспользоваться преимуществами некоторых расширенных функциональных возможностей.

Параметр `language` — языка— состоит из двух значений ISO 639 кода языка (кодировки) и необязательных двух значений ISO 3166 — кода страны после черточки или символа подчеркивания.

Параметр `spelling` — орфографии — требуют указания языка подключаемой библиотеки. Например, для english также известны такие значения, как `american`, `british` и Canadian.

Параметр jargon — циркона — содержит дополнительную информацию, необходимую для того, чтобы можно было отличить два различных списка слов, которые имеют тот же самый язык и параметры орфографии.

Параметр encoding — кодирования — кодирование нужных слов. Допустимые значения — `utf-8`, `iso8859- *`, `koi8-r`, `viscii`, `cp1252`, machine unsigned 16, machine unsigned 32.

Функция проверки орфографии работает в трех режимах:

- PSPELL_FAST — быстрый способ (наименьшеечисло предложений);
- PSPELL_NORMAL — штатный режим (большое количествопредложений);
- PSPELL_BAD_SPELLERS — медленный способ (много предложений).

Функция pspell_config_create () возвращает целое значение, которое соответствует идентификатору, указывающему на тот или иной подключаемый словарь. В дальнейшем при выполнении программы производится манипуляция непосредственно с самой функцией, а именно с этим самым идентификатором. Приведенный пример показывает, каким способом можно подключить английский словарь для проверки орфографических ошибок:

```
$ident = pspell_config_create("en", " ", " ", " ", PSPELL_FAST);
```

Конечно же, для начала полноценной работы проверки правильности написания английского текста понадобится далеко не одна функция. Функция pspell_config_create () позволила нам просто указать конфигурацию словаря. После того как мы определили параметры для полного принципа задания всех необходимых условий проверки орфографии, нам также необходимо будет воспользоваться функциями pspell_config_personal () и pspell_config_repl (). (см. п. 21.5).

Для задания режима способа работы используется функция pspell_config_mode (). Ее синтаксис:

```
int pspell_config_mode(int dictionary_link, int mode)
```

Использовать данную функцию следует только после определения конфигурации. Это очень важный аспект, так как эта функция использует описанный идентификатор. В нашем примере это значение переменной $ident. Именно это значение необходимо вставить в поле функции pspell_config_mode(), определяемое как dictionary_link. Вместо mode записывается тот режим работы, который на данном этапе обработки орфографии является наиболее предпочтительным для вас (PSPELL_FAST, PSPELL_NORMAL или PSPELL_BAD_SPELLERS).

Как правило, функция pspell_config_mode() используется перед запросом pspell_new_config (). Эта функция определяет, как много предложений будет возвращено функцией pspell_suggest (). Например:

```
<?
$ident = pspell_config_create("en");
pspell_config_mode($ident, PSPELL_NORMAL);
$link = pspell_new_config($ident);
pspell_check($link, "thedog");
?>
```

В нашем примере задается штатный режим работы строкой pspell_config_mode($ident, PSPELL_NORMAL). Этот же режим можно задать также при помощи функции pspell_config_create () следующим образом:

```
$ident = pspell_config_create("en", " ", " ", " ", PSPELL_NORMAL);
```

21.5. Вызов файлов контроля синтаксиса

После того как вы научились определять конфигурацию для присоединяемого словаря, перейдем к описанию способов присоединения файлов контроля правописания. Рассмотрим две наиболее распространенные функции:

- pspell_config_personal (),
- pspell_config_repl ().

Чтобы подключить собственный список слов, необходимо воспользоваться функцией pspell_config_personal (). Данная функция имеет синтаксис, аналогичный функции pspell_config_repl():

```
int pspell_config_repl(int dictionary_link, string file)
```

Как видно, функция возвращает целое значение, параметр dictionary_link необходим для возвращаемого значения функции pspell_config_create (). Строка string file соответствует пути, указывающему на файл, который содержит персональный список слов (в случае функции pspell_config_personal), и на файл, содержащий список слов, который предъявляется в случае обнаружения какой-либо ошибки (используется для функции pspell_config_repl ()). Приведенный пример показывает принцип работы данных функций на практике:

```
<?
$ident = pspell_config_create("en");
pspell_config_personal($ident, "/my_archive/dict/seller.pws");
pspell_config_repl($pspell_config, "/my_archive/dict/seller.repl");
$link = pspell_new_config($ident);
pspell_check($link, "thecar");
7>
```

Опишем работу приведенного примера. Первая строка позволяет нам производить конфигурацию используемого словаря. После того как все проделано успешно, программа переходит к строке pspell_config_personal ($ident, "/my_archive/dict/seller.pws"), которая в свою очередь помогает установить доступ к персональному списку слов, установленных программистом. После этой операции мы определяем путь к файлу, содержащему ответ на запрос с ошибкой, т. е. в случае обнаружения какого-либо несовпадения слов со словами, определенными в словаре, интерпретатор PHP произведет автоматическое обращение

к файлу, содержащему замены этих слов. Постом происходит выполнение строки `$link = pspell_new_config ($ident)`, которая позволяет производить чтение нового указанного нами словаря с использование параметров, установленных основной функцией конфигурации. После этого происходит проверка указанного нами слова, в данном случае это «thecar», на правильность написания.

21.6. Списки слов проверки орфографии и принципы работы с ними

Как всякий язык, PHP позволяет программисту помимо использования зарезервированного словаря создание самостоятельных списков слов, при помощи которых в дальнейшем будут проверяться те или иные слова. Это очень полезно, когда вам необходимо контролировать, например, ограниченный перечень слов, известных заранее.

Рассмотрим следующие функции:

- `pspell_add_to_personal()`,
- `pspell_add_to_session()`,
- `pspell_clear_session()`,
- `pspell_config_save_repl()`,
- `pspell_new()`,
- `pspell_new_config()`,
- `pspell_new_personal()`,
- `pspell_save_wordlist()`.

Это не полный перечень функций работы со списками слов, но изучив основные принципы работы с этими функциями, вы без проблем сможете профессионально программировать.

В существующие списки слов можно не только удалять, но и добавлять необходимые слова. Это позволяет сделать функция `pspell_add_to_personal()`. Функция имеет следующий синтаксис:

```
int pspell_add_to_personal(int dictionary_link, string word)
```

dictionary_link — идентификатор функции конфигурации, word — вносимое слово.

Функция `pspell_add_to_personal()` позволяет добавлять слово в персональный список слов. Если, чтобы открыть словарь, вы использовали функции `pspell_new_config()` совместно с `pspell_config_personal()`, то в этом случае для сохранения списка слов можно воспользоваться функцией `pspell_save_wordlist()`.

ВНИМАНИЕ

Эта функция не будет работать, если у вас не установлена библиотека pspell версии 11.2 или aspell.32.5 и позже.

Работу функции pspell_add_to_personal() можно изучить на следующем примере:

```
<?
$ident = pspell_config_create("en");
pspell_config_personal($ident, "/dict/sellere.pws");
$link = pspell_new_config($ident);
pspell_add_to_personal($link, "Sasha");
pspell_save_wordlist($link);
?>
```

Из примера видно, что при помощи функции pspell_add_to_personal() произойдет добавление слова Sasha в список слов, находящийся по пути /dict/sellere.pws, после чего произойдет операция сохранения данных и программа прекратитвыполнение.

Если вам необходимо произвести аналогичное дополнение нового слова к списку слов в текущей сессии, воспользуйтесь функцией pspell_add_to_session(). Функция аналогична принципу работы pspell_add_to_personal() и имеет такой же синтаксис. Единственное, что их отличает — pspell_add_to_session() производит дополнение в список листов в текущей сессии.

Помимо этого в текущем сеансе можно не только добавлять, но и удалять слова списка слов. Для этого используется функция pspell_clear_session().

Pspell_clear_session() очищает текущий сеанс. Список слов сеанса становится пустым, и, например, если вы попробуете сохранить слова с использованием функции pspell_save_wordlist(), ничего не случится. Обратите внимание например:

```
<?
$ident = pspell_config_create("en");
pspell_config_personal($ident, "/dict/seller.pws");
$link = pspell_new_config($ident);
pspell_add_to_personal($link, "Sasha");
pspell_clear_session($link);
pspell_save_wordlist($link);
?>
```

После выполнения примера слово Sasha не будет записано в наш список слов.

Рассмотрим еще одну функцию, представляющую для нас практический интерес. Функция pspell_save_repl() позволяет нам решать, сохранять ли список замены наряду со списком слов. Ее синтаксис:

```
int pspell_config_save_repl(int dictionary_link, boolean flag)
```

Функция pspell_config_save_repl() используется перед запросом pspell_new_config(). Это определяет, сохранит ли pspell_save_wordlist() слова замены наряду со списком слов. Обычно вместо этой функции принято использовать pspell_save_wordlist().

Чтобы произвести вызов нового словаря, необходимо воспользоваться функцией pspell_new(). Функция похожа на pspell_config_create(). Она оперирует практически аналогичными понятиями параметров, за исключением нового, добавленного параметра, увидеть который вы сможете из приведенного синтаксиса данной функции:

```
int pspell_new(string language [, string spelling [, string jargon
[, string encoding [, int mode]]]])
```

Помимо трех имеющихся, описанных нами ранее, добавлена еще одна разновидность режима работы:

- PSPELL_RUN_TOGETHER — словосочетания рассматриваются как одно целое, т. е, конструкция thecat будет являться правильным составом и никакой ошибки определяться не будет, хотя в этом случае должно содержаться место между двумя словами. Изменения этих параметров могут только воздействовать на результаты, возвращенные функцией pspell_check(). В свою очередь функция pspell_suggest() будет все еще возвращать предложения.

Пример работы функции pspell_new():

```
$link = pspell_new("en", "", "", "", (PSPELL_FAST|PSPELL_RUN_TOGETHER));
```

Строка приведенного примера позволяет понять, как производится задание тех или иных параметров. Параметр PSPELL_FAST позволяет производить быструю проверку, при этом режимом PSPELL_RUN_TOGETHER мы указали на то, что словосочетания, написанные вместе, например «thedog», не являются ошибкой и на них не стоит заострять внимание.

Для вызова нового словаря с основной конфигурацией используется функция pspell_new_config(). Функция сама по себе не задает конфигурацию, она позволяет производить обращение к ней с учетом уже определенной на начальном этапе создания скрипта. Функция имеет очень простой синтаксис:

```
int pspell_new_config(int config)
```

Другими словами, функция pspell_new_config() открывает новый словарь с установками, указанными в конфигурации, созданной при помощи функции pspell_config_create(), и изменяемый при помощи функции pspell_config_*(). Этот метод является более гибким и имеет все функциональные возможности, определенные функциями pspell_new() и pspell_new_personal().

ВНИМАНИЕ

Параметр конфигурации устанавливается один раз функцией pspell_config_create (). После этого происходит элементарное обращение к идентификатору конфигурации.

Пример реализации функции pspell_new_config ():

```
$ident = pspell_config_create("en");
pspell_config_personal($ident, "/dict/seller.pws");
pspell_config_repl($pspell_config, "/dict/seller.repl");
$link = pspell_new_config($ident);
```

Если вам необходимо вызвать новый словарь с персональным списком слов, необходимо воспользоваться функцией pspell_new_personal (). Синтаксис функции похож на синтаксис таких функций, как pspell_config_create () и pspell_new (). Единственное, что отличает pspell_new_personal от приведенных — задача, для выполнения которой и была создана данная функция. Синтаксис функции pspell_new_personal ():

```
int pspell_new_personal(string personal, string language [, string
spelling [, string jargon [, string encoding [, int mode]]]])
```

Функция pspell_new_personal () позволяет открывать новый словарь с персональным списком слов и возвращает идентификатор связи словаря для использования в других функциях pspell. Список слов может изменяться и быть сохранен при помощи функции pspell_save_wordlist (). Однако значения слов для замены не сохраняются. Чтобы их сохранять, необходимо создать конфигурацию, используя pspell_config_create (), установить персональный файл списка слов при помощи функции pspell_config_personal (), установить файл для замены слов функцией pspell_config_repl () и вызвать новый словарь при помощи функции pspell_new_config ().

В функции pspell_new_personal () добавлен новый параметр (personal). Персональный параметр (personal) устанавливает файл, в котором происходит добавление слов к персональному списку, после этого файл будет сохранен. Это должно быть абсолютное имя файла, начинающееся с символа /, потому что иначе это будет пониматься как $HOME, который является «корневым» (/root) для большинства систем, и скорее всего не тот путь, который вам необходим.

Приведем пример, позволяющий правильно производить работу с данной функцией:

```
$ident = pspell_new_personal("/dict/seller.pws",
"en", "", "", "", PSPELL_FAST|PSPELL_RUN_TOGETHER));
```

21.7. Функции непосредственной проверки орфографии

Если нужно проверить слова на правильность написания, используют функцию pspell_check(). Функция аналогична aspell_check() и имеет следующий синтаксис:

```
boolean pspell_check(int dictionary_link, string word)
```

В случае правильного написания функция возвращает значение, равное true в противном случае false. Например:

```
<?
$ident = pspell_new("en") ;
if (pspell_check($ident, "catt")) {
echo "Слово написано без ошибок ";
} else {
echo "Извините, слово содержит ошибку";
}
?>
```

Результатом выполнения этого примера будет строка «Извините, слово содержит ошибку», так как функция произведет возврат значения false.

Существует еще одна функция pspell_suggest(), полностью аналогичная aspell_suggest(), с принципами работы которой мы подробно ознакомились ранее (см. п. 21.2).

21.8. Функция игнорирования слов определенной длины

Иногда возникает необходимость проигнорировать, например, слова, имеющие определенное количество символов.

Если нужно пропустить проверку слов, состоящих из определенного количества символов, пользуются функцией pspell_config_ignore(). Ее синтаксис:

```
int pspell_config_ignore(int dictionary_link, int n)
```

Количество символов, содержащихся в игнорируемом слове, указывается вместо параметра int n. Значение числа n может быть любым. Однако используйте значения параметра n в разумных пределах, чтобы не получить каких-либо недоразумений в результате работы вашего скрипта.

Функция pspell_config_ignore() должна использоваться перед запросом функции pspell_new_config(). Эта функция позволяет пропускать короткие слова,

т. е. не производить их проверку. Количество символов, имеющихся в словах, которые необходимо пропустить, указывается непосредственно в функции pspell_new_config ().

Например:

```
$ident = pspell_config_create("en");
pspell_config_ignore($ident, 7);
$link = pspell_new_config($ident);
pspell_check($pspell_link, "abcdre");
```

В примере задается проверка правильности написания сочетания символов abcdre. Как видно, это сочетание символов не несет какого-либо логического смысла. Тогда функция pspell_check () должна соответствующе среагировать.

Заключение

Из этой главы вам необходимо запомнить, что существует два основных вида функции проверки орфографии в PHP:

- aspell,
- pspell.

Наибольшее распространение получила функция pspell. Данный вид функций проверки правописания включает в себя все принципы работы функции aspell. Поэтому, установив библиотеку функции pspell для работы дополнительно с функциями aspell, вам не понадобится устанавливать еще и библиотеку функций aspell. Вы просто можете пользоваться функциями aspell, используя при этом библиотеку функции pspell.

Глава 22

Математические функции и функции произвольной точности (BC)

Очень часто при написании кода программисту приходится сталкиваться с осуществлением каких-либо математических операций. Именно простые математические операции позволяют решать сложные задачи.

Язык программирования можно было бы считать не совсем полным в случае отсутствия возможности работы с математическими функциями. Принцип работы математических функций в PHP такой:

- функции работают только с диапазоном чисел типа long и double;
- в случае использования больших чисел следует обратиться к математическим функциями произвольной точности;
- приведенные ниже BC-функции задействованы только при условии, что PHP был скомпилирован в режиме — enable-bcmath, т. е. при включенных в конфигурацию функциях bcmath.

В этой главе рассмотрены следующие вопросы:

- математические константы и функции;
- функции произвольной точности (BC-функции).

22.1. Математические константы и функции

M_PI

Математическая константа, равная 3,14159265358979323846 (значение числа я).

Abs

Возвращает абсолютную величину числа. Если число с плавающей запятой, то возвращает число с плавающей запятой.

Синтаксис:

```
mixed abs(mixed number);
```

Acos

Возвращает арккосинус аргумента в радианах.

Синтаксис:

```
float acos(float arg);
```

Asin

Возвращает арксинус аргумента в радианах.

Синтаксис:

```
float asin(float arg);
```

Atan

Возвращает арктангенс аргумента в радианах.

Синтаксис:

```
float atan(floatarg.);
```

Atan2

Вычисляет арктангенс от двух переменных *x* и *y* аналогично вычислению арктангенса *y/x,* за исключением того, что знаки обоих аргументов используются для определения сектора результата.

Функция возвращает результат в радианах, находящихся между –PI и PI (включительно).

Синтаксис:

```
float atan2(float y, float x);
```

Base_convert

Возвращает строку, содержащую number, представленную по основанию tobase. Основание, в котором дается число which number, указывается в frombase. Основания frombase и tobase должны находиться в диапазоне от *2* до 36 включительно. Цифры в числах с основанием выше, чем 10, будут представлены буквами a—z со значениями a — 10, b — 11 и z — 36. Например:

```
$binary = base_convert($hexadecimal, 16, 2);
```

Синтаксис:

```
string base_convert(string number, int frombase, int tobase);
```

BinDec

Возвращает десятичный эквивалент двоичного числа, представленного аргументом binary_string.

BinDec конвертирует двоичное число в десятичное. Наибольшее число, которое может быть конвертировано — 31 битам, или 2 147 483 647 в десятичном виде.

Синтаксис:

```
int bindec(string binary_string);
```

Ceil

Округляет дробную часть к большему.

Синтаксис

```
int ceil(float number);
```

Возвращает следующее наибольшее целое значение number. Ceil () непродуктивно использовать для целых чисел.

СОВЕТ

Функция ceil () в PHP/FI *2* возвращала число типа float. Используйте $new = (double)ceil($number);, чтобы имитировать старое поведение функции.

Cos

Возвращает косинус аргумента в радианах.

Синтаксис:

```
float cos(float arg);
```

DecBin

Возвращает строку, содержащую двоичное представление аргумента number. Наибольшее число, которое может быть конвертировано — 2 147 483 647 в десятичном виде или 31 бит.

Синтаксис:

```
string decbin(int number);
```

DecHex

Возвращает строку, содержащую шестнадцатеричное представление аргумента number. Наибольшее число, которое может быть конвертировано, равно 2 147 483 647 в десятичном виде, или 7fffffff в шестнадцатеричном.

Синтаксис:

```
string dechex(int number);
```

DecOct

Возвращает строку, содержащую восьмеричное представление аргумента number. Наибольшее число, которое может быть конвертировано, равно 2 147 483 647 в десятичном виде, или 17 777 777 777 в восьмеричном.

Синтаксис:

```
string decoct(int number);
```

Exp

Возвращает число e, возведенное в степень arg.

Синтаксис:

```
float exp(float arg);
```

Floor

Округляет дробную часть к меньшему.

Синтаксис:

```
int floor(float number);
```

Возвращает предыдущее целое значение после number. Использование floor() нецелесообразно для целых чисел.

СОВЕТ

Функция floor () в PHP/FI 2 возвращала число типа float. Используйте $new = (double)floor($number);, чтобы имитировать старое поведение этой функции.

Getrandmax

Возвращает максимальную случайную величину, которая может быть возвращена вызовом функции rand ().

Синтаксис:

```
int getrandmax(void);
```

HexDec

Возвращает десятичный эквивалент числа, представленного аргументом hex_string. HexDec конвертирует шестнадцатеричную строку в десятичное число. Наибольшее число, которое может быть конвертировано, равно 7fffffff в шестнадцатеричном виде, или 2 147 483 647 в десятичном.

Синтаксис:

```
int hexdec(string hex_string);
```

Log

Возвращает натуральный логарифм от аргумента arg.

Синтаксис:

```
float log(float arg);
```

Log10

Возвращает логарифм по основанию 10 от аргумента arg.

Синтаксис:

```
float log10(float arg);
```

Max

Возвращает наибольшее число из перечисленных в параметрах.

Синтаксис:

```
mixed max(mixed arg1, mixed arg2, mixed argn);
```

Если первый элемент является массивом, max() возвращает максимальную величину массива. Если первый параметр — integer, string или double, то следует использовать как минимум два параметра, и в этом случае max () возвращает наибольшую из этих величин. Можно сравнивать неограниченное количество значений.

Если одна или более величин типа double, все остальные величины будут обращены к double и, соответственно, возвратится число типа double. Если ни одно из чисел не является double, то все будут обращены в целые и возвратится целое число.

Min

Возвращает наименьшее значение из указанных в аргументах.

Синтаксис:

```
mixed min(mixed arg1, mixed arg2, mixed argn);
```

Если первый параметр — массив, min () возвратит наименьшую величину массива. Если первый параметр — целое integer, string или double, следует указать не меньше двух параметров и min () возвратит наименьшую из них величину. Можно сравнивать неограниченно количество величин.

Если одна или более величин типа double, все остальные величины будут обращены к double и, соответственно, возвратится число типа double. Если ни одно из чисел не является double, то все будут обращены в целые и возвратится целое число.

Mt_rand

Генерирует наилучшее случайное число.

Синтаксис:

```
int mt_rand([int min], [int max]);
```

Множество генераторов случайных чисел, написанных на старой библиотеке libcs, имеют неясные или неизвестные характеристики и к тому же медленны. По умолчанию с функцией rand () PHP использует генератор случайных чисел, написанный на libc. Функция mt_rand () является его полной заменой. Она использует генератор случайных чисел с известными характеристиками (Mersenne Twister), который производит случайные числа, пригодные для использования в криптографии, и работает в четыре раза быстрее, чем средняя скорость, которую обеспечивает libc. Домашняя страница Mersenne Twister находится по адресу http://www.math.keio.ac.jp/~matumoto/emt.html.

Если функция вызывается без необязательных аргументов min и max, mt_rand () возвращает псевдослучайное число между 0 и RAND_MAX. Если нужно, например, получить случайное число между 5 и 15 (включительно), то можно использовать следующий вызов функции: mt_rand(5,15).

Не забудьте инициализировать генератор случайных чисел перед использованием функции mt_srand().

Mt_strand

Инициализирует генератор случайных чисел значением seed.

Синтаксис:

```
void mt_srand(int seed);
```

Например:

```
/* инициализируется количеством микросекунд с последней "целой"
секунды */
mt_srand((double)microtime()*1000000);
$randval = mt_rand();
```

Mt_getrandmax

Возвращает максимальную случайную величину, которая может быть возвращена вызовом функции mt_rand().

Синтаксис:

```
int mt_getrandmax(void);
```

Number_format

Форматирует число со сгруппированными разрядами (например, 1000000).

Синтаксис:

```
string number_format(float number, int decimals, string dec_point,
string thousands_sep);
```

number_format() возвращает форматированную версию числа number. Эта функция принимает один, два или четыре параметра (не три).

Если дан только один параметр, число number будет отформатировано без десятичных цифр, но с запятой между каждой группой разрядов.

Если дано два параметра, число number will будет отформатировано с десятичным знаком decimals с точкой впереди и запятой между каждой группой разрядов.

Если даны все четыре параметра, то число number будет отформатировано с десятичным знаком decimals, dec_point вместо точки перед десятичным знаком и thousands_sep вместо запятой между каждой группой разрядов.

OctDec

Синтаксис:

```
int octdec(string octal_string);
```

Возвращает **десятичный эквивалент** восьмеричного числа, представленного аргументом octal_string. Максимальное число, которое может быть конвертировано, равно 17 777 777 777, или 2 147 483 647 в десятичном виде.

Pi

Возвращает аппроксимированное значение числа л.

Синтаксис:

```
double pi(void );
```

Pow

Возвращает base, возведенное в степень exp.

Синтаксис:

```
float pow(float base, float exp);
```

Rand

Генерирует случайную величину.

Синтаксис:

```
int rand([int min], [int max]);
```

Если функция вызывается без необязательных параметров min и max, rand() возвращает псевдослучайную величину между 0 и RAND_MAX. При желании получить случайное число между 5 и 15 (включительно) используйте rand(5,15).

Не забудьте инициализировать генератор случайных чисел перед использованием srand(). Начиная с версии PHP 4.2.0 в вызове этой функции нет необходимости.

Round

Округляет число с плавающей запятой.

Синтаксис:

```
double round(double val);
```

Например:

```
$foo = round( 3.4 ); // $foo = 3.0
$foo = round( 3.5 ); // $foo = 4.0
$foo = round( 3.6 ); // $foo = 4.0
```

Sin

Возвращает синус аргумента.

Синтаксис:

```
float sin(float arg);
```

Sqrt

Возвращает квадратный корень аргумента.

Синтаксис:

```
float sqrt(float arg);
```

Srand

Инициализирует генератор случайных чисел.

Синтаксис:

```
void srand(int seed);
```

Например:

```
// инициализируется числом микросекунд с последней "целой" секунды
srand((double)microtime()*1000000);
$randval = rand();
```

Начиная с версии PHP 4.2.0 в вызове этой функции нет необходимости.

Tan

Возвращает тангенс аргумента.

Синтаксис:

```
float tan(float arg);
```

22.2. Функции произвольной точности (BC-функции)

Bcadd

Сложение двух чисел произвольной точности.

Синтаксис:

```
string bcadd(string левый операнд, string правый операнд, int
[масштаб]);
```

Прибавляет левый операнд к правому операнду и возвращает сумму типа string (строковая переменная). Факультативный параметр масштаб используется, чтобы установить количество разрядов после десятичной отметки в результате.

Bccotp

Сравнение двух чисел произвольной точности.

Синтаксис:

```
int bccomp(string левый операнд, string правый операнд, int
[масштаб]);
```

Сравнивает левый операнд с правым операндом и возвращает результат типа integer. Факультативный параметр масштаб используется для установки количества цифр после десятичной отметки, используемых при сравнении. При равенстве двух операндов возвращается значение 0. Если левый операнд больше правого операнда, возвращается +1, если левый операнд меньше правого операнда, возвращается –1.

Bcdiv

Деление для двух чисел произвольной точности.

Синтаксис:

```
string bcdiv(string левый операнд, string правый операнд, int
[масштаб]);
```

Делит левый операнд на правый операнд и возвращает результат. Факультативный параметр масштаб устанавливает количество цифр после десятичной отметки в результате.

Bcmod

Получение модуля левого операнда при помощи операнда модуль.

Синтаксис:

```
string bcmod(string левый операнд, string модуль);
```

Bcmul

Умножение для двух чисел произвольной точности.

Синтаксис:

```
string bcmul(string левый операнд, string правый операнд, int
[масштаб]);
```

Умножает левый операнд на правый операнд и возвращает результат. Факультативный параметр масштаб устанавливает количество цифр после десятичной отметки в результате.

Bcpow

Возведение одного числа произвольной точности в степень другого.

Синтаксис:

```
string bcpow(string x, string y, int [масштаб]);
```

Возводит *x* в степень *y*. Параметр масштаб может использоваться для установки количества цифр после десятичной отметки в результате.

Bcscale

Устанавливает масштаб по умолчанию для всех математических BC-функций.

Синтаксис:

```
string bcscale(int масштаб);
```

Bcsqrt

Возвращает квадратный корень операнда. Факультативный параметр масштаб устанавливает количество цифр после десятичной отметки в результате.

Синтаксис:

```
string bcsqrt(string операнд, int масштаб);
```

Bcsub

Вычитает одно число произвольной точности из другого.

Синтаксис:

```
string bcsub(string левый операнд, string правый операнд, int
[масштаб]);
```

Вычитает правый операнд из левого операнда и возвращает результат типа string. Факультативный параметр масштаб устанавливает количество цифр после десятичной отметки в результате.

Заключение

Познакомившись с математическими функциями PHP, вы понимаете, сколь широки просторы для вашей мысли в плане написания разнообразных скриптов с использованием конструкций математического расчета. Необходимо четко запомнить, что все эти математические функции работают только с диапазоном чисел типа long и double. Если вам нужно использовать большие числа, применяйте математические функции произвольной точности. Приведенные BC-функции работают только при условии, что PHP был скомпилирован в режиме `--enable-bcmath`, т. е. при включенных в конфигурацию функциях `bcmath`. Если у вас возникнут какие-либо трудности, прочтите главу еще раз и обратите внимание на синтаксис написания нужной вам функции. Когда дело касается таких далеко не самых простых функций, довольно легко допустить ошибку.

Глава 23
Функции даты/времени и работы с календарем

Календарное расширение в PHP представляет серию функций, которая упрощает преобразование между разными календарными форматами. Посредником преобразования является дневной Юлианский счет — самый старый счет дней (появился около 4000 до н.э.). Чтобы менять числа между разными календарными системами, вы должны сначала преобразовать вашу дату в дневной Юлианский счет, а только затем в календарную систему по вашему выбору.

ВНИМАНИЕ

Дневной Юлианский счет отличается от Юлианского календаря.

Эта глава состоит из следующих тем:

- функции работы с календарем;
- функции даты и времени.

23.1. Функции работы с календарем

С помощью этих функций вы сможете создавать для своих страниц интересные интерактивные скрипты.

JDToGregorian

Преобразовывает дневной Юлианский счет в Григорианскую дату в формате «месяц/день/год».

Синтаксис:

```
string jdtogregorian(int julianday);
```

GregorianToJD

Преобразовывает Григорианскую дату на дневной Юлианский счет.

Синтаксис:

```
int gregoriantojd(int month, int day, int year);
```

Правильный диапазон для Григорианского календаря — с 4714 г. до н. э. до 9999 г. н. э. Хотя это программное обеспечение может оперировать с датами в обратном порядке до 4714 г. до н.э., такое использование может быть бесполезно и незначительно. Григорианский календарь был учрежден 15 октября 1582 г. (5 октября 1582 г. в Юлианском календаре). Некоторые страны еще очень долго не принимали его. Напри-

мер, Великобритания ввела преобразования в 1752 г., СССР — в 1918 г., Греция — в 1923 г. Европейские страны использовали Юлианский Календарь до Григорианского. Рассмотрим пример:

```
<?php
$jd = GregorianToJD(10,11,1970);
echo("$jd\n");
$gregorian = JDToGregorian($jd);
echo("$gregorian\n");
?>
```

JDToJulian

Преобразовывает дату Юлианского календаря в дневной Юлианский счет в формате «месяц/день/год».

Синтаксис:

```
string jdtojulian(int julianday);
```

JulianToJD

Преобразовывает дату Юлианского календаря в дневной Юлианский счет.

Синтаксис:

```
int juliantojd(int month, int day, int year);
```

Правильный диапазон для Юлианского календаря — с 4713 г. до н.э. до 9999 г. н.э. Хотя это программное обеспечение может оперировать с датами в обратном порядке до 4713 г. до н.э., такое использование может быть не нужно. Календарь был создан в 46 г. до н.э., но не стабилизировался до 4 ст. н.э. Начало года различно у некоторых народов — не все соглашаются, что январь — это первый месяц.

JDToJewish

Преобразовывает дневной Юлианский счет в Еврейский календарь.

Синтаксис:

```
string jdtojewish(int julianday);
```

JewishToJD

Преобразовывает дату в Еврейском календаре на дневной Юлианский счет.

Синтаксис:

```
int jewishtojd(int month, int day, int year);
```

Еврейский календарь использовался несколько тысячелетий, но в течение начального периода не было никакой формулы, чтобы определить начало месяца. Новый месяц начинался, когда было замечено полнолуние.

JDToFrench

Преобразовывает дневной Юлианский счет во Французский Республиканский календарь.

Синтаксис:

```
string jdtofrench(int month, int day, int year);
```

FrenchToJD

Преобразовывает дату Французского Республиканского календаря в дневной Юлианский счет.

Синтаксис:

```
int frenchtojd(int month, int day, int year);
```

Эта программа преобразовывает даты в диапазоне от 1 до 14 (Григорианские даты с 22 сентября 1792 г. до 22 сентября 1806 г.). Это покрывает тот период, когда календарь использовался.

JDMonthName

Возвращает название месяца.

Синтаксис:

```
string jdmonthname(int julianday, int mode);
```

Параметр mode сообщает функции, в какой календарь нужно преобразовать дневной Юлианский счет (табл. 23.1).

Таблица 23.1. Календарные способы

Способ	Название	Значение
0	Григорианский (сокращенное)	Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
1	Григорианский	January, February, March, April, May, June, July, August, September, October, November, December
2	Юлианский (сокращенное)	Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
3	Юлианский	January, February, March, April, May, June, July, August, September, October, November, December
4	Еврейский	Tishri, Heshvan, Kislev, Tevet, Shevat, AdarI, AdarII, Nisan, Iyyar, Sivan, Tammuz, Av, Elul
5	Французский Республиканский	Vendemiaire, Brumaire, Frimaire, Nivose, Pluviose, Ventose, Germinal, Floreal, Prairial, Messidor, Thermidor, Fructidor, Extra

JDDayOfWeek

Возвращает день недели.

Синтаксис:

```
mixed jddayofweek(int julianday, int. mode);
```

Может возвратить string или int в зависимости от способа (mode) (табл. 23.2).

Таблица 23.2. Календарные недельные способы

Способ	Значение
0	Возвращает порядковый номер дня как int (0=воскресенье, 1=понедельник и т. п.)
1	Возвращает название дня недели как string (английское григорианское)
2	Возвращает string, содержащий сокращенный день недели (английский григорианский)

23.2. Функции даты и времени

Из этого параграфа вы узнаете, как написать обычный скрипт для показа времени или более сложные скрипты, где будут использованы функции даты и времени вместе с календарными.

Checkdate

Проверяет правильность даты/времени.

Синтаксис:

```
int checkdate (int month, int day, int year);
```

Возвращает true, если данная дата правильна, иначе false. Проверяет правильность даты, заданной аргументами. Дата считается правильной, если:

год между 1900 и 32 767 включительно;

месяц между 1 и 12 включительно.

День находится в диапазоне разрешенных дней данного месяца. Високосные годы учитываются.

Date

Формат локального времени/даты.

Синтаксис:

```
string date(string format, int timestamp);
```

Возвращает строку, отформатированную согласно данной строке, используя данную временную метку или текущее локальное время, если не задана временная метка.

В форматной строке должны использоваться следующие символы:

a — am или pm;

A — AMили PM;

d — день месяца, цифровой, 2 цифры (на первом месте нуль);

D — день недели, текстовый, 3 буквы, т. е. Fri;

F — месяц, текстовый, длинный, т. е. January;

h — час, цифровой, 12-часовой формат;

H — час, цифровой, 24-часовой формат;

i — минуты, цифровой;

j — день месяца, цифровой, без начальных нулей;

l (строчная L) — текстовый длинный день недели, т. е. Friday;

m — месяц, цифровой;

M — текстовый месяц, 3 буквы, т.е. Jan;

s — секунды цифр;

s — английский порядковый суффикс, текстовый, 2 символа, т. е. th, nd;

U — секунды с начала века;

Y — год, цифровой, 4 цифры;

w — день недели, цифровой, 0 означает воскресенье;

y — год, цифровой, 2 цифры;

z — день года, цифровой, т. е. 299.

Нераспознанные символы в форматной строке будут печататься, как есть.

Пример 23.1. Использование функции date ()

```
print(date( "l dS of F Y h:i:s A" ));
print("July 1, 2000 is on a ". date("l", mktime(0,0,0,7,1,2000)));
```

Функции date() и mktime() возможно использовать вместе, для того чтобы найти даты в будущем или прошлом.

Пример 23.2. Использование функций date () и mktime()

```
$tomorrow = mktime(0,0,0,date("m") ,date("d")+1,date("Y"));
```

```
$lastmonth = mktime(0,0,0,date("m")-1,date("d"), date("Y"));
$nextyear = mktime(0,0,0,date("m"), date("d", date("Y")+1);
```

Чтобы отформатировать даты на других языках, надо использовать функции `setlocale()` и `strftime()`.

Strftime

Форматирует локальное время согласно установкам locale.

Синтаксис:

```
string strftime(string format, int timestamp);
```

Возвращает строку, отформатированную согласно данной форматной строке, используя данную временную метку или текущее локальное время, если метка не задана. Названия месяцев, недель и т. д., зависящие от языка строки и связанные с текущим locale, устанавливаются с помощью setlocale().

В форматной строке следует использовать следующие спецификаторы преобразований:

%a — сокращенное название дня недели согласно текущему locale;

%A — полное название дня недели согласно текущему locale;

%Ь — сокращенное название месяца согласно текущему locale;

%в — полное название месяца согласно текущему locale;

%c — предпочтительное представление даты и времени для текущего locale;

%d — день месяца как десятичное число (в диапазоне от 0 до 31);

%H — час как десятичное число в 24-часовом формате (в диапазоне от 00 до 23);

%I — час как десятичное число в 12-часовом формате (в диапазоне от 01 до 12);

%j — день года как десятичное число (в диапазоне от 001 до 366);

%m — месяц как десятичное число (в диапазоне от 1 до 12);

%M — минуты как десятичное число;

%P — am либо pm согласно текущему времени или соответствующие строки для текущего locale;

%s — секунды как десятичное число;

%U — номер недели текущего года как десятичное число начиная с первого воскресенья в качестве первого дня первой недели;

%W — номер недели текущего года как десятичное число начиная с первого понедельника в качестве первого дня первой недели;

%w — день недели как целое число, воскресенье — нулевой день;

%x — предпочитаемое представление даты для текущего locale, не включающее время;

%x — предпочитаемое представление времени для текущего locale, не включающее дату;

%y — год как десятичное число без столетия (в диапазоне от 00 до 99);

%Y — год как десятичное число, включая столетие;

%Z — временная зона либо название или сокращение;

%% — символ %.

Пример 23.3. Использование функции strftime()

```
setlocale("LC_TIME", "C");
print(strftime("%A in Finnish is "));
setlocale("LC_TIME", "fi");
print(strftime("%A, in French "));
setlocale("LC_TIME", "fr");
print(strftime("%A and in German "));
setlocale("LC_TIME", "de");
print(strftime("%A.\n"));
```

Пример будет работать, если у вас установлены соответствующие locale.

Getdate

Получает информацию о дате/времени.

Синтаксис:

```
array getdate(int timestamp);
```

Возвращает ассоциативный массив, содержащий информацию о дате со следующими элементами:

"seconds" — секунды;

"minutes" — минуты;

"hours" — часы;

"mday" — день месяца;

"wday" — день недели, цифровой;

"mon" — месяц, цифровой;

"year" — год, цифровой;

"yday" — день года, цифровой, т.е. 299;

"weekday" — день недели, текстовый, полный, т. е. Friday;

"month" — месяц, текстовый, полный, т. е. January;

Gmdate

Форматирует GMT/CUT время/дату.

Синтаксис:

```
string gmdate(string format, int timestamp);
```

Аналогичная функции date(), за исключением того, что время возвращается в Гринвичском формате Greenwich Mean Time (GMT). Например, при запуске в Финляндии (GMT + 0200) первая строка примера 23.4 напечатает «Jan 01 1998 00:00:00», в то время как вторая строка — «Dec 31 1997 22:00:00».

Пример 23.4. Использование функции gmdate ()

```
echo date("M d Y H:i:s",mktime(0,0,0,1,1,1998));
echo gmdate("M d Y H:i:s",mktime(0,0,0,1,1,1998));
```

Mktime

Получает временную метку UNIX для даты.

Синтаксис:

```
int mktime(int hour, int minute, int second, int month, int day,
int year);
```

ВНИМАНИЕ

Обратите внимание на необычный порядок аргументов, который отличается от порядка аргументов *в* вызове функции mktime () из UNIX и выдает ошибку при неправильно заданных параметрах (см. ниже). Это очень часто встречающаяся ошибка в скриптах.

Mktime

Возвращает временную метку Unix согласно данным аргументам. Эта временная метка является целым числом, равным количеству секунд между точкой отсчета Unix (1 января 1970г.) и указанным временем.

Аргументы могут быть опущены справа налево; каждый опущенный таким образом аргумент будет установлен в текущую величину согласно локальной дате и времени.

Mktime полезна при арифметических действиях с датой и ее проверкой, она автоматически вычисляет корректную величину для вышедших за границы параметров. Например, каждая из следующих строк возвращает строку «Jan-01-1998».

Пример 23.5. **Применение функции mktime ()**

```
echo date("M-d-Y", mktime(0,0,0,12,32,1997));
echo date("M-d-Y", mktime(0,0,0,13,1,1997));
echo date("M-d-Y", mktime(0,0,0,1,1,1998));
```

Gmmktime

Получает временную метку UNIX для даты в GMT.

Синтаксис:

```
int gmmktime (int hour, int minute, int second, int month, int day, int year);
```

Идентична mktime (), за исключением передаваемых параметров, представляющих дату в GMT.

Time

Возвращает текущую временную метку UNIX.

Синтаксис:

```
int time (void);
```

Возвращает текущее время, измеренное в секундах с эпохи Unix (1 января 1970 г. 00:00:00 GMT).

Microtime

Возвращает текущую временную метку UNIX в микросекундах.

Синтаксис:

```
string microtime (void);
```

Возвращает строку «msec sec», где sec — текущее время, измеренное в секундах с эпохи Unix (1 января, 1970 г. 0:00:00 GMT), а msec — часть в микросекундах. Эти функции доступны только в операционных системах, поддерживающих системный вызов gettimeofday().

23.3. Практическое применение функций даты и времени

Познакомившись с функциями, вам, наверно, не терпится увидеть, как все это можно применить на практике и как вообще строятся подобные скрипты. Для начала можно взять наиболее простой и популярный скрипт показа времени на вашей Web-страничке. Реализовать это на PHP достаточно легко. Например:

```
<?
$ch = Date("H"); // читаем показания часов
$mi = Date("i"); // читаем показания минут
$h1 = "час"; // определяем переменные для часов
$h2 = "часа";
$h3 = "часов";
$m1 = "минут"; // определяем переменные для минут
$m2 = "минута";
$m3 = "минуты";
// проверяем, какое из слов подходит к показаниям часов
if ($ch == 1 or $ch == 21) { $h = $h1; } else {
if ($ch > 1 and $ch <= 4 or $ch > 20 and $ch <= 24) { $h = $h2; } else { $h = $h3; } }
// проверяем, какое из слов подходит к показаниям минут
if ($mi == 0 or $mi > 4 and $mi <= 20 or $mi > 24 and $mi <= 30 or $mi > 34 and $mi <= 40 or $mi > 44 and $mi <= 50 or $mi > 54 and $mi <= 60) { $m = $m1; } else {
if ($mi == 1 or $mi == 21 or $mi == 31 or $mi == 41 or $mi == 51) { $m = $m2; } else { $m = $m3; } }
// выводим результат на экран
echo "Сейчас $ch $h, $mi $m";
?>
```

Результат выполнения программы приведен на рис. 23.1.

С большинством того, что демонстрирует этот скрипт, вы уже знакомы по предыдущим примерам, но кое-что новое здесь есть. В первых же строчках скрипта определяется время. Сначала считываются показания часов, затем минут. Эти числа, соответственно, попадают в указанные нами переменные и уже могут быть выведены на экран. Но все не так просто. Если сделать логический вывод в соответствии с законами русского языка, то если 1, то час, а если 2, то часа, и т. д. В следующих

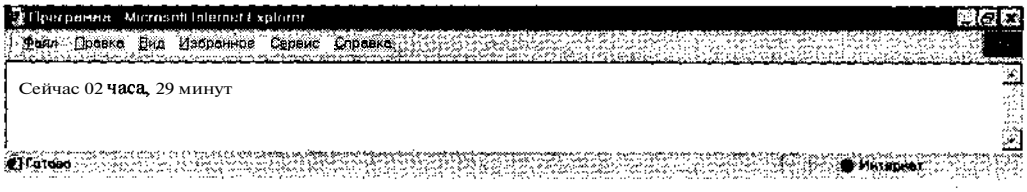


Рис. 23. 1. Результат выполнения скрипта

строках скрипта определяются слова для дальнейшего использования. Подходят три варианта — час, часов, часа. Считанное показание текущего времени сохранено в переменной $ch, и работать будем именно с ней. В первой строке проверки смотрим, равно ли $ch единице или 21. Не трудно догадаться, что это те числа, при которых следует поставить слово час. Если да, то переменной $h присваивается значение $hl. То же самое с помощью логического оператора or мы проделываем с оставшимися двумя переменными.

С минутами дело обстоит немного сложнее, ввиду того что минут по количеству больше, чем часов, но алгоритм уже известен и вывести на экран нужное слово труда не составляет.

Пример 23.6. Определение времени суток

```
<?php
$h=date('H');
if ($h>=6 && $h<=11) echo "Доброе утро!";
if ($h>=12 && $h<=17) echo "Добрый день!";
if ($h>=18 && $h<=23) echo "Добрый вечер!";
if ($h>=24 && $h<=5) echo "Доброй ночи!";
?>
```

Этот простой скрипт может выводить на вашей странице приветствие, зависящее от времени суток. Конечно, цифры желаемого времени вы можете проставлять любые, все зависит от вашего личного понятия дня и ночи.

Пример 23.7. Вывод даты

```
<?php
// определяем массив для месяцев
$q[]="";
$q[]="января";
$q[]="февраля";
$q[]="марта";
$q[]='апреля";
$q[]="мая";
$q[]="июня";
$q[]="июля";
$q[]="августа";
$q[]="сентября";
```

```
$q[]="октября";
$q[]="ноября";
$q[]="декабря";
// определяем массив для дней недели
$e[0]="воскресенье";
$e[1]="понедельник";
$e[2]="вторник";
$e[3]="среда";
$e[4]="четверг";
$e[5]="пятница";
$e[6]="суббота";
// считываем месяц
$m=date('m');
if ($m=="01") $m=1;
if ($m=="02") $m=2;
if ($m=="03") $m=3;
if ($m=="04") $m=4;
if ($m=="05") $m=5;
if ($m=="06") $m=6;
if ($m=="07") $m=7;
if ($m=="08") $m=8;
if ($m=="09") $m=9;
// считываем день недели
$we=date('w');
// считываем число
$chislo=date('d');
// извлекаем день из недели
$den_nedeli = $e[$we];
// извлекаем значение месяца
$mesyac = $q[$m];
echo "Сегодня ".$chislo." ".$mesyac.", ".$den_nedeli;
?>
```

Рассмотрим, как работает этот скрипт. Сначала нам нужно определить два массива, в которых будут храниться соответственно русское название месяца и русское название дня недели. Месяц не может быть нулевым, поэтому нам нужно позаботиться о вводе элемента массива с нулевым индексом. Если индекс массива не указан, он принимается равным внутреннему указателю. Если массив пусти еще не определен, внутренний указатель находится на первом элементе (имеющем индекс нуль). Ввод нового элемента массива перемещает внутренний указатель на единицу вверх, и таким образом обеспечивается последующий ввод значения массива в ячейку массива, имеющую индекс на единицу больший, чем предыдущая. В принципе, можно обеспечить ввод данных в массив разными способами. Но указанный здесь — самый легкий. Мы просто присваиваем поочередно нужные нам данные элементам массива, и таким образом заполняем его. Точно также и со вторым массивам. Только тут уже индекс нужных ячеек массива указывается явно, и внутренний указатель устанавливается на тот индекс, который задан, а после ввода значения перемещается вверх на единицу. Разница между этими двумя методами в том, что если массив уже был определен ранее и индекс при вводе не указан, заполнится ячейка массива, на которой находится внутренний указатель. А он ведь может стоять и в конце! А если индекс указан явно, внутренний указатель перемещается на его значение и запись происходит в явно указанную ячейку. Часто массивы бывают гораздо длиннее приведенных в примере, и удобнее воспользоваться специальной функцией, которая позволяет считать указанный в ней файл и ввести все, что есть в этом файле, в массив. Причем разделителем считается перевод строки, что очень удобно. Синтаксис этой функции — `$имя массива = file ("имя файла");`.

Дальше формируется массив с указанным именем и значениями, соответствующими строкам файла.

Когда массивы определены, нужно считать номер месяца. Он записывается с ведущим нулем, если номер месяца менее десяти, и поэтому нам нужно позаботиться об его отсечении. Тут можно применить разные алгоритмы и методы, но мы просто сравним полученное решение с рядом заранее известных вариантов и изменим номер месяца на правильный без нуля. Хотя так делать неправильно — есть более корректные методы, например проверить полученное значение на ведущий нуль, и если проверка истинна (самый первый символ в строке — нуль), удалить первый символ.

Далее по ходу скрипта считываются день недели и число. С числом делать ничего не нужно, так как дата будет понятна всем, а вот день недели и месяц должны подвергнуться обработке. Извлекаем из введенного нами ранее массива день недели. Номер дня недели указывает на ячейку нашего массива, где хранится нужное русское имя, и таким образом мы в любом случае получим правильное значение. Меняется номер дня недели и номер (индекс) ячейки, из которой считывается значение. Причем тут в отличие от массива с именем месяца нулю соответствует воскресенье, что мы и учли при вводе массива дней недели. Точно такую же операцию проводим и для месяца. Его номер указывает на ячейку массива, где хранится правильное имя месяца на русском языке. А дальше — выводим результат на экран в произвольной форме.

Заключение

Как видите, в PHP довольно легко и просто получить нужную дату или время, достаточно знать правила описания специальных функции. Также следует запомнить, что в PHP наиболее часто используется функция date() в формате `$date=date('параметр');`. Параметров может быть несколько, разделяются они между собой запятой. И конечно, не стоит забывать о том, что каждый параметр в разных регистрах выполняет разные функции.

Глава 24
Функции работы с классами и объектами

Детально изучив классы, а в частности принципы построения, способы обращения и вызова, перейдем непосредственно к рассмотрению вопроса, связанного с функциями работы с классами. Прежде всего эти функции созданы для того, чтобы получать информацию о существующих классах и их объектах. Благодаря этим функциям вы можете получить название класса, которому принадлежит тот или иной объект, а также его свойства и методы. Данные функции могут определить не только принадлежность классу его объекта, но также и его происхождение, т. е. какой класс является объектным распространением класса. Мы рассмотрим сначала функции вызова методов пользователя, выдаваемых массивом параметров, а затем непосредственно основные функции классов и объектов. Благодаря такой последовательности изложения материала вы сможете понять принципы программирования классов и совершения операций над ними. Эта глава состоит из следующих тем:

- вызов методов пользователя, выдаваемых массивом параметров;
- вызов методов пользователя классов;
- проверка классов;
- возврат параметров класса;
- возврат массива параметров объекта;
- определение существующих классов;
- программирование при помощи функций работы с классами.

24.1. Вызов методов пользователя, выдаваемых массивом параметров

После того как мы научились создавать функции, производить вызов этих функции, используя необходимые параметры, перейдем к автоматическому способу вызова методов пользователя. Чтобы произвести обращение к функции, содержа-

щей множество параметров, используют функцию call_user_func_array(). Эта функция позволяет задавать необходимое количество значений, используемых вызываемой функцией:

```
mixed call_user_func_array(string function_name [, array param])
```

Синтаксис наглядно показывает принцип работы функции. Строка параметра function_name отвечает за название функции, вызов которой и будет производиться. Параметр param задается в качестве значений массива и производит передачу указанных здесь значений в вызываемую функцию. Это очень удобно. Также что касается возвращаемого значения функции, то оно фиксированное, т. е. постоянное. Например:

```
<?php
function check($var, $val){
echo "***Вводимое значение $var сравнивается с имеющимся: <br>";
if ($var == "Sasha" & $val == "Sveta")
{
echo"Произошло совпадение с установленными параметрами: <br>";
echo "$var <br>";
echo "$val ";
}
else
echo "Сравнение прошло успешно, подобных значений $var и $val
нет";
echo "***\n <br>";
}
$c = "Sveta";
$host = 5;
$me = "Igor";
call_user_func_array('check', array("car",$host));
call_user_func_array('check', array("Sasha",$c));
call_user_func_array('check', array("dog",$me));
?>
```

Пример подобран таким образом, чтобы как можно проще попытаться объяснить вам принципы работы данной функции. В начале выполнения данного скрипта происходит задание значений параметров переменным $c, $host, $me. В реальном

режиме работы данные параметры могут задаваться непосредственно через форму. Для облегчения поставленной задачи мы будем использовать такой метод. После этого производим три раза вызов функции с массивом значений, вносимых в имеющуюся функцию по имени check. После передачи необходимых значений этой функции работу по выполнению скрипта начинает производить функция check(). Проделав в строгой последовательности все операции, получаем следующий результат:

```
***Вводимое значение car сравнивается с имеющимся:
Сравнение прошло успешно, подобных значений car и 5 нет***
***Вводимое значение Sasha сравнивается с имеющимся:
Произошло совпадение с установленными параметрами:
Sasha
Sveta ***
***Вводимое значение dog сравнивается с имеющимся:
Сравнение прошло успешно, подобных значений dog и Igor нет***
```

Думаем, в последовательности выводимого результата вам не составит труда разобраться.

Функция call_user_func_array() получила распространение в PHP 4—4.0.4.

ВНИМАНИЕ

Эта функция была добавлена к коду CVS после выпуска PHP 4.0.4pl1.

Аналогичную операцию проделывает другая, похожая на эту, функция call_user_func(). По принципу работы данные функции практически ничем не отличимы. Единственная особенность заключается в том, что последняя производит задание только одного значения параметра вызываемой функции. Функция call_user_func() имеет следующий синтаксис:

```
mixed call_user_func(string function_name [, mixed parameter
t, mixed...]])
```

Покажем работу данной функции на примере:

```
<?php
function check($var) {
print "Вам необходимо приобрести $var пиджак <br>";
}
call_user_func('check',"красный");
call_user_func('check', "синий");
?>
```

Из примера видно, как происходит задание передаваемого значения в вызываемую функцию check(). Последовательно заданные значения красный, синий в функцию check() дали следующий результат выполнения скрипта:

Вам необходимо приобрести красный пиджак

Вам необходимо приобрести синий пиджак

Функция call_user_func() также отличается от функции call_user_func_array() тем, что она получила распространение немного раньше, чем последняя.

Описанные в данном параграфе функции не работают с классами и объектами, но они помогут в изучении классов и будут неплохой опорой при написании скриптов.

24.2. Вызов методов пользователя классов

Функции работы с классами аналогичны элементарным функциям, и то, что они относятся к классам, не делает их очень уж сложными. Изучение начнем со следующих функций:

- call_user_method(),
- call_user_method_array().

Функция call_user_method() позволяет вызывать метод пользователя определенного объекта. Помимо этого за счет введенных параметров можно, не прибегая ни к каким манипуляциям, просто производить обращение ктой или иной функции (методу) имеющегося класса. Функция имеет следующий синтаксис:

```
mixed call_user_method(string method_name, object obj [, mixed
parameter [, mixed...]])
```

Строка method_name определяет имя метода обработки данных, т. е. имя функции класса, obj отвечает за определенный объект класса, parameter — параметр. Параметр может быть как цифрой, так и символом. Чем же удобнее использовать именно такой способ? Дело в том, что существуют ситуации, когда намного эффективнее будет использование функции, нежели последовательного обращения. Например, имея дело с «навороченными» классами, можно оценить все достоинства и недостатки.

Функция используется в PHP 3–3.0.3 и PHP 4 и выше.

Функция call_user_method() аналогична функции call_user_func(). Отличие их состоит в том, что первая производит обращение непосредственно к методу класса, а вторая — к самой функции. Поэтому мы сначала рассмотрели эти функции, хотя никакого прямого отношения к классам они не имеют.

Рассмотрим принцип работы функции call_user_method() на примере:

```
<?php
class Bus_stop {
```

```
var $number;
var $finish;

function Bus_stop ($name, $stop)
{
$this->number = $name;
$this->finish = $stop;
}
function check ($parameter="")
{
echo "Номер маршрута автобуса " . $this->number. " \n <br> ";
echo "Порядковый номер автобуса ".$parametr.", конечная остановка которого ".$this->finish."\n<Br>";
}
}
$bus = new Bus_stop ("5","ул.Одинцова") ;
echo "*Непосредственный вызов\n* <br>";
$bus -> check () ;
echo "\n*Косвенный вызов\n*<Br> ";
call_user_method("check", $bus, "1257");
?>
```

В этом скрипте создается класс Bas_stop(), который имеет свои методы. Первая функция класса позволяет указывать значения переменным, заданным при определении объекта. В нашем случае при помощи строки `$bus=new Bus_stop ("5", "ул.Одинцова") ;` производится создание объекта с необходимыми для этого параметрами (5 — это номер маршрута нашего автобуса, ул. Одинцова — это конечная остановка, т. е. пункт, где происходит разворот автобуса). После этого имеющиеся значения передаются в класс. Далее, чтобы получить результаты работы класса Bas_stop, нам необходимо вызвать его. При этом рассмотрим функцию check() отдельно:

```
function check ($parametr="")
{
echo "Номер маршрута автобуса " .$this->number. " \n <br> ";
```

```
    echo "Порядковый номер автобуса ".$parametr.", конечная оста-
    новка которого ".$this->finish."\n<Br>";
    }
```

Функция check () позволяет выводить результат с учетом введенных в класс значений. Именно при помощи этой функции мы сможем увидеть все отличия между непосредственным способом вызова и косвенным с описываемой нами функцией `call_user_method()`. Обратите внимание, при обращении к этой функции необходимо задавать значение, которое будет принимать переменная `$parametr`. Это будет сам номер автобуса, который ходит по маршруту номера 5. Сами ведь автобусы могут меняться, например один сломался, второй не завелся и т. д., поэтому и сам параметр тоже будет меняться. Способ вызова такой функции класса `Bus_stop` может быть произведен двумя способами:

- непосредственный вызов;
- косвенный вызов.

Рассмотрим эти способы в порядке, в котором они представлены, с учетом нашего примера.

Непосредственный способ вызова заключается в элементарном обращении объекта к вызываемому методу (см. гл. 15). В нашем примере это выглядит следующим образом:

```
$bus -> check();
```

Для задания параметра необходимо отобразить его значение при вызове функции. Мы тут это не делали, так как это элементарные шаги.

При косвенном вызове все происходит за счет использования функции `call_user_method()`:

```
call_user_method("check", $bus, "1257");
```

Как видно из параметров функции `check`, имя функции, т. е. вызываемого метода, $bus — определенный ранее объект со всеми его параметрами, "1257" — именно эти параметры, которые непосредственно передаются функции и переменная `$parametr` принимает значение 1257, после чего и происходит манипуляция данным параметром и не только. Это можно увидеть, ознакомившись с результатом выполнения программы:

```
*Вызов непосредственно*
Номер маршрута автобуса 5
Порядковьм номер автобуса ..., конечная остановка которого ул. Одинцова
*Вызов косвенно*
Номер маршрута автобуса 5
Порядковьм номер автобуса 1257, конечная остановка которого
ул. Одинцова
```

Обратите внимание, что в случае непосредственного вызова на месте, где стоит многоточие, должен быть указан номер 1257, но так как мы при обращении кданному методу check не сделали этого, соответственно он не был принят. Во втором же случае все наоборот.

Если вам необходимо помимо одного параметра задать еще несколько, то для этого используется функция косвенного вызова `call_user_method_array()`. Например, по маршруту с номером 5 ходит не один автобус, как было в первом случае, а несколько, предположим, с номерами 1257, 1267, 1258, 1278. В этом случае применить функцию call_user_method() невозможно, так как в функцию check будем задавать уже не одно значение, а массив значений.

Функция `call_user_method_array()` **используется в:**

- PHP 4,
- CVS.

Функция была добавлена к CVS после выпуска PHP4.0.4pl1.

`Call_user_method_array()` **имеет следующий синтаксис:**

```
mixed call_user_method_array(string method_name, object obj
[, array paramarr])
```

Все параметры аналогичны параметрам `call_user_method()`, единственное отличие состоит в том, что paramarr принимает не одно значение, а массив значений. В этот массив запишем значения 1257, 1267, 1258, 1278..

```
<?php
class Bus_stop {
var $number;
var $finish;

function Bus_stop($name, $stop)
{
$this->number = $name;
$this->finish = $stop;
}
function check($parametr="", $parametr1="", $parametr2="",
$parametr3="")
{
echo "Номер маршрута автобуса ".$this->number." \n <br> ";
echo "Порядковые номера автобусов на этом маршруте ".$parametr.",
```

```
".$parametr1.", ".$parametr2.", ".$parametr3.", конечная останов-
ка ".$this->finish."\n<br>";
}
}
$bus = new Bus_stop("5","ул.Одинцова");
echo "*Непосредственный вызов\n* <br>";
$bus -> check();
echo "\n*Косвенный вызов\n*<br> ";
call_user_method_array("check", $bus, array(1257, 1267, 1258,
1278));
?>
```

Результат программы

```
*Непосредственный вызов*
Номер маршрута автобуса 5
Порядковые номера автобусов на этом маршруте,,,, конечная оста-
новка ул. Одинцова
*Косвенный вызов*
Номер маршрута автобуса 5
Порядковые номера автобусов на этом маршруте 1257, 1267, 1258,
1278, конечная остановка ул. Одинцова
```

При непосредственном вызове в результате вывелось несколько запятых подряд, они появились за счет того, что не были переданы значения переменной `$parametr` в функции check.

24.3. Проверка классов

Проверка классов, а также методов является необходимым при создании или удалении классов. Функции, отвечающие за проверку классов и методов, прежде всего являются функциями, возвращающими булевый тип. Это хорошо тем, что при написании скрипта для проверки условия можно использовать эту функцию непосредственно в самой операции условия и от того, какой результат будет возвращен, и будет происходить выполнение того или иного блока. К функциям проверки классов и методов классов на определенность относятся:

- `method_exists()`,
- `class_exists()`.

Функция method_exists() проверяет существования метода указанного класса при определении объекта. Синтаксис данной функции:

```
bool method_exists(object object, string method_name)
```

Как и было сказано ранее, функция возвращает true или false. На месте параметра object, как правило, используют определенный объект, method_name — имя метода, т. е. имя существующей функции класса, объявленной при определении объекта. Когда все указанные параметры совпадают с существующими, просто происходит возврат значения true, в противном случае — false.

Для проверки существования самого класса, т. е. его имени, применят функцию class_exists(), похожую на method_exists, так как они обе возвращают одинаковые значения и принцип их работы практически аналогичен. Функция class_exists() имеет следующий синтаксис:

```
bool class_exists(string class_name)
```

Параметр class_name содержит имя класса, наличие которого и нужно проверить.

ВНИМАНИЕ

Функции получили распространение в PHP 4–4.0b4 версии. Поэтому при использовании версии PHP 3 вы будете получать ошибку.

Пример использования описанных функций:

```
<?php
class shop {
var $fruits;
var $fruits1;

function shop($apple, $banana)
{
$this->fruits = $apple;
$this->fruits1 = $banana;
}
functionprice($var, $var1)
{
echo "Цена на ".$this->fruits." равна $var \n <br> ";
```

```
echo "Цена на ".$this->fruits1." равна $varl \n <br> ";

}
}
$bus = new shop("Яблоки","Бананы");
$bus -> price(10,15);
if(class_exists(Bus_stop))
echo "Функция class_exists(Bus_stop) = true <br>";
else
echo "Функция class_exists(Bus_stop) = false <br>";

if(class_exists(shop))
echo "Функция class_exists(shop) = true <br>";
else
echo "Функция class_exists(shop) = false <br>";

if(method_exists($bus,check1))

echo "Функция method_exists ($bus, check1) = true <br>";
else
echo "Функция method_exists ($bus, check1) = false <br>";

if(method_exists($bus,price))

echo "Функция method_exists ($bus, price) = true <br>";
else
echo "Функция method_exists($bus, price) = false <br>";
?>
```

В примере показано, как происходит обращение функции к имеющимся классам и методам, а после этого и к неимеющимся. Обратите внимание на результат работы этого примера:

```
Цена на Яблоки равна 10
Цена на Бананы равна 15
Функция class_exists(Bus_stop) = false
Функция class_exists(shop) = true
Функция method_exists(Object, checkl) = false
Функция method_exists(Object, price) = true
```

Первые две строчки являются программными строками и на данном этапе они не представляют никакого интереса для нас. Все идущие далее строки и проявляют особенности работы функций. Когда мы проверили на существование класса по имени Bus_stop, его, конечно же, не оказалось, следовательно функция вернула значение false. Класс под именем shop существует, поэтому функция вернула true. Аналогична и проверка на существование метода.

24.4. Возврат параметров класса

Функции, отвечающие за возврат параметров, прежде всего зависят от самой конструкции класса. Например, если происходит какое-либо несопоставление или непонимание при выполнении программ, то это как правило называют ошибкой и данные функции ничего вообще не возвращают. Мы рассмотрим три наиболее распространенные функции:

- `get_class ()`,
- `get_parent_class ()`,
- `is_subclass_of ()`.

Функции get_class () используется непосредственно для получения строки имени класса объекта, указанного в параметрах этой функции. Другими словами, функция возвращает имя класса того объекта, который указан в качестве параметров. Отсюда можно сделать вывод, что возвращаемый тип будет string. Таким образом, ее можно сразу использовать в функциях вывода. Синтаксис функции:

```
string get_class(object obj)
```

В поле параметров и будет содержаться наш объект.

ВНИМАНИЕ

Функция get_class о возвращает имя класса в форме нижнего регистра.

Если в конце рассмотренного ранее примера добавить строку:

```
echo get_class($bus);,
```

то на экран браузера будет выведена следующая надпись: shop. Это и есть то возвращаемое имя класса, о котором мы говорили, описывая работу функции `get_class()`.

Когда необходимо получить имя исходного класса объекта, используют функцию get_parent_class(). Ее синтаксис:

```
string get_parent_class(object obj)
```

Синтаксис функции get_parant_class () аналогичен синтаксису функции get_class (). В принципе эти функции аналогичны в применении.

Если у вас возникла необходимость определения принадлежности объекта подклассу указанного класса, воспользуйтесь функцией is_subclass_of(). Функция имеет следующий синтаксис:

```
bool is_subclass_of(object obj, string superclass)
```

Функция возвращает значение булевого типа, т. е. если объект obj принадлежит классу, который является подклассом superclass, то функция возвратит true, в противном случае — false.

Функции являются новыми в PHP, поэтому используются только в PHP 4—4.0b4 и выше.

24.5. Возврат массива параметров объекта

Иногда при работе с классами возникает необходимость получить значения объектов, переменных или методов. Такая возможность упрощает программу и делает ее намного короче и понятнее. Для выполнения таких операций были созданы три функции:

- get_class_methods(),
- get_class_var(),
- get_object_vars().

Рассмотрим функции в том порядке, в котором они располагаются. Функция get_class_methods() создана для того, чтобы возвращать массив значений. Значения эти будут соответствовать именам методов в имеющемся классе. Имя класса, имена методов которого будут возвращены, и будет указываться в функции get_class_methods(). Функция имеет следующий синтаксис:

```
array get_class_methods(string class_name)
```

СОВЕТ

Что касается версии PHP 4.0.6, то там вы можете определить объект непосредственно вместо class_name. Например:

```
$class_methods = get_class_methods($class);
```

Приведем пример использования функции get_class_methods():

```
<?php
class new_class {

function new_class() {
return (true);
}

function func1() {
return(true);
}
function f unc2 () {
return(true);
}

function func3() {
return(true);
}
}
$class = new new_class();
$class_methods = get_class_methods (get_class($class)) ;
foreach ($class_methods as $method_name) {
echo "$method_name\n <br>";
}
?>
```

Данный пример подробно показывает принцип применения функции get_class_methods () на практике. В примере рассматриваются четыре метода. После объявления этих методов происходит создание объекта класса. Строка

```
$class_methods = get_class_methods(get_class($class));
```

возвращает массив значений, т. е. имен функции. В свою очередь параметр, заключенный в функцию get_class_methods (), представляет собой строку get_class ($class). Эта функция позволяет вернуть строку с именем класса, принадлежащего объекту. Именно для таких операций и создавались эти функции.

После получения массива значений методов его необходимо вывести. Для этого можно применить блок программы с использованием операции последовательного вывода значений нашего массива. Результат программы будет следующий:

```
new_class
func1
func2
func3
```

Именно столбец имен функций.

Функция get_class_vars () позволяет получить ассоциативный массив определенных переменных указанного класса. Если переменная не является константой, эта функция ее не возвратит, т. е. функция исключит константу. Функция get_class_vars () имеет следующий синтаксис:

```
array get_class_vars(string class_name)
```

Результирующий массив элементов, который возвратит эта функция, будет следующей формы:

```
varname => value,
```

где varname — имя определенной переменной, а value — ее значения, т. е. функция, как было сказано ранее, возвратит ассоциативный массив.

ВНИМАНИЕ

Неинициализированные переменные класса не будут сообщены функции get_class_vars (). Когда это не учитывают, возникает много ошибок.

Рассмотрим пример:

```
<?php
class new_class {
var $var1; // переменная не определена
var $var2 = "foo";
var $var3 = 100;
var $var4; // переменная не определена
var $var5 = 400;

function new_class() {
return(true);
}
```

```
}
$class = new new_class();
$class_vars = get_class_vars(get_class($class));
foreach ($class_vars as $name => $value) {
echo "$name : $value\n <br>";
}
?>
```

В начале класса мы объявили несколько переменных, при этом некоторым их них задали значения, другие же остались неопределены. И не важно, на каком месте они стояли. Результат все равно будет соответствующий:

```
var2 : foo
var3 : 100
var5 : 400
```

Обратите внимание на то, что переменные, которые были неопределены, в имеющийся ассоциативный массив вообще не вошли.

Когда нужно проделать аналогичную операцию, но только с объектами, используют функцию get_object_vars(). Она позволяет получать значения переменных созданного объекта, конечно, при условии, что они определены, и имеет следующий синтаксис:

```
array get_object_vars(object obj)
```

Функция возвращает ассоциативный массив определенных объектных свойств для указанного объекта obj. Если переменные, объявленные в классе (объект obj которого является образцом), не были подчинены каким-либо значениям (не имели значения), то эти переменные не будут возвращены в ассоциативный массив.

Рассмотрим пример:

```
<?php
class GetName {
var $name1;
var $name2;
var $name3;
function GetName ($name1, $name2) {
$this->name1 = $name1;
$this->name2 = $name2;
}
```

```
function setName($name3) {
$this->name3 = $name3;
}
function Name() {
return array("name1" => $this->name1,
"name2" => $this->name2,
"name3" => $this->name3);
}
}
$class1 = new GetName ("Sasha", "Sveta");
print_r(get_object_vars($class1));
echo"<br>";
$class1->setName("Igor");
print_r(get_object_vars($class1));
?>
```

При выполнении данного примера будут выведены две строки.

Первый ассоциативный массив будет иметь только два значения. Это связано прежде всего с тем, что переменной $name3 не было задано какое-либо значение. Операцией:

```
$class1->setName("Igor");
print_r(get_object_vars($class1));
```

присваивается значение переменной $name3. И ассоциативный массив принял сразу не два значения, а три. В результате выполнения этого скрипта будет выведено:

```
Array ([name1] => Sasha [name2] => Sveta)
Array ([name1] => Sasha [name2] => Sveta [пате3] => Igor)
```

Все рассмотренные функции работают в версиях PHP 4—4.0RC1 и выше.

24.6. Определение существующих классов

Для решения задачи определения классов была создана функция get_declared_classes(). Она возвращает массив с названиями определенных классов и имеет следующий синтаксис:

```
array get_declared_classes(void)
```

Для более подробного разъяснения уточним, что эта функция возвращает матрицу названий классов, объявленных в текущем сценарии.

ВНИМАНИЕ

В версии PHP 4.0.1pl2 три дополнительных класса возвращены в начале матрицы: stdClass(определен в Zend/zend.c), OverloadedTestClass (определен в ext/standard/basic_functions.c) и Directory (определен в ext/standard/dir.c).

Дополнительные классы могли бы присутствовать в зависимости от того, какую из библиотек вы устанавливаете в PHP.

Функция работает в PHP 4—4.0RC2 и выше.

24.7. Программирование при помощи функций работы с классами

Рассмотрев все возможные функции, продемонстрируем их применение на практике. Напишем пример с применением имеющихся у нас знаний и покажем основные принципы работы часто встречаемых описанных нами функций классов и объектов.

Для начала создадим два файла, один из которых будет содержать основные классы и подклассы со своими методами и свойствами:

Файл classes-1.inc

```
<?php
// базовый класс со свойствами и методами
class Vegetable {
var $edible;
var $color;
function Vegetable ( $edible, $color="зеленый" ) {
$this->edible = $edible;
$this->color = $color;
}
function is_edible ( ) {
return $this->edible;
}
function what_color ( ) {
return $this->color;
}
```

```
} // конец класса Vegetable
// расширение базового класса
class Spinach extends Vegetable {
var $cooked = false;
function Spinach() {
$this->Vegetable( true, "зеленый" );
}
function cook_it() {
$this->cooked = true;
}
function is_cooked() {
return $this->cooked;
}

} // конец класса Spinach
?>
```

В имеющемся файле мы создали два класса, один из которых — главный, второй является распространением первого. Каждый из имеющихся классов имеет свои методы и свойства. Теперь создадим еще один класс, который будет совершать доступ к этому файлу, а также при помощи изученных функций попытаемся получить все необходимые имена классов, методов и т. д.:

Файл test.php

```
<pre>
<?php
include "classes-1.inc";
// сервисные функции
function print_vars ($obj) {
$arr = get_object_vars ($obj) ;
while (list($prop, $val) = each($arr))
echo "\t$prop = $val\n";
}
```

```
function print_methods ($obj) {
$arr = get_class_methods(get_class($obj)) ;
foreach($arr as $method)
echo "\tfunction $method()\n";
}
function class_parentage ($obj, $class) {
global $$obj;
if (is_subclass_of($$obj, $class) ) {
echo "Объект $obj принадлежит классу " .get_class($$obj) ;
echo " и является подклассом классу $class\n";
} else {
echo "Объект $obj не принадлежит подклассу $class\n";
}
}
// создание двух объектов
$veggie = new Vegetable (true,"синий") ;
$leafy = new Spinach ();
// вывод информации об объекте
echo "Объект veggie содержит следующие классы: CLASS
".get_class($veggie)."\n";
echo "Объект leafy распространяется на следующие классы: CLASS
".get_class($leafy);
echo ",<br> главный класс " .get_parent_class($leafy) . "\n";
// показ свойств veggie
. echo "\nveggie: Свойства\n";
print_vars($veggie) ;
// и методов leafy
echo "\nleafy: Методы\n";
print_methods($leafy);
echo "\nПроисхождение:\n";
•class_parentage ("leafy", "Spinach");
class_parentage ("leafy", "Vegetable") ;
```

```
?>
</pre>
```

В результате выполнения программы на экран будет выведено:

```
Объект veggie содержит следующие классы: CLASS vegetable
Объект leafy распространяется на следующие классы: CLASS
spinach, главный класс vegetable
veggie: Свойства
edible = 1
color = синий
leafy: Методы
function vegetable()
function is_edible()
function what_color()
function spinach()
function cook_it()
function is_cooked()
Происхождение:
Объект leafy не принадлежит подклассу Spinach
Объект leafy принадлежит классу spinach и является подклассом
классу Vegetable
```

На этом изучение классов, объектов и функций работы с ними считаем завершенным. В случае какого-либо вопроса при работе этого примера просто попытайтесь изменить его и посмотреть, что случится с результатом работы. Или же обратитесь к соответствующему описанию той или иной функции в нашей книге.

Заключение

В этой главе была предоставлена полная структура функций, а также методы работы с ними. Обратите внимание прежде всего на сами функции и возвращаемые ими значения. Также не забывайте, что все настоящие знания приходят только тогда, когда вы самостоятельно проделываете те или иные задачи. Для более полного запоминания выделим изученные функции:

- вызов методов пользователя выдаваемых массивом параметров:

```
call_user_func O ,
call_user_func_array;
```

- вызов методов пользователя классов:
  `call_user_method_array()`,
  `call_user_method()`;
- проверка классов:
  `method_exists()`,
  `class_exists()`;
- возврат параметров класса:
  `get_class()`,
  `get_parent_class()`,
  `is_subclass_of()`;
- возврат массива параметров объекта:
  `get_class_methods()`,
  `get_class_var()`,
  `get_object_vars()`;
- определение существующих классов:
  `get_declared_classes()`.

Глава 25
Функции для манипуляций со строками

Еще в школе нас учили: для того чтобы уметь читать, необходимо знать буквы и как они произносятся. Таки в языке программирования, чтобы понять основные принципы языка, как минимум необходимо овладеть грамотой программирования строк. Строки являются неотъемлемым атрибутом практически любой программы и рано или поздно вам все равно придется с ними встретиться. Можно сказать, что без строк вообще немыслимо взаимодействие пользователя с вашим скриптом, ведь данные, передаваемые в ваш скрипт, как правило, представляют собой не что иное, как строки. При игнорировании строк ваши программы были бы ограничены одними сухими расчетами, а это в наше время не приветствуется. Это прежде всего скажется на сервисе вашего ресурса и вызовет скорее всего отрицательные эмоции, нежели положительные. Если бы программисты не научились так профессионально орудовать со строками, как они это делают в наши времена, компьютеры бы никогда не достигли своей нынешней популярности. Ведь строки являются фундаментом программирования.

Строка — это комбинация символов, составляющих слова и предложения. В начале нашей книги мы рассказали об основных концепциях строк и способах выражения (см. гл. 8). В этой главе мы рассмотрим основные функции, позволяющие выполнять элементарные операции со строками, а также приведем примеры. PHP имеет широкий диапазон таких функций. Если возникают какие-то трудности при решении конкретных задач, связанных с употреблением функций манипуляций со строками, то следует создавать функции самим на основании полученных знаний.

В этой главе рассмотрены следующие темы:

- функции удаления пробелов;
- работа с ASCII-кодами;
- шифрование строк;
- функции вывода строк на печать;
- деление и соединение строк;
- работа с кодом HTML;
- доступ с операциями замены строк или подстрок, сравнение строк;
- операции поиска символов;
- перевод строк в верхний и нижний регистр;
- перевод строки в другую кодовую таблицу.

25.1. Функции удаления пробелов

Каждый из нас хотя бы раз совершал ошибки при вводе какой-либо информации в форму. Допустим, задумались, заболтались, отвлеклись — и вместо того чтобы ввести один пробел, вы ввели два или более. На первый взгляд казалось бы, что в этом может быть плохого? Например, ваша программа совершает работу со строками и производит конкретный подсчет символов, скажем, до пробела. После этого выделяет слово и производит операцию над ним, считая, что все посчитанные позиции являются символами. Далее, пропустив пробел, переходит к подсчету следующих символов, при этом в случае внесения не одного пробела, а двух, программа посчитает эти две позиции как символы, в итоге могут возникнуть непредсказуемые ошибки.

В результате при получении строк, введенных пользователем в форму, они передаются в наш скрипт такими, какими они были введены — со всеми пробелами и неточностями. Процесс передачи данных из HTML-формы в PHP-скрипт был описан ранее (см. п. 9.6), поэтому будем приводить примеры скриптов, без HTML-тегов. Это поможет вам сконцентрировать внимание на более важных темах.

Функции, которые удаляют пробелы:

- `chop()`,
- `trim()`,
- `ltrim()`,
- `rtrim()`.

Чтобы произвести удаление лишних пробелов в обрабатываемой строке, необходимо воспользоваться функцией chop (). Прежде всего разберемся, что значит лишние пробелы и в каком случае они действительно являются лишними.

Строка представляет собой совокупность символов, разделенных пробелами. Правильным считается, когда между каждым словом установлено не более одного пробела (за исключением условий, когда это действительно необходимо, в этом случае о применении функции chop () речи не может идти вообще). Эта функция удаляет все повторяющиеся пробелы между словами. Ее синтаксис:

```
string chop(string str)
```

Функция chop () возвращает «правильную строку», т. е. строку, в которой будут удалены все лишние пробелы. Сама строка задается непосредственно на месте параметра str как в самой функции, так и при помощи переменной. Последний способ широко распространен среди программистов и является рекомендуемым, поэтому при изучении старайтесь уделять больше времени работе с самими переменными, а не с какими-то конкретными значениями.

ВНИМАНИЕ

Функция chop () в PHP отлична от функции chop () в Perl, так как функция chop () в Perl удаляет последнюю букву в указанной строке.

Рассмотрим пример работы функции chop ():

```
<?php
$name = "Удалено два пробела, далее — три !!!";
$name1 = "Самая обычная строка!";
$line = chop($name);
$line1 = chop($name1);
$line2 = chop("ввв ввв ввв ввв ввв ввв !");
echo $line. "<br>";
echo $line1. "<br>";
echo $line2. "<br>";
?>
```

В данном примере приведено три случая. Первый — это строка с большим числом пробелов. Эту строку содержит переменная $name. Далее — обычная строка, показывающая, что функция не произведет никакой ошибки, если ей будет указана правильная строка. В этом случае она просто произведет поиск лишних пробелов и вернет ее в строку $line1. В качестве третьего применения функции строка указывается в самой функции без использования при этом переменных. Как уже говорилось, этот случай является не совсем удобным, хотя знать его надо. После выпол-

нения функции chop () получаются «правильные» строки, т. е. строки с удаленными пробелами. Функция echo () выводит эти функции на экран браузера. Результат работы данного скрипта будет следующий:

```
Удалено два пробела, далее — три !!!
Самая обычная строка!
ввв ввв ввв ввв ввв ввв !
```

Обратите внимание, что функция также считается с запятой, как с существующим символом. Это касается также всех остальных символов (чисел, кавычек ит. д.).

Иногда необходимо произвести удаление лишних пробелов с конца или с начала строки или же одновременно.

Чтобы удалить пробел в начале строки, используется функция `ltrim()`. Для более простого запоминания можно написать `lefttrim` — это значит, что функция удаляет пробелы с левой стороны строки, т. е. с начала. По принципу работы функция аналогична chop (), это также можно отнести и к синтаксису функции itrim ():

```
string Itrim (string str)
```

Функция itrim () возвращает строку, указанную вместо параметра str, с удаленными лишними пробелами в начале строки. Функция также удаляет символы: \n, \r, \t, \v, \0.

Аналогично функции Itrim () осуществляют свою работу функции `rtrim()` и `trim()`. Функция rtrim () (`righttrim` — с правой стороны) удаляет все пробелы с конца строки. Ее синтаксис:

```
string rtrim(string str)
```

Как видно, он ничем не отличается от синтаксиса функции itrim (), аналогично можно сказать и про функцию trim ():

```
string trim(string str)
```

Функция `trim()` удаляет все пробелы как с начала строки, так и в конце одновременно. В этом состоит ее основное отличие от приведенных функций Itrim () и `rtirm()`. Приведем пример:

```
<?php
$name = "  ppp ppp ppp ppp ppp  ";
$name1 = "  ppp ppp ppp ppp ppp  ";
$name2 = "  ppp ppp ppp  ";
$line = ltrim($name);
$line1 = rtrim($name1);
```

```
$line2 = trim($name2) ;
echo "***".$line."***<br>";
echo"***".$line1."***<br>";
echo "***".$line2."***<br>";
?>
```

В данном примере для каждой функции создана своя строка с большим количеством символов. После применения рассмотренных функций все очень сильно изменилось:

```
1.***ppp ppp ppp ppp ppp ***
2.*** ppp ppp ppp ppp ppp***
3.***ppppppppp***
```

Номера строк проставлены специально, чтобы более подробно описать каждую из выведенных строк. Звездочки применены для того, чтобы определить, в каком месте есть пробел, в каком нет. Так, например, можно сказать, что в первой строке в начале удалены все пробелы. Во второй строке наоборот, — все пробелы удалены в конце строки. В третьей строке пробелы как сначала, так и с конца строки отсутствуют.

ВНИМАНИЕ

По результату этого скрипта также можно заметить, что функции trim(), ltrim() и rtrim(), также как и chop(), удаляют лишние пробелы между символами. Помимо того, что они удаляют все пробелы с двух сторон (с начала и с конца строки, соответственно), эти функции еще возвращают «правильные строки», т. е. строки с удаленными лишними пробелами между символами.

Описанные функции удаления пробелов используются в PHP 3 и PHP 4.

25.2. Работа с ASCII-кодами

Прежде чем приступить к изучению этого вопроса, рассмотрим, что такое ASCII-коды, для чего они применяются и почему получили широкое распространение.

При создании компьютеров инженеры долго искали рациональное решение задачи способа хранения символов, которые вы видите на вашем мониторе. Не вникая в подробности, можно сказать просто, что символ расположен на мониторе и при нажатии клавиши происходит его появление. Но задумывались ли вы когда-нибудь над тем, как же осуществляется работа с этим самым символом в вашем компьютере, каким способом происходит удержание этого символа в памяти вашего компьютера? На самом деле это очень сложный процесс, поэтому мы расскажем поверхностно, углубляясь только в ту часть, которая действительна поможет вам при создании скриптов.

Символы, которые вы видите на экране вашего монитора, сохраняются в памяти компьютера при помощи кодов. Разработчиками была создана таблица кодов ANSI (American National Standards Institute), используемых при хранении символов в памяти компьютера или файлах. Таблица кодов ANSI содержит расширенный набор кодов ASCII (American Standard Codes for Information Interchange). Начальные 128 кодов ASCII были разработаны для телетайпных коммуникаций. Первые 32 кода — управляющие, хотя только четыре из них используются в программах под Windows. Коды от 32 до 127 принадлежат обычным алфавитно-цифровым символам латинского языка, специальным символам и знакам операций. Коды от 128 до 255 принадлежат дополнительному набору символов. Обратите внимание на то, что дополнительные символы, используемые программами под MS-DOS, отличаются от дополнительных символов, используемых программами под Windows.

Теперь перейдем к изучению функций, осуществляющих работу с этими кодами. Описываемые функции будут либо возвращать символ согласно значению таблицы кодов ASCII, либо, наоборот, преобразовывать его из символа в код, соответствующий коду этого символа в таблице кодов ASCII. Процесс преобразования является двухсторонний, поэтому и будет рассмотрено две функции:

- `chr()`,
- `ord()`.

Функция `chr()` позволяет возвращать строку символа, соответствующего коду ASCII, указанного в качестве параметров данной функции. Синтаксис функции `chr()`:

```
string chr(int ascii)
```

Функция `chr()` возвращает односимвольную строку, соответствующую указанному коду ASCII на месте параметра ASCII. Рассмотрим пример:

```
<?php
$string = chr(56);
$string1 = chr(57);
$string2 = chr(58);
$string3 = chr(59);
$string4 = chr(60);
$string5 = chr(61);
echo $string."<br>";
echo $string1."<br>";
echo $string2."<br>";
echo $string3."<br>";
```

```
echo $string4."<br>";
echo $string5."<br>";
?>
```

Здесь приведен список преобразования ASCII-кодов в обычные символы. В примере специально взят перечень элементов, чтобы вы могли проследить систематическую последовательность изменяющихся ASCII-кодов. Результатом работы данного примера будет совокупность символов:

```
8
9
:
;
<
=
```

Коды ASCII с 56 по 61 соответствуют выведенным символам. Когда применяются данные операции и насколько они эффективны, можно понять из того, что эти операции очень распространены при работе с памятью.

Рассмотрим другую аналогичную функцию работы с ASCII-символами — `ord()`. Она выполняет противоположные функции `chr()` операции, т. е. преобразовывает символ в целочисленное значение, соответствующее коду ASCII-таблицы. Синтаксис функции `ord()`:

```
int ord(string string)
```

Для примера создадим скрипт, который будет осуществлять следующую операцию. Имеется HTML-форма, в которую при необходимости пользователь может вводить либо код ASCII, либо сам символ. Далее происходит выполнение скрипта и в результате пользователь видит на экране либо символ, соответствующий коду ASCII, либо, наоборот, сам код ASCII, соответствующий символу. Для начала создадим HTML, содержащий две формы:

Файл `index.htm`:

```
<html>
<head>
<title>Программа</title>
</head>
<body>
```

Преобразование кодов ASCII в символы

```
<form action="test.php" method = "get">
```

```
<input type="Text" name="name"size ="3" >
<input type="Submit" name="ame" value="Получить символ">
</form>
```

Преобразование символа в ASCII-коды

```
<form action="test.php" method = "get">
<input type="Text" name="name1" size ="1">
<input type="Submit" name="ame1" value="Получить ASCII-код">
</form>
</body>
</html>
```

При помощи этих форм можно осуществлять ввод необходимых значений кода ASCII либо символов. После того как ввод осуществлен, вы можете нажать на кнопку и этим самым передадите данные этой формы в скрипт test.php. Скрипт, получив данные, производит обработку и выведение результата. При этом скрипт ссылается на значения переменных, которые были заданы при вызове этого скрипта нашей формой. Чтобы понять, как производит обработку данных скрипт test.php, приведем его содержимое:

Файл test.php:

```
<?php
include("index.htm");
$string = chr($name);
echo $string."<br>";
$string1 = ord($name1);
if ($string1 != "0")
echo $string1."<br>";
?>
```

На этом примере можно полностью ознакомиться с принципами работы функции ord(). Функции chr() и ord() являются важными при решении разнообразных задач. Например, можно без труда производить операции с кодами ASCII-символов, при этом не прибегая к манипуляции самими символами. Со временем вы научитесь применять эти функции не только в простых программах, и это значительно облегчит работу с PHP.

Функции chr() и ord() используются в PHP 3 и PHP 4.

25.3. Шифрование строк

Как правило, чтобы передать важную информацию, прежде всего необходимо каким-либо способом ограничить ее от доступа посторонних лиц. Конечно, это весьма существенно при передаче очень важной информации, в случае же обычного сообщения не стоит налагать какие-либо ограничения, так как, попадая в чужие руки, эта информация не может нанести никакого вреда ни вам, ни вашим партнерам по бизнесу. Мы же рассказываем об информации, которая играет весьма важную роль в жизни лица, а может и организации, которая осуществляет передачу этой самой информации.

Существует большое множество путей, ограничивающих доступ к передаваемой информации. Это всевозможные кодировщики, блокировщики и разнообразные программы. Сразу возникает вопрос: как эти программы осуществляют процесс кодировки. Они делают это за счет своей разветвленной структуры функций, которые и позволяют создавать программы, производящие кодирование информации. PHP является довольно-таки новым языком программирования, но можно сказать, что в этом отношении он не уступает более известным и распространенным. Разработчики PHP постарались включить большое количество всевозможный функций, решающих самые сложные проблемы программирования.

Рассмотрим следующие функции:

- `crypt()`,
- `md5()`,
- `crc32()`.

Функция crypt () осуществляет стандартный процесс шифрования методом DES. Думаем, не стоит рассказывать более углубленно об этом методе, так как программисты, профессиональные пользователи операционной системы Unix, давно знакомы с ним. Если у вас возникают какие-либо проблемы с этим методом, просто обратитесь к документации по Unix, там более подробно рассказано о методе шифрования DES. Функция `crypt()` имеет следующий синтаксис:

```
string crypt(string str [, string salt])
```

Функция crypt () осуществляет шифрование строки, используя стандартный метод Unix DES. В качестве параметра str в функцию передается строка, которую необходимо зашифровать, а также дополнительная символьная строка (параметр salt). На основании указанной дополнительной строки и будет основываться шифрование. Если параметр salt отсутствует, то он будет установлен случайным образом.

Некоторые операционные системы способны поддерживать больше одного способа шифрования. Иногда метод шифрования DES заменяется основанными на MD5 алгоритмами. Как вы уже заметили, именно параметр salt указывает на тип шифрования.

Во время установки PHP определяются возможности функции шифрования и то, будет ли поддерживать параметр salt другие методы шифрования. Если параметр

salt не установлен, то PHP автоматически генерирует стандартный двухсимвольный ключ DES, еслиже в системе по умолчанию установлен тип шифрования MD5, то будет сгенерирован MD5-совместимый ключ.

PHP устанавливает константу CRYPT_SALT_LENGTH, которая сообщает, способен ли регулярный двухсимвольный параметр salt осуществлять шифрование на вашей операционной системе или же потребуется применение 12-символьного способа шифрования MD5.

При использовании параметра salt необходимо помнить, что указанный тип шифрования будет генерироваться только один раз. Если вы произведете вызов этой функции рекурсивно, то может возникнуть защита функции, в результате чего вы не получите необходимый вам результат.

Стандартное шифрование DES crypt () содержит ключ в двух первых символах потока вывода. Нет других функций дешифрации, кроме crypt (), использующей однопроходный алгоритм.

На системах, где функция crypto поддерживает типы кодирования кратного числа, происходит установка значений приведенных ниже констант, равная 0 или 1 в зависимости от того, является ли данный тип доступным:

Константа	Пояснение
CRYPT_STD_DES	Стандартное кодирование с генерированием двухсимвольного ключа DES
CRYPT_EXT_DES	Расширенное кодирование с генерированием девяти-символьного ключа DES
CRYPT_MD5-MD5	12-символьное кодирование, тип шифрования MD5, начинающийся с $1 $
CRYPT_BLOWFISH	Расширенное кодирование с генерированием 16-символьного ключа DES, начинающегося с 2

Приведем пример:

```
<?php
$stringstr = "Строка для шифрования";
crypt ($stringstr, CRYPT_MD5-MD5) ;
?>
```

Думаем, что какие-либо пояснения к данному примеру делать излишне.

Функция md5 () вычисляет хэш типа шифрования MD5 строки. Данный тип распространен при использовании функции crypt () . Функция md5 () имеет следующий синтаксис:

```
string md5  (string str)
```

Вместо параметра str задается хэш типа шифрования MD5. Другими словами, функция md5 О вычисляет значение MD5 для строки, заданной вместо параметра str, используя алгоритм RSA Data Security, Inc. MD5 Message-Digest (более подробную информацию смотрите на сайте www.php.net).

Функции crypt () и md5 () используются в PHP 3 и PHP 4.

Функция crc32 О возвращает полином строки. Ее синтаксис:

```
int crc32(string str)
```

crc32 () генерирует циклическую контрольную сумму статической неопределенности с длиной задаваемой строки 32 бита. Строка списывается в качестве параметра функции str. Это, как правило, используется для того, чтобы подтвердить целостность передаваемых данных.

Функция crc32 () работает в PHP 4.0.1 и выше.

25.4. Функции вывода строк на печать

Научившись осуществлять всевозможные операции вычисления, преобразования и тому подобное, вы сталкиваетесь с проблемой, как теперь все это показать пользователю. Большинство скриптов работают для того, чтобы произвести какую-либо обработку и затем вывести эти значения на экран пользователя, чтобы он мог увидеть результат работы написанного нами скрипта.

Сеть в основном рассчитана на визуальное восприятие, поэтому было бы, конечно, неестественным, если бы язык PHP не имел функций вывода данных на экран браузера. Хотя сложно вообще вспомнить язык программирования, лишенный таких функции, скорее всего это малоизвестный, нераспространенный и действующий для решения задач специального назначения. Что касается изучаемого языка программирования PHP, то он к таким не относится, поэтому имеет обширный перечень функций, позволяющий производить вывод полученного результата на экран.

Все функции PHP осуществляют построковый вывод данных. Также PHP осуществляет работу по способу форматирования строки таким образом, как этого требует программист. Все это будет описано при изучении следующих функций:

- sprintf(),
- printf(),
- print(),
- echo().

Чтобы произвести форматирование строки, в соответствии с требованиями программиста необходимо воспользоваться функцией sprintf(). Ее синтаксис:

```
sprintf(string format, mixed [args]...);
```

Функция sprintf () возвращает строку, обрабатываемую в соответствии с форматирующей строкой format.

Форматирующая строка содержит директивы: обычные символы (кроме %), которые копируются прямо в результат, и описания изменений, каждое из которых выполняет определенные действия. Это прежде всего применяется к функциям `sprintf()` И `printf()`.

Каждое описание изменений состоит из знака %, после которого и следуют один или несколько элементов в указанном порядке:

1. Дополнительное описание заполнения (padding specifier), которое говорит, какие символы будут использоваться для заполнения результата до правильного размера строки. Это могут быть пробелы или 0 (символ нуля). По умолчанию заполняется пробелами. Альтернативный символ заполнения может быть определен одинарной кавычкой.

2. Дополнительное описание выравнивания (alignment specifier), которое говорит, что результат должен быть выровнен по левому или по правому краю. По умолчанию выравнивание происходит по правому краю; символ а —выравнивание по левому краю.

3. Дополнительное описание ширины (width specifier), которое говорит, с каким минимальным количеством символов (minimum) может производиться данная замена.

4. Дополнительное описание точности (precision specifier), которое говорит, сколько десятичных знаков следует отображать для чисел с плавающей точкой. Это описание не действует на остальные типы, кроме double.

5. Описание типа (type specifier), которое говорит о том, как тип данных аргумента должен трактоваться.

Рассмотрим возможные типы:

Символ	Значение
%	Символ процента. Аргумент не требуется
b	Аргумент трактуется как integer и представляется как двоичное число
c	Аргумент трактуется как integer и представляется как символ с ASCII значением
d	Аргумент трактуется как integer и представляется как десятичное число
f	Аргумент трактуется как double и представляется как число с плавающей точкой
o	Аргумент трактуется как integer и представляется как восьмеричное число
s	Аргумент трактуется и представляется как строка
x	Аргумент трактуется как integer и представляется как шестнадцатеричное число (с буквами в нижнем регистре)
X	Аргумент трактуется как integer и представляется как шестнадцатеричное число (с буквами в верхнем регистре)

```
<?php
// первый блок
$num1 = 15.2;
$num2 = 16.4;
$num = ($num1+$num2)/2;
$is = sprintf ("%01.2f", $num);
echo "Результатом форматированной строки будет число ".$is."<br>";
// второй блок
$num4 = 15;
$num5 = 16;
$num6 = 17;
$is1 = sprintf ("%04b-%03b-%02b", $num4, $num5, $num6) ;
$is2 = sprintf ("%04d-%03d-%02d", $num4, $num5, $num6) ;
$is3 = sprintf ("%04f-%03f-%02f", $num4, $num5, $num6) ;
$is4 = sprintf ("%04o-%03o-%02o", $num4, $num5, $num6) ;
$is5 = sprintf ("%04s-%03s-%02s", $num4, $num5, $num6) ;
$is6 = sprintf ("%04x-%03x-%02x", $num4, $num5, $num6) ;
$is7 = sprintf ("%04X-%03X-%02X", $num4, $num5, $num6) ;
echo "Двоичное число — аргумент b: ".$is1."<br>";
echo "Десятичное число — аргумент d: ".$is2."<br>";
echo "Число с плавающей точкой — аргумент f: ".$is3."<br>";
echo "Восьмеричное число — аргумент o: ".$is4."<br>";
echo "Строка — аргумент s: ".$is5."<br>";
echo "Шестнадцатеричное число с буквами в нижнем регистре —
аргумент x: ".$is6."<br>";
echo "Шестнадцатеричное число с буквами в верхнем регистре —
аргумент X: " .$is7."<br>";
?>
```

Рассмотренный пример можно мысленно разделить на два блока, в первом блоке приводится простая математическая операция и форматирование строки с результатом этой операции. После того как форматирование произведено, происходит вывод результата.

Во втором блоке программы мы специально применили большое количество аргументов, чтобы вы могли проследить за изменением результата при применении того или иного аргумента. В результате выполнения программы на экран будет выведено:

```
Результатом форматированной строки будет число 15.80
Двоичное число — аргумент Ь: 1111-10000-10001
Десятичное число — аргумент d: 0015-016-17
Число с плавающей точкой — аргумент f: 0015.000000-016.000000-
17.000000
Восьмеричное число — аргумент o: 0017-020-21
Строка - аргумент s: 0015-016-17
Шестнадцатеричное число с буквами в нижнем регистре — аргумент x:
000f-010-11
Шестнадцатеричное число с буквами в верхнем регистре — аргумент
X: OOOF-010-11
```

Изучив основную функцию форматирования строк, перейдем к описанию функций, позволяющих все эти данные выводить на экран. Функция printf() позволяет выводить строки с учетом всех описанных ранее параметров форматирования. Функция printf() является аналогией функции sprintf(). Она отличается только возвращаемым значением. Функция sprintf() возвращает строку в переменную, которую потом можно будет вывести при помощи функции echo О или какой-либо другой, позволяющей отображать строки на экране. А функция printf () выполняет форматирование указанной в ней строки и при этом одновременно производит вывод строки. Рассмотрим синтаксис функции:

```
int printf(string format [, mixed args...])
```

Если есть переменная, имеющая какое-либо значение, которое необходимо вывести, используют функцию print(). Она очень проста в использовании и позволяет выводить любые строки и символы в окно браузера. Ее синтаксис:

```
print(string arg);
```

Вместо параметра arg задается строка. Например:

```
<?php
$str = "Привет школа!!!";
print $str;
print "Hello world!";
print. ("Hello world!");
?>
```

Обратите внимание на разные способы задания значения функции print().

Существует еще одна функция, очень популярная и широко распространенная среди программистов — echo(). Она аналогична функции print(). Думаем, что за этот период вы уже ознакомились с этой функцией, так как она очень часто использовалась в примерах. Ее синтаксис:

```
echo(string arg1, string [argn]...)
```

Фактически echo() является не функцией, а конструкцией языка. Если вы хотите передать больше одного параметра через echo, вам не нужно включать параметры в пределах круглых скобок. Приведем пример, характеризующий конструкцию echo:

```
<?php
echo "Привет мир! <br>";
echo "Позволяет выводить сложные
предложения.
Новая строка будет также выведена.
<br>";
echo "Позволяет выводить\n сложные предложения. Это новое предложение будет\n также выведено.<br>";
echo "escap характеристики \" Как приведенные \".";
// можно использовать переменные внутри выражений echo
$foo = "footoo";
$bar = "boodoo";
echo "foo is $foo"; // foo is footoo
// при использовании одиночных кавычек значение переменной
// печататься не будет, а будет выведено просто имя этой переменной
echo 'foo is $foo'; // foo is $foo
// когда вы не используете кавычки, а просто указываете переменные,
//конструкция echo выведет значения этих переменных
echo $foo; // footoo
echo $foo,$bar; // footooboodoo
//так как echo не является функцией,
//приведенный ниже код является неверным
($some_var) ? echo('true'): echo('false');
```

```
// а приведенный ниже пример будет работать
($some_var) ? print('true'): print('false'); // функция print
echo ($some_var) ? 'true': 'false';
?>
```

Описанные выше функции работают в PHP 3 и PHP 4.

25.5. Деление и соединение строк

PHP способен производить разнообразные преобразования со строками, к этим преобразованиям также можно отнести деление и соединение строк. За реализацию этих операций отвечают следующие функции:

- explode(),
- implode(),
- join(),
- strtok().

Функция explode() разбивает строки на строки. Чтобы разбить заданную строку, необходимо указать, относительно чего будет осуществляться разделение. Например, возможно множество вариантов: относительно пробелов, символов и т. д. Синтаксис функции explode():

```
array explode(string separator, string string [, int limit])
```

Функция explode() возвращает массив значений, полученных при делении заданной строки. Сама строка указывается вместо параметра string, параметр separator является как раз той точкой, относительно которой и будет происходить процесс деления. В этом вы убедитесь, изучив приведенный пример. Если установлен параметр limit, полученный массив будет состоять из максимального количества ограниченных элементов, где последний элемент будет содержать полную строку.

Параметр limit был добавлен в PHP 4.0.1.

Приведем пример:

```
<?php
// первый блок
$part = "Часть1 Часть2 Часть3 Часть4 Часть5 Часть6 Часть7";
$piec = explode(" ", $part);
print_r($piec);
echo "<br>";
// второй блок
```

```
$partl = "Часть1 Часть2 Часть3 Часть4 Часть5 Частьб Часть7";
$piecl = explode("т", $partl);
print_r ($piecl);
echo "<br>";
// третий блок
$part2 = "Часть1 Часть2 Часть3 Часть4 Часть5 Частьб Часть7";
$piec2 = explode ("ц", $part2);
print_r($piec2);
echo "<br>";
?>
```

Данную программу можно разделить на три блока. В первом блоке происходит разделение строки с учетом пробела. Функция explode () будет осуществлять исследование строки, проверяя последовательно каждый символ. Как только она найдет пробел, автоматически произойдет перенос указанного участка символов в массив. Затем функция продолжает проверять строку. После того как найдены все пробелы, будет создан массив символов разделенного предложения.

Во втором блоке происходит практически то же самое, только функция осуществляет поиск не по пробелу, а по букве «т». Это приведено для того, чтобы показать вам, что предложение можно разделить не только по словам, как кажется на первый взгляд, но и по символам с учетом указанного символа в строке параметров separator.

Третий блок программы приведен для того, чтобы вы могли сравнить ,что произойдет, если функция осуществит разделение строки, и что будет происходить, если функция не найдет указанного символа. В результате работы этой функции на экран будет выведено:

```
Array ([0] => Часть1 [1] => Часть2 [2] => Часть3 [3] => Часть4 [4] =>
Часть5 [5] => Частьб [б] => Часть7)

Array ([0] => Час [1] => ь1 Час [2] => ь2 Час [3] => ь3 Час [4] => ь4
Час [5] => ь5 Час [6] => ьб Час [7] => ь7)

Array ([0] => Часть1 Часть2 Часть3 Часть4 Часть5 Частьб Часть7)
```

Каждая отдельная строка соответствует блоку. Во второй строке деление осуществлялось при помощи символа «т». Обратите внимание, что этого символа нет.

ВНИМАНИЕ

В процессе деления функция explode () просто удаляет тот символ, относительно которого осуществляется деление строки. Другими словами, после того как функция удаляет символ, осуществляется процесс записи содержимого массива.

Теперь осуществим обратный процесс. Все, что мы делали при помощи функции explode(), вернем обратно, т. е. соединим элементы полученного ранее массива в строку. Для решения этой задачи идеально подходит функция implode(). Она имеет следующий синтаксис:

```
string implode(string glue, array pieces)
```

В качестве параметра glue необходимо подставлять символ, который будет соединять два элемента массива. Все аналогично функции explode(), только в обратном порядке. Параметр pieces отвечает за массив, элементы которого мы собираемся соединить.

Приведем пример:

```
<?php
echo "<h3>Осуществляется процесс деления строки при помощи функции explode()</h3><br>";
$part = "Часть1 Часть2 Часть3 Часть4 Часть5 Частьб Часть7";
$piec = explode (" ", $part);
print_r($piec);
echo "<br>";
$part1 = "Часть1 Часть2 Часть3 Часть4 Часть5 Частьб Часть7";
$piec1 = explode("т", $part1);
print_r($piec1);
echo "<br>";
$part2 = "Часть1 Часть2 Часть3 Часть4 Часть5 Частьб Часть7";
$piec2 = explode("ц", $part2);
print_r($piec2);
echo "<br><br>";
echo "<h3>Осуществляется процесс соединения при помощи функции implode()</h3><br>";
$string = implode(" ", $piec);
$string1 = implode("т", $piec1);
$string2 = implode("ц", $piec2);
echo $string."<br>";
echo $string1."<br>";
echo $string2."<br>";
?>
```

В начале примера осуществляется процесс разделения строки на массив символов, затем при помощи функции implode () выполняется обратный процесс. Обратите внимание, как происходит установление параметров этой функции и что получается в результате:

```
Осуществляется процесс деления строки при помощи функции explode()
Array ([0] => Часть1 [1] => Часть2 [2] => Часть3 [3] => Часть4
[4] => Часть5 [5] => Часть6 [б] => Часть7)
Array ([0] => Час [1] => ь1 Час [2] => ь2 Час [3] => ь3 Час [4] =>
ь4 Час [5] => ь5 Час [6] => ьб Час [7] => ь7)
Array ([0] => Часть1 Часть2 Часть3 Часть4 Часть5 Частьб Часть7)
```

Осуществляется процесс соединения при помощи функции imploded ()

```
Часть1 Часть2 Часть3 Часть4 Часть5 Частьб Часть7
Часть1 Часть2 Часть3 Часть4 Часть5 Частьб Часть7
Часть1 Часть2 Часть3 Часть4 Часть5 Частьб Часть7
```

Как видно, все три строки приняли первоначальный вид.

Функция `join()` является полной копией функции implode (). Синтаксис этих функций аналогичный. Неизвестно, для чего необходимо было создавать две идентичные функции. Возможно потому, что функция implode () является довольно-таки старой и разработчики PHP решили ввести новую функцию. Вероятно, с появлением новых версий языка разработчики смогут просто добавить какие-либо параметры и функция станет отличной от предыдущей, но довольно-таки знакомой разработчикам.

Если есть необходимость разделить строку на определенное количество символов (например, с пробелом в качестве разделителя), то для этого используют функцию strtok (). Ее синтаксис:

```
string strtok(string arg1, string arg2)
```

Первый параметр функции strok () — argl указывает разделитель, второй параметр — arg2, указывает непосредственно строку, которую нужно разделить при помощи функции strtok ().

Следует отметить, что только первый вызов функции strtok () использует строковый аргумент. Для каждого последующего вызова функции strtok () необходим только разделитель, так как это позволяет контролировать положение в текущей строке.

Чтобы разбить новую строку, нужно еще раз вызвать strtok ().

СОВЕТ

Вы можете вставлять несколько разделителей в параметр. Строка будет разделяться при обнаружении любого из указанных символов.

Также будьте внимательны к разделителям «0». Это может вызвать ошибку в определенных выражениях.

Рассмотрим пример:

```
<?php
$part = "Часть1 Часть2 Часть3 Часть4 Часть5 Часть6 Часть7";
echo $part."<br>";
$piec = strtok($part," ");
while ($piec)
{
echo $piec;
$piec = strtok(" ");
}
?>
```

Рассмотрим эту же строку и произведем процесс деления, учитывая, что разделителем у нас будет служить обычный пробел. Результат скрипта будет следующим:

```
Часть1 Часть2 Часть3 Часть4 Часть5 Часть6 Часть7
Часть1Часть2Часть3Часть4Часть5Часть6Часть7
```

Как видно на первый взгляд, это две строки, хотя на самом деле так сказать нельзя. Первая совокупность слов действительно представляет строку, так как у нее существуют положенные пробелы между словами. А нижняя — просто последовательность символов, так как отсутствуют пробелы между словами, хотя на самом деле они были. Таким способом можно осуществлять любое разбиение строк с учетом необходимого разделителя.

Описанные выше функции деления и сложения строк используются в PHP 3 и PHP 4.

25.6. Работа с кодом HTML

Сама работа программиста на PHP связана так или иначе с HTML. Поэтому прежде чем приступать к изучению PHP, необходимо как минимум выучить основные понятия HTML. Не сомневаемся, что все вышесказанное вас не касалось, так как вы

уже на должном уровне владеете HTML. В этом параграфе мы рассмотрим функции работы с кодом HTML.

Сам процесс создания скриптов неразрывно связан с HTML-тегами, поэтому было бы странно, если бы разработчики PHP не подумали в первую очередь о нас с вами и не создали функции работы с кодом HTML.

Мы расскажем о следующих функциях:

- `nl2br()`,
- `htmlspecialchars()`,
- `htmlentities()`,
- `get_meta_tags()`.

Одна из самых простых функции `nl2br()` переводит символы новой строки в HTML-тег разрыва строки. Синтаксис этой функции:

```
string nl2br(string string);
```

Возвращает string с
, вставляемыми перед каждой новой строкой.

Чтобы скрипт выводил строку с тегами и эти теги не воспринимались браузером как мета-теги, а, наоборот, отражались на экране браузера, используют функцию htmlspecialchars (). Она осуществляет перевод специальных символов в код HTML (см. табл. 25.1). Рассмотрим синтаксис функции:

```
string htmlspecialchars(string string [, int quote_style])
```

Некоторые символы имеют специальное значение в HTML и должны быть представлены HTML-объектами, если им необходимо сохранить свои значения. Функция htmlspecialchars () возвращает строку с некоторым преобразованием, это очень полезно использовать при частом программировании. Если нужно, чтобы все особенности HTML-объекта были преобразованы, применяют функцию htmlentities () вместо htmlspecialchars ().

Определенные символы имеют особое значение в HTML и должны быть заменены кодами HTML, если они такие имеют. Функция htmlspecialchars () возвращает строки с произведенными такими изменениями. Она полезна для очистки полученного от пользователя текста от разметки HTML (доски сообщений, гостевые книги).

Второй необязательный параметр, `quote_style`, сообщает функции, что делать с одинарными и двойными кавычками. `ENT_COMPAT` является обратно совместимым способом, который только переводит символ с двойной кавычкой и оставляет непереведенную одинарную кавычку. Если `ENT_QUOTES` установлен, и одиночные, и двойные кавычки переведутся, а если установлен `NOQUOTES` — ни одиночные, ни двойные кавычки не будут переведены.

В табл. 25.1 приведены существующие замены:

Таблица 25.1. Преобразование специальных символов

Символ	Преобразование
& (амперсанд)	**Становится** &
" (двойные кавычки)	Становится " когда ENT_NOQUOTES не установлен
' (одинарные кавычки)	Становится ' только когда установлен ENT_QUOTES
< (знак меньше)	Становится <
> (знак больше)	Становится >

ВНИМАНИЕ

Функция htmlspeeialchars о не заменяет ничего, кроме указанного в табл. 25.1. Для полной обработки необходимо обратится к функции htmlentities (). Второй параметр в функции htmlspeeialchars () был добавлен в PHP 3.0.17.

Рассмотрим пример:

```
<?php
$new = htmlspecialchars("<a href='test'>Ссылка</a>", ENT_QUOTES);
$new2 = htmlspecialchars("<h1>Текст с использованием знака
амперсанда & </h1>");

echo $new."<br>";
echo $new2."<br>";
?>
```

Из примера видно, что функция позволяет выводить в первом случае все содержимое строки на экран браузера даже с учетом одинарных кавычек, во втором случае основное внимание необходимо обратить на амперсанд (&). Результатом работы этого примера будут следующие строки:

```
<a href='test'>Ссылка</a>
<h1>Текст с использованием знака амперсанд & </hl>
```

Как видите, все осталось на своих местах и браузер даже не попытался произвести обработку приведенных тегов.

Аналогичной функцией является htmlentities (). Она позволяет переводить все возможные символы в коды HTML. Данная функция имеет синтаксис, аналогичный htmlspecialchars():

```
string htmlentities(string string [, int quote_style])
```

Все символы, которые имеют соответствующий код HTML, заменяются на этот HTML-код и будут видны в браузере.

В настоящее время применяется кодовая таблица ISO-8859-1.

Второй параметр функции htmlentities() — quote_style — был добавлен в PHP 3.0.17.

Функция get_meta_tags позволяет осуществлять работу с HTML-кодом, а в частности извлекать все содержимое атрибутов тега <meta> из файла и возвращать в массиве. Синтаксис функции get_meta_tags():

```
array get_meta_tags(string filename[, int use_include_path])
```

Функция get_meta_tags() возвращает ассоциативный массив, созданный на основании полученных атрибутов name и content тега <meta>. В этом массиве в качестве ключа, т. е. индекса ассоциативного массива, будет использоваться значение атрибута name, а само возвращаемое значение данного элемента массива будет равно значению атрибута content, указанного в теге <meta>. Отсюда следует, что в дальнейшем вы можете без труда осуществлять операции с полученными значениями данного массива.

Специальные символы в значении свойства заменяются символом «_», остальные переводятся в нижний регистр.

Элемент <meta> используется для включения различной информации о документе, а также представляет возможность сообщать дополнительные инструкции как клиентской части (браузеру), так и серверной. Он используется в форме «свойство—значение». Например, чтобы указать автора документа, используется следующая строка:

```
<meta name="Author" content="Alehander Mazurkevich">
```

В данном примере определяется свойство (Author), которому присваивается значение (Alehander Mazurkevich). Вы можете определять любые свойства и присваивать им любые значения.

ВНИМАНИЕ

Функция get_meta_tags() позволяет получать только свойства NAME и CONTENT. Другие совокупности атрибутов получить не удается.

Для примера создадим два файла. Первый будет HTML-файл, содержащий следующую совокупность тегов HTML-страницы. Файл назовем newname.html:

```
<html>
<head>
<title>Test</title>
```

```
1. <meta name="Author" content="Alehander Mazurkevich">
2. <meta http-equiv="Content-Type" content="text/html">
3. <meta name="Copyright" content ="SinkORG">
4. <meta name="Description" content="PHP, PHP3, PHP Documentation">
5. <meta name="Generator" content="FrontPage">
6. <meta name="Keywords" content="Documentation, PHP scripts">
</head>
<body>
Test
</body>
</html>
```

Здесь нас интересует пронумерованный блок с мета-тегами. Строка 1 описывает автора документа, строка 2 используется для указания инструкций серверу, 3 — информация об авторском праве, 4 — описание документа (применяется непосредственно для поисковых машин), 5 — название программы, с помощью которой создавался документ, 6 — ключевые слова (для поисковых машин). Теперь посмотрим, какие значения этих мета-тегов вернутся в ассоциативный массив. Пример скрипта приведен ниже:

```
<?php
$path = "Z:/home/localhost/www/newname.htm";
$arra = get_meta_tags ($path);
print_r($arra);
?>
```

Приведенный скрипт позволяет вывести на экран браузера ассоциативный массив значений атрибутов name и content. Значение атрибута name будет указано в качестве ключа ассоциативного массива, а принадлежащий этому атрибуту атрибут content будет указан в качестве значения массива. Получить доступ к любому из них не составит никакого труда. Полученый массив:

```
Array ([Author] => Alehander Mazurkevich [Copyright] => SinkORG
[Description] => PHP, PHP3, PHP Documentation [Generator] =>
FrontPage [Keywords] => Documentation, PHP scripts)
```

Обратите внимание, что атрибут http-equiv не был возвращен в массив.

ВНИМАНИЕ

Установка параметра use_include_path со значением, равным "Г, приведет к тому, что PHP будет пытаться открыть файл по стандартному пути include.

Описанные функции работы с HTML-тегами работают в PHP 3 и PHP 4.

В качестве исключения можно выделить функцию get_meta_tags(). Она используется в PHP 3-3.0.4 и PHP 4.

25.7. Доступ с операциями замены строк или подстрок, сравнение строк

Обычно возникает необходимость просто произвести обращение к строке либо отдельной ее части. Во время работы у вас может появиться необходимость вырезать какую-либо часть строки либо переместить ее, либо изменить, в общем произвести какие-то действия. PHP позволяет осуществлять такие задачи. В этой части главы мы рассмотрим следующие функции:

- strcasecmp(),
- strncasecmp(),
- substr(),
- strstr(),
- strnatcmp(),
- substr_replace(),
- str_replace().

Первые две функции позволяют производить сравнение строки определенного количества символов. Чтобы произвести бинарное сравнение двух строк, используют функцию strcasecmp(). Она позволяет сравнивать строки без учета регистра. Ее синтаксис:

```
int strcasecmp(string str1, string str2)
```

Параметры str1 и str2 задают значения сравниваемых строк. Функция возвращает целое число, равное нулю, если две строки идентичны с учетом двоичного исчисления. Если str1 меньше, чем str2, то функция возвратит целое значение меньше нуля. Если str1 будет больше str2, функция возвратит целое значение больше нуля. Зная эти тонкости, вы можете спокойно работать с оператором условия. В приведенном ниже примере будут рассмотрены способы сравнения.

Если нужно сравнить не всю строку сразу, а определенное количество символов этой строки, применяют функцию strncasecmp(). Она имеет следующий синтаксис:

```
int strncasecmp(string str1, string str2, int len)
```

На месте параметра len указывается количество символов, отсчитываемых с начала строк str1 и str2. Это целое число. Все, что касается функции strcasecmp(), работает и при использовании функции strncasecmp(). Приведем пример, характеризующий функции strcasecmp() и strncasecmp():

```
<?php
$varl = "TABLE";
$var2 = "table";
if (!strcasecmp ($varl, $var2))
{
echo 'Значение переменной $varl эквивалентно $var2 без учета регистра <br>';
}

$var3 = "World";
$var4 = "Hello";
if ($int = strcasecmp($var3, $var4))
{
echo ' Значения переменных $var3 и $var4 не одинаковы между собой<br>';
}
$var5 = "World";
$var6 = "He";
if ($int = strcasecmp($var5, $var6) )
{
echo 'Значения переменных $var5 и $var6 не одинаковы между собой<br>';
}
$var7 = "Vitalik";
$var8 = "Vitiya";
if ( !strncasecmp($var7,$var8, 3))
{
echo "Первые три символа эквивалентны <br>";
}
?>
```

В первом случае приведенного примера происходит сравнение двух строк, одна из которой приведена с учетом регистра, другая, наоборот, — без учета. Во второй части программы происходит сравнение двух строк с одинаковой длиной, но раз -

ными значениями. Функция сравнивает бинарный код этих строк, и за счет того, что значения этого кода будут различны, функция выдаст соответствующий результат. В третьей части программы происходит обычное сравнение количества символов строк, только эти строки имеют разные длины. Четвертая часть — пример сравнения строк при помощи функции strncasecmp(). Результат этого скрипта приведен ниже:

```
Значение переменной $varl эквивалентно $var2 без учета регистра
Значения переменных $var3 и $var4 не одинаковы между собой
Значения переменных $var5 и $var6 не одинаковы между собой
Первые три символа эквивалентны
```

Каждый отдельный блок скрипта выводит одну строку согласно полученному результату.

Функция strcasecmp() работает в PHP 3–3.0.2 и PHP 4; функция strncasecmp() — в PHP 4–4.0.2.

Очень часто при работе со строками программисту необходимо получить какую-либо определенную последовательность символов имеющейся строки. Для этого используется функция substr(). Она позволяет получать любую часть строки с учетом указанных параметров. Ее синтаксис:

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

Функция возвращает вырезанную часть символов строки string, определяемую параметрами start (начало) и length (длина).

Если параметр start положительный, то возвращаемая строка будет начинаться со start-го символа строки string.

```
$string = substr("abcdef", 1); // возвратит "bcdef"
$string = substr("abcdef", 1, 3); // возвратит "bcd"
```

Если параметр start отрицательный, то возвращаемая строка будет начинаться со start-го символа от конца строки string.

```
$string = substr("abcdef", -1); // возвратит "f"
$string = substr("abcdef", -2); // возвратит "ef"
$string = substr("abcdef", -3, 1); // возвратит "d"
```

Если параметр length указан и он положительный, то возвращаемая строка закончится и через length символов от начала start. Если length указан и он отрицательный, то возвращаемая строка закончится через length от конца строки string.

```
$rest = substr("abcdef", 1, -1); // возвратит "bcde"
```

Функция strstr() позволяет находить указанный символ и выводить все далее следующие за ним символы в строку. Другими словами, осуществляет поиск указанного символа и при встрече первого символа, который был указан, функция возвратит оставшуюся часть строки. Приведем синтаксис функции:

```
string strstr(string haystack, string needle)
```

В качестве параметра haystack указывается строка, вместо параметра needle — тот фрагмент символов, который необходимо найти. После указания всех параметров функция произведет возврат строки, начиная с той части, откуда был указан параметр needle. При этом указанные символы needle не будут удалены, а будут добавлены в начало строки. Приведем пример:

```
<?php
$string='Sasha+Igor';
$foo = strstr ($string, '+');
print $foo;
$string1 = 'Sasha+Igor+Sveta';
$fool = strstr ($string1, '+S');
print $fool;
?>
```

Результат программы:

```
+Igor
+Sveta
```

ВНИМАНИЕ

Функция strstr () является чувствительной к регистру. Функция stristr () является полной аналогией strstr(), но не чувствительна к регистру.

Иногда возникает необходимость произвести сортировку строк. При этом можно осуществлять стандартную сортировку строк и натуральную. Например:

```
<?php
$arr1 = $arr2 =
array("Pic12.jgp","Pic10.jgp","Pic2.jgp","Pic1.jgp");
echo "Стандартная сортировка строк\n";
usort($arr1,"strcmp");
print_r($arr1);
echo "\nНатуральный порядок сортировки строк\n";
```

```
usort($arr2,"strnatcmp");
print_r($arr2);
?>
```

В примере для сортировки были использованы функции strcmp() и strnatcmp(). Результат работы этого скрипта:

```
Стандартная сортировка строк
Array ( [0] => Pic1.jgp
[1] => Pic10.jgp
[2] => Pic12.jgp
[3] => Pic2.jgp )
Натуральный порядок сортировки строк
Array ( [0] => Pic1.jgp
[1] => Pic2.jgp
[2] => Pic10.jgp
[3] => Pic12.jgp )
```

Функции strnatcmp () и strcmp() практически аналогичны и имеют следующий синтаксис:

```
int strnatcmp(string str1, string str2)
int strcmp(string str1, string str2)
```

Функция strnatcmp () сортирует строки с учетом натурального порядка. Она возвращает значение, равное нулю, если строки идентичны. Обратите внимание, что функции strnatcmp() и strcmp() чувствительны к регистру. Функция strcmp() осуществляет стандартную сортировку строк, как показано в примере.

Для замены текста в пределах части строки в PHP существует несколько функций. Функция substr_replace() позволяет заменять указанный текст в заданной части строки. Рассмотрим синтаксис функции substr_replace():

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

Функция substr_replace() возвращает строку, измененную с учетом указанных параметров. В качестве параметров задаются string — строка, которую мы собираемся изменять, replacement — та строка, при помощи которой мы собираемся изменять, start — целое значение — количество символов, отсчитываемых от начала строки, length — длина строки, которую будем изменять, подставив строку replacement вместо указанного количества символов.

Приведем пример:

```
<?php
$var = ' QWERTY:ASDFGH';
echo "Первоначальная строка: $var<hr>\n";
echo 'Замена с указанными параметрами: ($var, "XXX", 0), в результате получаем: <br> ';
echo substr_replace($var, 'XXX', 0). "<br><br>\n";
echo 'Замена с указанными параметрами: ($var, "XXX", 0, strlen ($var)) , в результате получаем:<br>';
echo substr_replace($var, 'XXX', 0, strlen ($var)) . "<br><br>\n";
echo 'Замена с указанными параметрами: ($var, "XXX", 0, 0), в результате получаем: <br>';
echo substr_replace($var, 'XXX', 0, 0). "<br><br>\n";
echo 'Замена с указанными параметрами: ($var, "XXX", -10, 5), в результате получаем:<br> ';
echo substr_replace($var, 'XXX', -10, 5). "<br><br>\n";
echo 'Замена с указанными параметрами: ($var, "XXX", 10, -1) , в результате получаем: <br>';
echo substr_replace($var, 'XXX', 10, -1). "<br><br>\n";
echo 'Замена с указанными параметрами: ($var, "XXX", -7, -1), в результате получаем:<br> ';
echo substr_replace($var, 'XXX', -7, -1). "<br><br>\n";
echo 'Замена с указанными параметрами: ($var, "", 10, -1), в результате получаем: <br>';
echo substr_replace($var, '', 10, -1). "<br><br>\n";
?>
```

Результатом этого примера будут следующие строки:

Первоначальная строка: QWERTY:ASDFGH

Замена с указанными параметрами: ($var, "XXX", 0), в результате получаем:

XXX

Замена с указанными параметрами: ($var, "XXX", 0, strlen ($var)) , в результате получаем:

XXX

Замена с указанными параметрами: ($var, "XXX", 0, 0), в результате получаем:

XXXQWERTY:ASDFGH

Замена с указанными параметрами: ($var, "XXX", -10, 5), в результате получаем:

QWEXXXSDFGH

Замена с указанными параметрами: ($var, "XXX", 10, -1), в результате получаем:

QWERTY:ASDXXXH

Замена с указанными параметрами: ($var, "XXX", -7, -1), в результате получаем:

QWERTYXXXH

Замена с указанными параметрами: ($var, "", 10, -1), в результате получаем:

QWERTY:ASDH

Функция substr_replace 0 была добавлена в PHP 4.0.

Функция str_replace () относится к этому же ряду функций, осуществляющих замену строк. Функция str_replace () подчиняется следующему синтаксису:

mixed str_replace(mixed search, mixed replace, mixed subject)

Функция str_replace() возвращает постоянную строку с измененными символами. Параметр search представляет собой ту часть строки, которую мы будем заменять на новую, указанную в качестве параметра replace. В свою очередь сама строка указывается вместо параметра subject. В результате вместо совокупности символов search, содержащихся в строке subject, получается строка mixed с совокупностью символов, указанных в параметре replace. Приведем пример, характеризующий работу функции str_replace():

```
<?php
$string = "Мама купила красную шапку";
$strnew = str_replace("красную","фиолетовую", $string);
echo "Первоначальная строка: " .$string."<br><br>";
echo "Преобразованная строка: ".$strnew."<br><br>";
?>
```

Результатом данного скрипта будут следующие строки:

Первоначальная строка: Мама купила красную шапку

Преобразованная строка: Мама купила фиолетовую шапку

Отсюда видно, что слово красную функция str_replace () заменила на указанное слово фиолетовую.

Функция str_replace () была добавлена в PHP 3.0.6.

Функция str_replace() работает в PHP 3–3.0.6 и PHP 4; substr_replace() — в PHP4–4.0b4; остальные функции — в PHP 3 и PHP 4.

25.8. Операции поиска символов

Рассмотрим способы получения доступа к указанным символам в строке, а также получения позиции символов при помощи следующих функций:

- strpos(),
- strrpos().

Функция strpos () находит первую позицию указанного символа и возвращает целое число, равное количеству символов с начала этой строки. Приведем синтаксис функции strpos ():

```
int strpos(string haystack, string needle [, int offset])
```

Strpos () возвращает целое значение, равное количеству символов, отсчитанных с конца. Параметр haystack устанавливает строку, в которой будет осуществляться поиск указанного символа. Сам символ устанавливается параметром needle.

Следует отметить, что needle в этом случае может быть только единственным символом.

Если в качестве параметра needle указывается строка, то только первый символ будет использован.

Если needle не найден, то возвращается false. Если параметр needle не является строкой, то он переводится в десятичное число и рассматривается как числовое значение символа.

Третий параметр offset отвечает за количество символов, от которых будет производиться поиск указанного символа needle. В этом случае длина, возвращенная функцией strpos (), все равно будет отсчитываться с начала строки. Приведем пример:

```
<?php
$string = "Мама купила красную шапку";
$strnew = strpos($string, "п");
$strnew1 = strpos($string, "п",10);
echo "Первоначальная строка: ".$string."<br><br>";
echo "Количество символов, начиная с начала строки до первого
```

```
символа 'п': ".$strnew."<br><br>";

echo "Количество символов, начиная с начала строки до первого
символа 'п', пропустив при этом первых 10 символов:
".$strnew1."<br><br>";

?>
```

Функция возвратит две строки, первая из них будет соответствовать количеству символов, отсчитанных с начала указанной строки, при этом третий параметр указан не был. Во втором случае был указан третий параметр, в результате чего поиск символа «п» будет начинаться не с начала строки, а с десятого символа. Далее будет осуществляться поиск до первого символа «п», после чего функция подсчитает количество символов до найденного символа «п» и произведет возврат строки, равной целому числу символов. В этом можно убедиться, обратившись к результату приведенного скрипта:

```
Первоначальная строка: Мама купила красную шапку

Количество символов, начиная с начала строки до первого символа
'п': 7

Количество символов, начиная с начала строки до первого символа
'п', пропустив при этом первых 10 символов: 22
```

Функция strrpos () по принципу работы аналогична функции strpos (), единственное ее отличие заключается в том, что функция strrpos () находит позицию последнего появления символа в строке. Функция имеет следующий синтаксис:

```
int strrpos(string haystack, char needle)
```

Как было уже сказано, strrpos () возвращает номер позиции последнего появления символа needle в строке haystack.

Описанные функции используются в PHP 3 и PHP 4.

25.9. Перевод строк в верхний и нижний регистры

В каждой стране мира существуют свои стандарты письма. Но нет ни одной страны, где можно было бы обходиться без слов с большой буквы. Существуют даже ситуации, когда слово, находящее в середине предложения, должно писаться с большой буквы (например, существительное в немецком языке). Объясним, что значит верхний и нижний регистры.

Когда вы используете строку только с прописными буквами, это называется верхним регистром. В противном случае — нижним регистром. В PHP для изменения регистра используются следующие функции:

- strtolower(),
- strtoupper(),

- ucfirst(),
- ucwords().

Все приведенные функции позволяют переводить строки символов и слов в верхний или нижний регистр. Все приведенные функции по своей структуре имеют аналогичный синтаксис:

```
string strtoupper(string string)
```

Строка, которую необходимо преобразовать, задается в качестве параметра string.

Данные функции выполняют следующие функции (табл. 25.2):

Таблица 25.2. Функции преобразования строк

Название функции	Операция, выполняемая этой функцией
strtolower()	Переводит строку в нижний регистр
strtoupper()	Переводит строку в верхний регистр
ucfirst()	Переводит первый символ строки в верхний регистр
ucwords()	Переводит первый символ каждого слова строки в верхний регистр

ВНИМАНИЕ

Буквенные символы определяются текущими локальными установками.

Приведем пример, характеризующий одновременно работу всех функций, содержащихся в табл. 25.2:

```
<?php
$string = "мама купила красную шапку";
$string1 = "ШКОЛА ЗАКРЫВАЕТСЯ НА РЕМОНТ";
$strnew = strtolower($string1);
$strnew1 = strtoupper ($string);
$strnew2 = ucfirst($string);
$strnew3 = ucwords($string);
echo 'Первоначальная строка $string : '.$string."<br>";
echo 'Первоначальная строка $string1:'.$string1."<br><br>";
echo 'Преобразование строки $string1 при помощи функции
strtolower():<br>';
echo $strnew. "<br><br>";
```

```
echo 'Преобразование строки $string при помощи функции
strtoupper():<br>';
echo $strnew1. "<br><br>";
echo 'Преобразование строки $string при помощи функции
ucfirst():<br>';
echo $strnew2. "<br><br>";
echo 'Преобразование строки $string при помощи функции
ucwords():<br>';
echo $strnew3. "<br><br>";
?>
```

Результатом работы скрипта будут следующие строки:

```
Первоначальная строка $string : мама купила красную шапку
Первоначальная строка $string1 :ШКОЛА ЗАКРЫВАЕТСЯ НА РЕМОНТ
Преобразование строки $string1 при помощи функции strtolower():
школа закрывается на ремонт
Преобразование строки $string при помощи функции strtoupper():
МАМА КУПИЛА КРАСНУЮ ШАПКУ
Преобразование строки $string при помощи функции ucfirst():
Мама купила красную шапку
Преобразование строки $string при помощи функции ucwords():
Мама Купила Красную Шапку
```

Как видно, в первом случае функция полностью установила верхний регистр с учетом всех символов строки. Во втором случае все наоборот. Третий случай соответствует переводу первого символа строки в верхний регистр. Последний случай показывает, что при помощи функции ucwords () первый символ каждого слова строки будет установлен в верхний регистр.

Функция ucwords () работает в PHP 3–3.0.3 и PHP4; остальные функции работы с регистром — в PHP 3 и PHP 4.

25.10. Перевод строки в другую кодовую таблицу

Очень часто посетители сайтов жалуются на ту или иную кодировку. Вы же можете решить эти проблемы, применив функцию перевода строки из одной русской кодовой таблицы в другую. Функция имеет следующий синтаксис:

```
string convert_cyr_string(string str, string from, string to)
```

Функция возвращает строку, полученную в результате преобразований с учетом указанных параметров. Строка, передаваемая в функцию, указывается на месте str. Параметр from указывает ту кодовую таблицу, из которой будет происходить преобразование, т. е. ту, в которой находится приведенная строка в данный момент. Третий параметр to устанавливает таблицу кодировки, в которую будет переведена указанная строка.

Указание кодировки происходит посредством символов латинского алфавита:

Символ	Обозначение кодовой таблицы
k	koi8-r
w	windows-1251
i	iso8859-5
a	x-cp866
d	x-cp866
m	x-mac-cyrillic

Приведем пример:

```
<?php
$string = "Мама купила красную шапку";
echo "Начальная строка: ".$string. "<br><br>";
$strnew = convert_cyr_string($string, w, k) ;
echo "Кодовая таблица koi8-r: " .$strnew."<br><br>";
$strnew1 = convert_cyr_string($strnew,k,i) ;
echo "Кодовая таблица iso8859-5: " .$strnew1. "<br><br>";
$strnew3 = convert_cyr_string($strnew1,i,d) ;
echo "Кодовая таблица x-cp866: " . $strnew3."<br><br>";
$strnew4 = convert_cyr_string ($strnew3,d,m);
echo "Кодовая таблица x-mac-cyrillic: " .$strnew4."<br><br>";
$strnew5 = convert_cyr_string($strnew4,m,w) ;
echo "Кодовая таблица windows-1251 : " . $strnew5."<br><br>";
?>
```

В примере преобразовываются строки из одной кодировки в другую. Как правило, вся работа в браузере осуществляется в стандартной кодировке Windows-1251, которая является одной из самых распространенных. Перевод строк мы начали именно с этой кодировки. Покажем, что у нас получилось в результате этого преобразования:

```
Начальная строка: Мама купила красную шапку
Кодовая таблица koi8-r: нБНБ ЛХРЙМБ ЛТБУОХА ШБРЛХ
Кодовая таблица iso8859-5: jРЬР ЪгЯШЫР ЪаРбЭго иРЯЪг
Кодовая таблица x-cp866: Њ ¬ ЄгЇЕ« Єа б-го и ЇЄг
Кодовая таблица x-mac-cyrillic: Њама купила красную шапку
Кодовая таблица windows-1251: Мама купила красную шапку
```

Функция `convert_cyr_string()` **работает в PHP 3–3.0.6 и PHP 4.**

Заключение

Спектр функций, работающих со строками, настолько широк, что позволяет PHP решать различные задачи. Строки имеют большой диапазон применения, поэтому профессиональное владение функциями работы со строками позволяет легко и быстро находить рациональные решения.

Глава 26
Функции работы с файлами

Уже в самом начале развития компьютерной техники люди знали, что доступ к информации, содержащейся на компьютерах, будет производиться непосредственно через файлы. Файлы представляют собой совокупность имен, содержащихся на винчестере. В наши времена, получив такое широкое распространение, файлы заняли свое место в компьютерном мире. С развитием программного обеспечения, появлением все новых и новых редакторов, также происходит создание и новых типов файлов, но это уже другая тема, для которой, возможно, будет мало и книги.

Если есть файлы, то должны быть и средства работы с ними. Поэтому при написании какого-либо языка программирования обязательно уделяется внимание функциям, операциям или другим подходам для работы с файлами. Файл — это совокупность символов, имеющая свое расширение. Данная совокупность символов не может повторяться, поэтому каждый файл имеет имя, отличное от предыдущего. Конечно же, неотъемлемым атрибутом любого языка программирования помимо всего прочего является также и способность как можно более плотно взаимодействовать с файловыми операциями. Под файловыми операциями понимается всевозможное преобразование, копирование, получение размера, даты, чтение и т. д. необходимого файла. В Сети работа с файлами вообще является распространенным занятием, поэтому рассматриваемый язык не может быть обделен функциями работы с файлами.

PHP содержит развитую систему функций, работы с файлами. Программист, овладев данными функциями, сможет без каких-либо трудностей решить любую задачу, связанную с созданием программ, выполняющих работу с файлами. В этой главе рассмотрены следующие темы:

- получение пути файлов;
- копирование файлов;
- основные операции над файлами;
- чтение и проверка файлов;
- определение атрибутов файлов;
- создание и удаление директории;
- доступ к строке файлового пути;
- получение информации о файле;
- создание уникального имени;
- установка времени модификации файла;
- разные функции.

26.1. Получение пути файлов

Перед тем, как производить какие-либо манипуляции с файлами, необходимо их получить — либо путь, либо само имя файла. Для этой операции в PHP имеется две функции:

- `basename()`,
- `dirname()`.

Когда вам известен полный путь к файлу и его имя, то получить строку с его именем помогает функция `basename()`. Функция возвращает не полный путь файла, а само имя файла с его расширением. `Basename()` имеет следующий синтаксис:

```
string basename (string path)
```

Вместо параметра path вписывается сам путь к файлу либо же переменная, содержащая этот путь. В результате выполнения функция возвратит строку со значением имени файла. Например:

```
<?php
$path = "/dir/dir1/files/my.doc";
$file = basename($path);
$file1 = basename("/dir/dir1/files/my.doc");
$file2 = basename("\dir\dir1\files\my.doc");
echo $file1,"<br>";
echo $file,"<br>";
```

```
echo $file2,"<br>";
?>
```

Пример характеризует работу функции basename() во всех ее проявлениях. При первом вызове функции basename() в качестве параметра мы просто применили переменную, во втором случае вместо переменной вписали сам путь. Обратите внимание, что путь записывается в двойных кавычках. В третьем случае мы воспользовались функцией basename(), чтобы показать, как зависит изменение направления слеша.

ВНИМАНИЕ

Правила хорошего тона программирования в PHP подразумевают использование слеша только в таком направлении — \. Хотя в Windows возможно использование как в прямом - \, так и в обратном - /.

Полученный результат:

```
my.doc
my.doc
my.doc
```

Как видно, все имена получены верно независимо от способа задания параметров.

Когда возникает необходимость получить непосредственно путь без имени файла, т. е. совокупность каталогов, через которые необходимо пройти, чтобы добраться до желаемого файла, используется функция dirname(). Ее синтаксис:

```
string dirname(string path)
```

В этой функции вместо параметра path записывается путь к файлу либо переменная, значение которой указывает на путь к файлу. Например:

```
<?php
$path = "/dir/dir1/files/my.doc";
$file = dirname($path);
$file1 = dirname("/dir/dir1/files/my.doc");
$file2 = dirname ("\dir\dir1\files\my.doc");
echo $file1,"<br>";
echo $file,"<br>";
echo $file2,"<br>";
?>
```

В результате действия этой функции получается:

```
/dir/dir1/files
/dir/dir1/files
\dir\dirl\files
```

Как видно, теперь вместо имен файлов получили путь к нему.

Третий способ указания пути является нерациональным и применять его на практике не рекомендуется. Приведен он здесь, только чтобы показать, что использование такого слеша не вызовет ошибки при написании скрипта.

Описанные нами функции используются в PHP 3 и PHP4.

26.2. Копирование файлов

Ни одна операционная система не обходится без таких элементарных функций, как копирование и переименование файлов. Только представьте на минуту, что вам предлагают установить сверхбыструю операционную систему, но с одним условием, что там будут отсутствовать функции копирования или переименования файлов. Уверены, что бы ни предлагали вам, вы не согласитесь установить такую операционную систему на свой компьютер, и все дело в каких-то, на первый взгляд совсем незаметных, функциях. Собственно говоря, так и любой язык программирования проиграет в популярности в случае отсутствия этих функции. С течением времени эти функции станут незаменимыми помощниками вам при работе в PHP. Рассмотрим две функции:

- `copy()`,
- `rename()`.

Функция copy () копирует файл из одного места в другое, предварительно указанное при помощи вносимых параметров этой функции. Функция copy () подчинена следующему синтаксису.

```
int copy(string source, string dest)
```

В данной функции вместо параметра source указывается строка пути файла с именем копируемого файла, также можно использовать переменные. Если указывается просто имя файла без дополнений (каталога, диска и т. д.), то интерпретатор подразумевает, что копируемый файл находится в том же месте, где и сам исполняемый файл, т. е. файл, в котором содержится скрипт. Параметр dest указывает на путь, куда будет копироваться файл. Сама функция возвращает значение true в результате успешного копирования (если не возникнет никаких ошибок), в противном случае — false. Например:

```
<?php
$path = "Z:/home/localhost/www/1.txt";
$file = basename($path);
```

```
if (copy($file, "Z:/home/localhost/1.txt"))
{
if (copy ($file, "Z:/home/localhost/2.txt"))
echo"Копирование файла $file произведено удачно!";
}
else
{
echo"Копирование файла $file не выполнено!<br>";
echo"Исправьте ошибки и проделайте операцию копирования заново";
}
?>
```

Опишем работу приведенного примера. Сначала задается путь к файлу 1.txt После этого используется функция basename(), чтобы получить само имя этого файла. Имя файла передается в переменную $file. Именно с этим именем будут производиться манипуляции, т. е. именно этот файл будет копироваться. Если произойдет успешное копирование файла 1.txt в указанном направлении: Z:/home/localhost/1.txt, то указатель выполнения скрипта перейдет в следующий блок, т. е. опять будет выполняться проверка условия. Мы воспользовались этой операцией, чтобы заострить ваше внимание на том, что функция copy() помимо копирования файлов может также производить копирование файла с изменением его имени. При выполнении программы файл 1.txt будет скопирован два раза. Первый раз — Z:/home/localhost/1.txt, второй — Z:/home/localhost/2.txt). В директории localhost появится два идентичных файла, только с разными именами. Один будет иметь имя 1.txt, второй 2.txt.

ВНИМАНИЕ

При использовании функции copy() необходимо четко указывать путь, куда будет копироваться файл, а также его имя. Если вы просто укажете директорию, в которую необходимо скопировать файл, программа вызовет ошибку.

В результате выполнения программы на экран будет выведено:

Копирование файла 1.txt произведено удачно!

При указании каких-либо неверных параметров программа вернет следующие строки:

Копирование файла 1.txt не выполнено!

Исправьте ошибки и проделайте операцию копирования заново

Если нужно переименовать файл, используют функцию rename(). Она перемещает файл из одного места и записывает в другое, при этом изменяя его имя. Функция remane() имеет следующий синтаксис:

int rename(string oldname, string newname)

Параметр oldname указывает имя файла, который мы собираемся переименовать. Следующий за ним параметр указывает новое имя файла — newname. Функция rename() в случае положительного исхода возвращает значение true, в случае отрицательного — false. Например:

```
<?php
$path = "Z:/home/localhost/www/1.txt";
$file = basename ($path);
if (rename($file, "Z:/home/localhost/newname.txt") )
{
echo"Изменение имени файла $file произведено удачно!";
}
else
{
echo"Изменение имени файла $file не выполнено!<br>";
echo"Исправьте ошибки и проделайте операцию копирования заново";
}
?>
```

В результате выполнения этого скрипта произойдет удаление файла 1.txt по пути Z:/home/localhost/www/1.txt, перемещение его по пути Z:/home/localhost/ с новым именем newname.txt, хотя все параметры данного файла будут полностью соответствовать файлу 1.txt, т. е. произойдет просто переименование файла и перемещение его в другой каталог. Результатом работы данного скрипта будет следующая строка:

Изменение имени файла 1.txt произведено удачно!

Описанные функции используются в PHP 3 и PHP 4.

26.3. Основные операции над файлами

Изучив материал этого параграфа, вы научитесь получать доступ к файлам для работы с их содержимым: производить чтение строк файлов, перемещать указатель в конец и определять, что именно это место и является завершением файла. Рассмотрим следующие функции:

- fopen (),
- fclose(),
- popen(),
- pclose(),
- feof(),
- fgets(),
- fgetss(),
- fread().

Очень часто, чтобы получить доступ к файлу, необходимо установить на него указатель (т. е. что-то вроде курсора, только невидимый). Для выполнения этой операции в PHP существует несколько функций, мы рассмотрим одну из самых главных, часто используемых и наиболее эффективных — функцию fopen(). Она позволяет открыть файл или URL. Функция fopen() имеет следующий синтаксис:

```
int fopen(string filename, string mode [, int use_include_path])
```

Если параметр filename начинается с http:// (без учета регистра), то открывается соединение HTTP 1.0 с заданным сервером и указатель файла возвращается на начало текста ответа. Поскольку редиректы HTTP не обрабатываются, нужно включать в указание директории завершающие слеши.

Если filename начинается с ftp:// (без учета регистра), то открывается ftp-соединение с задазанным сервером и указатель возвращается на искомый файл. Если сервер не поддерживает режим пассивного ftp, данная операция завершится ошибкой. Можно открывать файлы как для чтения, так и для записи через ftp (но не обе операции одновременно).

Если filename начинается как-нибудь иначе, открывается файл вашей файловой системы и указатель возвращается на открытый файл.

Если при открытии файла происходит ошибка, функция возвращает false.

Параметр mode в зависимости от поставленной задачи может принимать следующие значения.

r — открыть только для чтения, помещает указатель на начало файла.

r+ — открыть для чтения и для записи, помещает указатель на начало файла.

w — открыть только для записи, помещает указатель на начало файла и очищает все содержимое файла. Если файл не существует, то создается новый.

w+ — открыть для чтения и для записи, помещает указатель на начало файла и очищает все содержимое файла. Если файл не существует, создается новый.

a — открыть только для записи, помещает указатель на конец файла. Если файл не существует, создается новый.

a+ — открыть для чтения и для записи, помещает указатель на конец файла. Если файл не существует, создается новый.

ВНИМАНИЕ

Параметр mode также может содержать символ `'b'` — используется в системах, различающих бинарные и текстовые файлы (не используется в Unix). Если данное значение не имеет смысла, оно игнорируется.

Третий параметр — `use_include_path` — устанавливается как «1». Его используют, если нужно исследовать файл в `include_path`, в файле конфигурации `php.ini`.

Например:

```
1. UNIX include_path
include_path=.:/home/httpd/php-lib
2. Windows include_path
include_path=".;c:\www\phplib"
```

Рассмотрим примеры работы функции `fopen()`:

```
1. $filepointer = fopen("/home/localhost/file.txt", "r");
2. $ filepointer = fopen("/home/localhost/file.gif", "wb");
3. $ filepointer = fopen("http://www.any_name_domain.com/", "r");
4. $ filepointer = fopen("ftp://user:password@example.com/", "w");
```

Мы пронумеровали строки для того, чтобы можно было легче описать последовательность функций. При программировании нумерация не применяется. В первой строке происходит открытие файла по имени `file.txt` с установленным параметром `"r"`. В этом случае указатель `$filepointer` будет помещен на начало открытого файла, но этот файл можно только читать, производить какие-либо другие операции, например запись или удаление каких-либо текстовых блоков, нельзя.

Вторая строка — на основании указанных параметров можно сделать вывод, что файл `file.gif` открывается только для записи. После того как он был открыт, сразу происходит помещение указателя на начало файла и очистка всего содержимого файла. Если файл не существует, то создается новый файл. За это все отвечает параметр `"w"`, но помимо этого был указан и параметр `"b"`, который используется в системах, различающих бинарные и текстовые файлы.

В третьей строке примера доступ к файлу осуществляется посредством протокола `http://` (HyperText Transfer Protocol).

В четвертой строке — `ftp://` (File Transfer Protocol).

Если у вас возникают проблемы с чтением и записью в файл при использовании PHP как серверного модуля, помните, что применяемые файлы и директории должны быть доступными для серверных процессов. На платформе Windows будьте осторожны, избегайте обратных слешей в путях и используйте слеши в следующем направлении:

```
$filepointer = fopen("c:\\dir1\\ dir2\\test.txt", "r" );
```

После того как вы открыли файл, произвели необходимые действия над его содержимым, нужно закрыть файловый указатель. Это показывает интерпретатору, что все действия над содержимым этого файла будут прекращены. Просто запомните, что если вы производите вызов функции, открытие файла и установление файлового указателя, ниже должна быть приведена функция, которая будет закрывать этот файл с файловым указателем. Для функции fopen() используется функция fclose(). Она имеет следующий синтаксис:

```
int fclose(int fp)
```

На месте параметра fp записывается переменная файлового указателя. В вышеописанном примере переменной файлового указателя является $f ilepointer.

При удачном выполнении программы, т. е. при правильном задании параметров функции fclose(), она возвратит значение true, в противном случае — false. Обратите внимание: указатель, используемый в функции f close(), должен быть действующим и указывать на файл, открытый функцией fopen(). Рассмотрим пример:

```
<?php
$fp = fopen ("Z:\\home\\localhost\\newname.txt", "r");
$str = fgets ($fp, 5) ;
// на этом месте применяется код для работы с содержимым файла
If (fclose($fp))
{
echo"Закрытие файла newname.txt осуществлено успешно<br>";
}
else
{
echo"Функция выполнила ошибку<br>";
}
?>
```

Пример очень простой, открывает и устанавливает указатель файла, а после этого производит закрытие. Условный оператор проверяет файл на правильность выполнения операции. Пример не содержит каких-либо ошибок, поэтому результатом выполнения этой программы будет строка:

```
Закрытие файла newname.txt осуществлено успешно
```

Основные тонкости работы этой функции вы освоите при самостоятельном экспериментировании. На начальном этапе главное понять, как правильно производить работу с функцией fclose().

Чтобы открыть файл, в PHP существует далеко не одна функция. Мы рассмотрели наиболее распространенные и эффективные — fopen() и fclose(). Теперь же рассмотрим функции, которые по принципу работы аналогичны, но имеют кое-какие тонкости при вызове.

Функция popen() производит открытие файла с установлением указателя. Она не получила такого широкого распространения, как fopen(), что обусловлено способом работы этой функции. Синтаксис функции popen():

```
int popen(string command, string mode)
```

Функция возвращает файловый указатель, идентичный возвращаемому fopen(), но этот указатель мононаправленный (может использоваться только для чтения или только для записи) и должен быть закрыт pclose(). Этот указатель можно использовать с fgets(), fgetss() (см. ниже).

Второй параметр mode полностью аналогичен mode в файле fopen().

При правильной работе функция возвращает значение, равное true, в противном случае — false. Например:

```
$filepointer = popen("/bin/ls", "r");
```

После открытия файла функцией popen() можно производить операции, связанные только с чтением или записью файла. Проделав все необходимые операции, нужно закрыть существующий файл. Если в начале вы воспользовались функцией popen(), чтобы закрыть файл, над которым производились операции, необходимо вызвать функцию pclose(). Функция fclose() в этом случае вызовет ошибку. Синтаксис функции pclose():

```
int pclose(int fp)
```

Функция pclose() закрывает файловый указатель к каналу, открытому popen(). Файловый указатель должен быть действующим и возвращенным успешным вызовом popen(). В качестве параметра fp нужно указать переменную файлового указателя, на который ссылается функция popen(). Функции popen() и pclose() также не могут существовать по отдельности, как и fopen() и fclose().

Функция pclose() возвращает true при отсутствии каких-либо ошибок, в противном случае — false. Рассмотрим пример:

```
<?php
$filepointer = popen("/bin/ls", "r");
// тут может быть ваш код
pclose ($filepointer);
?>
```

Между функциями popen() и pclose() выполняются необходимые операции работы с содержимым файла.

Перейдем непосредственно к изучению процессов, связанных с работой содержимого файла и указателя.

Чтобы проверить, находится ли указатель в конце файла, необходимо воспользоваться функцией feof(). Она очень полезна при последовательном считывании строк файла, при этом эта функция позволяет определить, когда достигается конец файла. Функция имеет очень простой и понятный синтаксис:

```
int feof(int fp)
```

Функция возвращает true, если указатель файла равен EOF (end of file — конец файла) или в случае ошибки; в противном случае возвращается false. Указатель должен быть действующим и указывать на файл, успешно открытый fopen() или popen(). Как правило, эта функция применяется в циклах для создания условия выполнения. Если указатель достиг конца файла, необходимо прервать выполнение цикла, иначе это может вызвать ошибку. Например:

```
<?php
if ($fp = fopen ("Z:\\home\\localhost\\www\\newname.txt", "r"))
echo"Работа функции fopen() произведена успешно!<br>";
do
{
$str = fgets ($fp, 3) ;
echo $str," <br>";
}
while (!feof($fp)) ;
if (!fclose($fp))
{
echo "Функция f close () выполнила ошибку<br>";
}
else
{
echo"Закрытие файла newname.txt осуществлено успешно<br>";
}
?>
```

В данном примере использована функция fgets() возврата указанного количества символов из файла, расположенного по адресу Z:\\home\\localhost\\www\\newname.txt.

После того как указатель достиг конца файла, мы прерываем выполнение цикла и производим закрытие файла функцией fclose(). Результат действия данной программы:

```
Работа функции fopen() произведена успешно!
Уд
ач
но
е
вы
по
лн
ен
" ие
п
ро
гр
ам
мы
!
Закрытие файла newname.txt осуществлено успешно
```

Как видно из результата, файл newname.txt содержал следующую фразу. «Удачное выполнение программы!». Как только указатель достиг конца файла, выполнение цикла было прервано.

В рассмотренном примере была использована функция fgets(). Это очень распространенная функция, созданная для того, чтобы получать необходимое количество символов, начиная с того места, в котором расположен файловый указатель. При задании количества символов происходит перемещение этого файлового указателя, и он остается в том месте, в котором заканчивается заданное в функции fgets количество символов. Синтаксис функции fgets():

```
string fgets(int fp, int length)
```

Функция fgets () возвращает строку длинной, равную length. Параметр length принимает значения, например 2, 3,...26, т. е. численные значения, в зависимости оттого, как много символов вам необходимо получить. Параметр length читает я по одному байту из файла, заданного в параметре файлового указателя fp. Чтение заканчивается, если обработано указанное количество (length) символов — или

до символов перевода строки и возврата каретки, или до EOF. Один байт прочитается в любом случае. В случае возникновения ошибки возвращается false.

Указатель должен быть действующим и указывать на файл, успешно открытый fopen() или popen().

Рассмотрим пример:

```
<?php
$fp = fopen ("Z:\\home\\localhost\\www\\newname.txt", "r");
while(!feof($fp)) {
$buf = fgets($fp, 3000);
echo $buf;
}
fclose($fp)
?>
```

Данная программа произведет последовательно вызов 3000 символов до тех пор, пока указатель не достигнет конца файла. Результат примера не будем приводить, так как не имеет никакого смысла просто переписывать все содержимое файла. Просто проделайте эту операцию самостоятельно для какого-нибудь файла.

Когда нужно получить строки из какой-то Web-страницы и при этом HTML-теги должны быть удалены, то не надо придумывать какие-либо новые операции по удалению этих зловредных тегов, можно просто воспользоваться функцией fgetss(). Она получила распространение при работе с HTML-страницами. Функция fgetss () позволяет производить удаление всех тегов в строке, длину которой вы указали. По характеру работы функция аналогична fgets(). Синтаксис функции fgetss():

```
string fgetss (int fp, int length[, string allowable_tags])
```

Функция также возвращает строку символов, отличие же ее от fgets() состоит в том, что fdetss() позволяет удалить HTML и PHP-теги из текста. Помимо изученных двух параметров (fru length) данная функция располагает также третьим — string allowable_tags. Этот параметр является необязательным и применяется при необходимости. Параметр allowable_tags указывается в качестве тега или тегов, удаление которых не требуется. Если вы указали, допустим, тег
 в качестве третьего параметра, то интерпретатор произведет удаление всех тегов HTML, содержащихся в полученной строке, кроме тега
.

Для примера создадим файл newname.txt, содержащий следующий текст:

```
<html><body><h3>Добро пожаловать <br> в новый виртуальный мир
</h3X/body></html>
```

Обратите внимание, что текст содержит теги HTML. После этого напишем следующий скрипт:

```
<?php
$fp = fopen ("Z:\\home\\localhost\\www\\newname.txt", "r");
while (!feof ($fp))
{
$buf = fgetss($fp, 50);
echo $buf;
}
fclose($fp)
?>
```

При выполнении этого скрипта все теги будут удалены, и результат программы будет выглядеть следующим образом:

```
Добро пожаловать в новый виртуальный мир
```

Обычное предложение без каких-либо разделений, строковых переносов и шрифтовых выделений. Теперь применим к этому файлу немного другой скрипт, в котором у функции fgetss () будет присутствовать третий параметр.

```
<?php
$fp = fopen ("Z:\\home\\localhost\\www\\newname.txt", "r");

while (!feof ($fp))
{
$buf2 = fgetss($fp, 50, "<br>");
echo $buf2;
}
fclose ($fp)
?>
```

В примере в качестве третьего параметра указан тег
, это значит что скрипт удалит все теги в строке, кроме
. В этом можно убедиться, посмотрев на результат работы скрипта:

```
Добро пожаловать
в новый виртуальный мир
```

Таким образом, начальная строка разбита на две.

Третий параметр функции fgetss () allowable_tags был добавлен в PHP 3.0.13.

В PHP существуют функции бинарного чтения файлов. Чтобы прочесть байты из файла, на который ссылается файловый указатель, используют функцию fread().

Она имеет следующий синтаксис:

```
string fread(int fp, int length)
```

Функция fread () читает байты из файла, на который ссылается файловый указатель fp, до заданной длины (length). Чтение заканчивается, когда прочитано такое количество байтов, которое было указано в параметре length, или достигнут EOF. Например:

```
$path = "/home/localhost/www/newname.txt";
$fp = fopen($path, "r");
$contents = fread($fp, filesize ($path));
fclose ($fp);
```

Функция возвращает строку со всем содержимым файла. В примере использована функция filesize (), которая позволяет получить размер файла (см. п. 26. 4).

ВНИМАНИЕ

В системах, которые имеют различия между двоичным файлом и текстовым (например, Windows), файл должен быть открыт при помощи включенного параметра 'b' в функцию fopen (). Например:

```
$path = "c:\\files\\newpic.gif";
$fp = fopen($path, "rb");
$contents = fread($fp, filesize ($path));
fclose($fp);
```

Все описанные функции используются в PHP 3 и PHP 4.

26.4. Чтение и проверка файлов

Иногда бывает просто необходимо получить параметры файла. Для чего это нужно? Например, пользователь вашей Web-странички захочет записать какой-либо файл. Если его модем не поддерживает передачу данных на быстром уровне, его всегда будет волновать размер файла. Также бывает нужно получить такую информацию, как время создания, либо просто проверить, существует ли указанный файл по тому или иному адресу.

Для этих и других аналогичных операций применяют функции:

- file(),
- readfile(),
- file_exists(),
- fileatime(),
- filectime(),
- filemtime(),
- filetype(),
- filesize().

Чтобы вызывать все строки содержимого указанного файла в массив, используют функцию file(). Она позволяет получить массив строк файла. Ее синтаксис:

```
array file(string filename [, int use_include_path]
```

Каждый элемент возвращенного массива соответствует строке файла (вместе с символом возврата строки). Параметр filename указывает путь и имя файла.

Существует также третий параметр, который является необязательным, в случае его использования вам необходимо выставить значение «1», в этом случае произойдет исследование файла в include_path в файле конфигурации php.ini. Например:

```
<?php
// получение массива строк Web-страницы и его вывод
$plot = file( 'http://www.your_domain_name.com');
while (list ($line_num, $line) = each ($plot))
{
echo "<b>Line $line_num:</b> ". htmlspecialchars ($line). "<br>\n";
}
?>
```

В этом примере произойдет вывод Web-страницы на экран браузера. Сначала каждая строка указанного файла будет помещена в массив $plot. Когда массив будет заполнен, то ввод строк файла прекратится и программа приступит к выводу содержимого этого массива в таком же порядке, как и вводила.

Если нужно вывести Web-страницу как строку, можно сделать так:

```
$fcontents = join(' ', file ('http://www.nessery_domain_name.com'));
```

В этом случае произойдет возврат строки, включающей все содержимое Web-страницы. Вывести на печать эту строку не составит никакого труда. Вы можете сами выбрать, через массив или же напрямую через строку выводить содержимое существующего файла. Оба способа хороши, и выбор прежде всего будет зависеть непосредственно от решаемой задачи.

Чтобы вывести все содержимое файла, не прибегая к каким-либо операциям, просто воспользуйтесь функцией readfile(). Она может выполнять два действия одновременно. Первое — функция производит вывод всего содержимого файла, т. е. на экране браузера автоматически появится все содержимое файла. Помимо этого данная функция возвратит целое число, равное количеству байтов, выведенных на экран браузера. Синтаксис функции readfile():

```
int readfile(string filename [, int use_include_path])
```

Функция имеет параметры, аналогичные функции file(). Отличие состоит в том, что функция readfile() производит чтение файла и запись его на стандартное устройство вывода. Также функция readfile() возвращает количество прочитанных байтов.

При возникновении ошибки возвращается false и выводится сообщение об ошибке, за исключением функции, вызванной как @readfile.

Если параметр filename начинается с http:// (без учета регистра), открывается соединение HTTP 1.0 к указанному серверу и текст ответа выводится на стандартное устройство вывода. Поскольку редиректы HTTP не обрабатываются, нужно включать в указание директории завершающие слеши.

Если параметр filename начинается с ftp:// (без учета регистра), открывается ftp-соединение с указанным сервером и файл ответа отображается на стандартном устройстве вывода.

Если сервер не поддерживает режим пассивного ftp, то вызов завершится ошибкой.

Если filename начинается как-нибудь иначе, будет открыт файл файловой системы и его содержимое выведется на стандартное устройство.

Приведем пример:

```
<?php
$bites = readfile("Z:\\home\\localhost\\www\\newname.txt");
echo "<br>Количество прочитанных байтов, содержащихся в файле,
равно $bites";
?>
```

С учетом ранее созданного файла newname.txt пример выведет следующий результат:

```
Добро пожаловать в новый виртуальный мир
Количество прочитанных байтов, содержащихся в файле, равно 40
```

Теперь, получив это значение, вы без проблем можете производить работу над ним, изменяя, как вам необходимо.

СОВЕТ

Чтобы производить доступ к файлу, не требуется указатель, просто вызывайте функцию readfile() и считайте, что все содержимое указанного файла уже выведено на экран браузера.

Прежде чем производить возврат каких бы то ни было параметров файла, необходимо убедиться в наличии этого файла. Чтобы определить, существует ли файл по указанному пути, была создана функция file_exists(). Ее синтаксис:

```
bool file_exists(string filename)
```

Функция возвращает значение, равное true, если указанный файл существует, в противном случае — false. На месте параметра filename указывается путь к проверяемому файлу. Например:

```
<?php
if (!file_exists("Z:\\home\\localhost\\www\\newname.txt"))
echo "Ошибка! Указанный файл по этому пути не существует";
else
echo "Указанный файл существует!";

?>
```

Указанный файл существует, следовательно, результатом выполнения этого скрипта будет строка:

```
Указанный файл существует!
```

ВНИМАНИЕ

Функция file_exists не производит проверку файлов на удаленном Web-сервере. Для правильной проверки файла он должен существовать на том сервере, на котором происходит выполнение скрипта, т. е. файл со скриптом должен быть на том же сервере, что и проверяемый файл.

Иногда помимо того, чтобы получить содержимое файла, нужно иметь доступ к его характеристикам.

Под характеристиками понимается то, что способно описывать файл, выделить из ряда подобных, определить признаки, по которым можно произвести сортировку. Прежде всего это время доступа, время изменения, тип, размер и др.

Чтобы получить время последнего обращения к файлу, используют функцию fileatime(). Ее синтаксис:

```
int fileatime(string filename)
```

Функция возвращает значение времени последнего обращения к файлу, в случае какой-либо ошибки выдает false. Возвращаемое время выглядит, как в Unix timestamp. Результат этой функции записывается в хэш.

Функция применяется только к файлам, размещенным непосредственно на самом сервере, как и работающий скрипт, содержащий эту функцию. Функция не распространяет свою работу на удаленные файлы.

ВНИМАНИЕ

Время последнего обращения к файлу (atime) изменяется всякий раз, когда происходит чтение содержимого файла. Регулярное обращение к очень большому количеству файлов и каталогов может быть очень дорогостоящим, так как Интернет пока не бесплатный. На некоторых файловых системах Unix модификации последнего времени обращения (atime) могут быть отключены. Это сделано для того, чтобы увеличить рабочие характеристики и быстродействие таких приложений, как, например, USENET. На таких файловых системах функция fileatime () не будет работать.

Чтобы получить время последнего изменения файла, в PHP существует функция filectime (). По принципу работы и способу задания параметров эта функция аналогична fileatime (). Ее синтаксис:

```
int filectime(string filename)
```

Функция возвращает значение времени последнего изменения файла в форме Unix timestamp. В случае возникновения какой-либо ошибки функция возвращает false. Filectime() используется только для операционных систем Unix.

ВНИМАНИЕ

В большинстве файловых систем Unix файл является измененным, когда его inode-данные изменены, т. е. тогда, когда разрешение, владелец, группа или другие мета-данные от inode-файла изменены.

В некоторых документациях по Unix упоминается, что время последнего изменения файла (ctime) имеет значение как время создания файла. Это неверно.

Для получения времени модификации файла в PHP для всех операционных систем, кроме Unix, существует функция filemtime(). Ее синтаксис:

```
int filemtime(string filename)
```

Функция возвращает время, когда данные блоков файла были изменены или записаны, т. е. время, когда изменялось содержимое файла.

Приведем пример:

```
fileatime("Z:\\home\\localhost\\www\\newname.txt");
filectime("Z:\\home\\localhost\\www\\newname.txt");
filemtime("Z:\\home\\localhost\\www\\newname.txt");
```

Все значения, возвращаемые этими функциями, будут отличными друг от друга.

Чтобы получить тип файла, необходимо воспользоваться функцией `filetype()`. Ее синтаксис:

```
string filetype(string filename)
```

Функция `filetype()` позволяет возвращать тип файла, т. е. она будет возвращать одно из значение: fifo, char, dir, block, link, `file` и unknown.

В случае возникновения какой-либо ошибки функция вызовет значение false.

Результат функции сохраняется в хэше.

Так же, как и описанные функции получения времени, функция `filetype()` не распространяется на удаленные файлы (remote files).

Функция `filesize()` определяет размер файла. Ее синтаксис:

```
int filesize(string filename)
```

Функция `filesize()` возвращает целое число, равное количеству байтов, содержащихся в файле.

Описанные функции используются в PHP 3 и PHP 4.

26.5. Определение атрибутов файлов

Атрибут — признак или свойство, характеризующее объект.

Каждый файл имеет свой набор атрибутов (исполнимость, читаемость, изменяемость и т. д.). В данном параграфе рассмотрены следующие функции получения атрибутов:

- `is_dir()`,
- `isexecutable()`,
- `is_file()`,
- `is_link()`,
- `is_readable()`,
- `is_writeable()`.

Эти функции практически одинаковы по принципу работы и синтаксису, поэтому сначала опишем общие принципы работы этих функции, а затем расскажем, для чего каждая из функций применяется.

Обратите внимание, что все возвращаемые значения этих строк сохраняются в хэше.

При возникновении какой-либо ошибки каждая их этих функции вернет значение, равное false.

Все функции, кроме is_file(), не производят работу с удаленными файлами.

ВНИМАНИЕ

Функция is_link() не работает в операционной системе Windows.

При работе с функцией is_readable() и is_writeable() помните, что PHP может иметь доступ к файлу как пользователь, под чьим id запущен Web-сервер (часто 'nobody').

Описываемые функции подчиняются следующему синтаксису:

```
bool is_dir(string filename)
```

Для примера использована функция is_dir(), хотя на ее месте может быть любая другая из приведенного списка функций. Все функции возвращают булевый тип (true или false), в качестве параметра filename применяется путь к исследуемому файлу.

Теперь опишем каждую функцию в отдельности:

Is_dir() возвращает true, если заданный путь указывает на каталог, т. е. если параметр filename является каталогом, функция is_dir() возвратит true, в противном случае — false. Например:

```
is_dir("Z:/home/localhost/www");
```

Is_executable() проверяет, относится ли файл к классу исполнимых, если да, то возвращает true, иначе — false.

Is_file() проверяет, относится ли файл к классу обычных файлов. Функция возвращает true, если указанный файл является обычным и существует. В противном случае — false.

Is_link() проверяет, относится ли файл к символическим ссылкам. Если указанный файл является символической ссылкой, функция возвратит true, в противном случае — false.

Is_readable() проверяет, относится ли файл к классу читаемых. Если файл существует и является доступным для чтения, функция возвращает значение true, в противном случае — false.

Is_writeable() проверяет, относится ли файл к классу записываемых. Функция возвратит true, если указанный файл в этой функции существует и доступен для записи, в противном случае — false.

Описанные здесь функции используются в PHP 3 и PHP 4.

В качестве исключения необходимо выделить функцию is_writable(), которая работает в PHP 4—PHP4.0b2 и выше.

26.6. Создание и удаление директории

Директории (или папки, каталоги) играют важную роль в расположении файлов на жестком диске вашего компьютера. Если бы не было директорий, то люди все равно придумали бы что-нибудь, что упростило бы систему расположения файлов на жестком диске. Поэтому очень часто помимо операций работы с файлами приходится производить кое-какие операции с директориями. Язык PHP позволяет работать не только с файлами, но и с директориями, в которых они хранятся. Очень часто возникает необходимость просто создать или удалить ту или иную директорию. Для этого существуют две важные и широко распространенные в PHP функции:

- mkdir(),
- rmdir().

Чтобы создать директорию с необходимым именем, используют функцию mkdir(), которая создает директорию по указанному адресу. Ее синтаксис:

```
int mkdir(string pathname, int mode)
```

Вместо параметра pathname указывается путь и имя директории. Если нужно указать параметр mode в восьмеричной системе, то число должно начинаться с 0. Параметр mode указывается обязательно.

В ходе выполнения функция возвращает true, если создание директории прошло успешно, в противном случае — false.

Рассмотрим пример:

```
<?php
if (mkdir("Z:/home/localhost/www/4", 0700»
echo"Директория по имени 4 создана успешно";
else
echo"Ошибка программы";
?>
```

Если вы не укажете какой-либо параметр программы, функция работать не будет и вернет false. При выполнении программы в каталоге www у нас не было директории по имени 4. После выполнения скрипта она появилась.

ВНИМАНИЕ

Когда программа уже хоть раз создала каталог (например, в нашем случае программа создала каталог по имени 4), то при повторном вызове этого же скрипта функция вызовет ошибку. Прежде чем создавать каталог по неизвестному для вас пути, убедитесь в том, что имя нового каталога будет отличным от всех имеющихся.

Для удаления папок в PHP существует функция rmdir(). Ее синтаксис:

```
int rmdir(string dirname);
```

Вместо параметра dirname необходимо задавать путь к удаляемому каталогу. Обратите внимание, что при удалении указанного в параметрах каталога необходимо, чтобы он был пустым. При возникновении ошибки функция возвращает значение, равное 0. Функция также может применяться совместно с оператором условия.

Рассмотрим пример:

```
<?php
if (!rmdir("Z:/home/localhost/www/4"))
echo"Ошибка программы";
else
echo"Каталог успешно удален";
?>
```

При выполнении этого скрипта созданная нами папка будет удалена и выведена фраза: «Каталог успешно удален».

Описанные функции используются PHP 3 и PHP 4.

26.7. Доступ к строке файлового пути

Иногда в процессе работы нужно обратиться к определенной части пути файла, чтобы произвести изменения, например, не всего пути, а именно какого-либо отдельного каталога. Изученные выше функции позволяют получать путь, но этот путь нельзя разделить. В этом параграфе мы рассмотрим функции, которые осуществляют более серьезные операции работы с путем к тому или иному файлу:

- pathinfo(),
- realpath().

Для получения информации о файловом имени используют функцию pathinfo(). Она возвращает массив значений. Ее синтаксис:

```
array pathinfo(string path)
```

В качестве параметра path задается полный путь к файлу. После этого функция передает значения этого пути в массив, при этом соответствующим образом разбивая строку пути. Необходимо заметить, что функция возвращает не просто массив, а ассоциативный массив. Происходит разделение пути на три части, которые заносятся в массив по отдельности. Чтобы вызвать каждую отдельную часть, необходимо воспользоваться параметрами ассоциативного массива: dirname (директория), basename (имя файла) и extension (расширение).

Рассмотрим пример:

```
<?php
$path = pathinfo("Z:\\home\\localhost\\www\\newname.txt");
echo $path["dirname"]. "\n <br>";
echo $path["basename"]. "\n<br>";
echo $path["extension"]. "\n<br>";
?>
```

Скрипт выведет следующий результат:

```
Z:\home\localhost\www
newname.txt
txt
```

Как видно из результата, в первом случае вывели директорию, во втором — имя файла, в третьем — расширение.

Функция `infopath()` работает в PHP 4–4.0.3.

Функция realpath () возвращает модифицированную строку пути. Ее синтаксис:

```
string realpath (string path)
```

Вместо параметра path указывается путь. Функция realpath () удаляет все символические связи и разрешенные ссылки вида `/./`, `/../` и дополнительные / особенности во входном пути и возвращает полное составное имя. Заканчивающийся путь не будет иметь никакой символической связи `/./` или `/../` компонентов. Рассмотрим пример:

```
<?php
$path = realpath("/../../newname.txt");
echo $path. "\n<br>";
?>
```

Результатом выполнения скрипта будет следующая строка:

```
z:\newname.txt
```

Произошло удаление всех компонентов `/../`, обратите внимание, что функция также произвела замену наклона слеша.

Функция работает PHP 4–4.0b4.

26.8. Получение информации о файле

Иногда нужно получить полную информацию о файле или символической ссылке, при этом затратив как можно меньше усилий и времени. Для решения этой задачи в PHP существует две функции:

- `stat()`,
- `lstat()`.

Функция `stat()` позволяет получить массив всех имеющихся параметров указанного файла. Ее синтаксис:

```
array stat (string filename)
```

Вместо параметра `filename` указывается путь к файлу. Функция `stat()` собирает статистику о файле, указанном в функции. Возвращает массив статистической информации о файле со следующими элементами:

- устройство(device);
- узел(inode);
- режим защиты inode (inode protectionmode);
- номер ссылки (number oflinks);
- ID пользователя или владельца (user ID ofowner);
- ID группы владельца (group IDowner);
- тип устройства, если устройство inode* (device type ifinode device);
- размер в байтах (size inbytes);
- время последнего доступа (time oflastaccess);
- время последней модификации (time oflastmodification);
- время последнего обмена (time oflastchange);
- размер блока для I/O файловой системы* (blocksize forfilesystem I/O);
- количество занятыхблоков (number ofblocksallocated).

Для примера произведем исследования файла `newname.txt`:

```
<?php
$plot = stat("Z:\\home\\localhost\\www\\newname.txt");
while (list ($line_num, $line) = each ($plot))
{
echo "<b>Line $line_num:</b> ". htmlspecialchars ($line).
"<br>\n";
}
?>
```

* Только для систем, поддерживающих тип st_blksize. В других системах (например, Windows) возвращается значение, равное 1.

После того как произойдет выполнение этого примера, на экране браузера выведется массив значений:

```
Line 0: 25
Line 1: O
Line 2: 33206
Line 3: 1
Line 4: O
Line 5: O
Line 6: 25
Line 7: 40
Line 8: 1010696400
Line 9: 1010676818
Line 10: 1010673781
Line 11: -1
Line 12: -1
```

Функция работает в PHP 3 и PHP 4.

Функция `lstat()` является аналогией функции `stat()`, она также возвращает массив значений параметров указанного файла. Единственное отличие `stat()` от функции `lstat()` заключается в том, что последняя работает также и с символьными ссылками. Рассмотрим синтаксис функции:

```
array lstat(string filename)
```

Функция `lstat()` собирает статистику файла или символьной ссылки, названной именем файла. Если параметр `filename` является символьной ссылкой, то будет возвращено состояние символьной ссылки, но ни в коем случае не состояние файла, указанного символьной ссылкой.

Функция `lstat()` используется в PHP 3–3.0.4 и PHP 4.

26.9. Создание уникального имени

Допустим, что в ходе написания программы появилась необходимость создания временного уникального файла. PHP позволяет это сделать. Операция создания уникальных файлов осуществляется при помощи следующих функций:

- `tempnam()`,
- `tmpfile()`.

Для создания уникального имени файла применяется функция tempnam(). Ее синтаксис:

```
string tempnam(string dir, string prefix)
```

Функция tempnam () создает уникальное имя файла в указанной директории. Если директория йе существует, tempnam() может генерировать имя файла во временной директории системы.

Функция возвращает новое временное имя файла, или нулевую строку при ошибке.

Поведение функции tempnam () прежде всего зависит от операционной системы. На системе Windows системная переменная TMP будет регулироваться параметром dir, на Linux системная переменная TMPDIR имеет преимущество, в то время как SVR4 будет всегда использовать параметр, указанный в dir, конечно, при условии, что каталог, на который указывает параметр dir, существует. При возникновении каких-либо сомнений ознакомьтесь с документацией вашей системы.

Приведем пример:

```
$tmpfname = tempname("/tmp", "FOO");
```

ВНИМАНИЕ

Поведение этой функции было изменено в версии PHP 4.O.3. Временный файл создается, чтобы можно было избежать «условия быстрого движения», при котором файл могбы появляться в файловой системе при генерировании строки и раньше возвращения скрипта к созданию файла.

Функция работает в PHP 3 и PHP 4.

Чтобы просто создать временный файл, необходимо обратиться к функции tmpfile(). Она имеет следующий синтаксис:

```
int tmpfile(void)
```

Функция tmpfile () позволяет создавать временный файл с уникальным названием в режиме записи. Файл производит работу после того, как была вызвана функция fopen(). Файл автоматически удаляется, как только происходит вызов функции fclose () либо завершение выполнения скрипта.

Функция tmpfile () используется в PHP 3–3.0.13 и PHP 4–4.0b4.

26.10. Установка времени модификации файла

Существует возможность программного изменения времени модификации файла. Вам не потребуется производить какие-либо операции непосредственно над самим содержимым указанного файла, просто примените функцию touch(), и время с учетом правильной установки параметров будет изменено на то, которое действительно необходимо.

Для чего это используется? Например, вы написали скрипт, который проверяет лог-файл на изменения или же просто периодически заносит новые сведения. Обращение к данному файлу осуществляется посредством изменяемого времени, т. е. возникает необходимость, чтобы каждый день происходило изменение времени модификации, иначе может возникнуть какая-либо ошибка. Когда произвести модификацию физически невозможно из-за некоторых существенных факторов, появляется необходимость проделать это программно. В таком случае используют функцию установления времени модификации. Ее синтаксис:

```
int touch(string filename[, int time])
```

Вместо параметра filename указывается имя файла, время модификации которого будет изменяться. Параметр time отвечает непосредственно за установленное время, т. е. это время, на которое будет изменяться существующее. Параметр time может отсутствовать, в этом случае устанавливается текущее время.

В случае отсутствия указанного файла функция просто создаст новый и присвоит ему установленное время модификации.

Функция touch() возвращает true при успешном выполнении, при возникновении какой-либо ошибки функция возвратит false.

Рассмотрим пример:

```
<?php
if (!touch ($path))
{
print "Ошибка, невозможно изменить время модификации файла".
basename($path);
}
else
{
print "Время модификации файла ". basename($path)." было изменено
<br>\n";
}
?>
```

В процессе выполнения приведенного примера дата и время модификации файла будут изменены на текущее, так как в нашем случае не указано время, на которое необходимо изменять. Проверить правильность изменения времени можно при помощи функции filemtime(). В результате работы данного скрипта функция touch(), приведенная в операторе условия, будет возвращать значение true, что в свою очередь приведет к выполнению следующего блока программы. В итоге на экран браузера будет выведено:

```
Время модификации файла newname.txt было изменено на
```

Обратите внимание: теперь, когда мы знаем принцип работы функции basename () (см. п. 26.1), мы без проблем можем ее применять для вывода только файла, без указания конкретного пути до этого файла.

Функция модификации времени файла touch () работает в PHP 3 и PHP 4.

26.11. Разные функции

Изучим следующие функции:

- chmod(),
- fputs(),
- rewind(),
- ftell().

Если нужно изменить режим файла, используют функцию chmod (), которая позволяет менять режим на необходимый в данный момент. Ее синтаксис:.

```
int chmod(string filename, int mode)
```

Изменяет режим файла, указанного в параметре filename, на режим, указанный параметром mode. Заметим, что mode не присваивает автоматически восьмеричное значение, для этого необходимо воспользоваться префиксом с нулем. Например:

```
chmod("/dir1/dir2/anyfile", 755); // десятичный, неверный режим
chmod("/dir1/dir2/anyfile", "u+rwx,go+rx"); // строка, неверный режим
chmod("/dir1/dir2/anyfile", 0755); // восьмеричный, верный режим
```

При правильном выполнении функция возвращает значение true, в противном случае — false.

ВНИМАНИЕ

Параметр mode не устанавливается автоматически в качестве восьмеричного значения, поэтому функция, содержащая строку параметров (такую, как "g+w") не будет осуществлять работу должным образом.

Функция работает в PHP 3 и PHP 4.

Помимо того, что в PHP можно получать информацию из файла, можно также ее туда вносить. Если вам необходимо внести какую-либо строку в файл, соответствующий заданному указателю, используйте функцию fputs (), которая позволяет вносить любую строку символов в указанный файл. Содержимое этого файла может быть либо стерто, либо дополнено нужной строкой. Все это зависит от режима, определяемого функцией fopen() при определении указателя. Рассмотрим синтаксис функции fputs ():

```
int fputs(int fp, string str[, int length])
```

На месте параметра str указывается строка, вносимая в файл, fp — файловый указатель, определяется с учетом функции `fopen()`. Параметр length отвечает за длину символов, которые можно вносить в указанный файл. Если ваша строка состоит из 10 символов, а length — 5, то в ваш файл с учетом размещения и, например, указателя будет внесено только 5 первых символов, остальные будут обрезаны. Параметр length опционален, т. е. при его отсутствии записывается вся строка str.

Рассмотрим пример работы функции `fputs()`:

```
<?php
$fp = fopen("Z:\\home\\localhost\\www\\newname.txt", "w");
fputs($fp, "Работа над ошибками закончена!", 15);
fclose($fp);
readfile("Z:\\home\\localhost\\www\\newname.txt");
?>
```

Пример показывает, как работает функция при указании длины вносимой в файл `newname.txt` строки. Из всей приведенной строки в файл `newname.txt` будет внесено только 15 символов. На экране браузера будет выведено:

```
Работа над ошиб
```

Ровно 15 символов с учетом пробелов.

Теперь рассмотрим пример работы функции fputs () без указания параметра length:

```
<?php
$fp = fopen("Z:\\home\\localhost\\www\\newname.txt", "w");
fputs($fp, "Работа над ошибками закончена!");
fclose($fp);
readfile("Z:\\home\\localhost\\www\\newname.txt");
?>
```

В данном случае в окно браузера будет выведено:

```
Работа над ошибками закончена!
```

Вся строка передана в указанный файл. Иногда нужно использовать функцию fputs () без указания параметра length. Все зависит от того, что именно надо получить в результате работы функции fputs ().

Функция fputs () используется в PHP 3 и PHP4.

При работе с содержимым файла происходит перемещение файлового указателя. Иногда в процессе работы возникает необходимость в установлении файлового указателя на начало файла. Для этого используют функцию rewind(). Ее синтаксис:

```
int rewind(int fp)
```

Функция rewind() устанавливает файловый указатель, заданный вместо параметра fp, на начало файла. При возникновении ошибки возвращается 0.

Файловый указатель должен быть действующим и указывать на файл, успешно открытый функцией fopen(). Например:

```
<?php
$fp = fopen("dir/dir1/dir2/yourfile", "w");
rewind($fp);
fclose($fp);
?>
```

Функция ftell() возвращает место файлового указателя, т. е. позицию указателя, установленного при помощи функции fopen(). Синтаксис функции ftell():

```
int ftell(int fp)
```

Функция ftell() возвращает позицию указателя в файле, на который ссылается файловый указатель, и устанавливает его на месте параметра fp. Таким образом, происходит смещение в потоке файла.

При возникновении ошибки функция ftell() возвращает false.

Файловый указатель должен быть действующим и указывать на файл, успешно открытый fopen() или popen().

Описанные функции ftell() и rewind() используются в PHP 3 и PHP 4.

26.12. Пример программирования

Файл можно включить в скрипт функцией include(). Но это не всегда рационально. Например, вы — гениальный дизайнер — нарисовали прекрасный сайт некой фирме, но вам очень не хочется обновлять каждую неделю их прайс-лист. Вы можете за обновление брать деньги, но много денег за это никто не даст, а повозиться придется. Как обычно готовят прайс-лист в фирме? Скорее всего у них есть какая-нибудь таблица в неком табличном редакторе типа Lotus или Excel. Вот ее бы и положить на сайт, но файл с таблицей хитрого формата, да и размера немалого.

Владея PHP, вы можете легко уйти от рутины. Любой табличный редактор позволяет сохранять листы как текст с разделителями табуляции. Рассмотрим, как можно такой файл красиво вывести в виде таблицы со строками разного цвета.

```
<?
$fp = fopen("name_of_file.tsv; 'r' ) ; // открываетсяфайл,
//созданный в программе Exel
if ($fp):
?>
<table border = 0 cellpadding = 2 cellspacing = 0>
<?
for ($i = 0; !feof($fp); $i++) ; // производится чтение строки файла,
//пока не достигается конец, также одновременно
// удаляются пробелы в конце и строки разбиваются на поля
list($a,$b,$c) = split ("\t", chop (f gets ($fp, 1000)));
?>
<tr bgcolor = >>#<?echo $i%2?'f0f0f0'; 'ffffff' ?>
>>>
<td> <?echo $a ?> </td>
<tdx?echo $b ?> </td>
<td> <?echo $c?></td>
</tr>
<?endfor?>
</table>
<?fclose($fp) ; endif ?>
```

PHP имеет достаточное количество встроенных функций для работы с файлами, но часто хочется этот набор несколько расширить, например чтобы сделать чат. В теории все просто: все посетители пишут в один файл и читают последние несколько строк этого файла. Обратите внимание на функцию **tail()**, которая работает очень быстро независимо от размеров читаемого файла. Например:

```
<?
if (!$tail_inc): // чтобы не включать файл дважды
$tail_inc = 1;
function tail($file, $num) { // читаем $num последних строк
                             // файла $file
Global $tail_start_buf; // предполагаемая длина строки
if ($tail_start_buf = = 0) $tail_start_buf = 80;
```

```
$appxlen = $tail_start_buf; //примерная длина строки для расчета
$flen = filesize($file); // размер файла
$out = array(); // то, что вернется
$fp = @fopen($file, 'r');
if ($fp) {
do {
if ($num*$appxlen> $flen) $pos = 0; // вычисляем, откуда читать файл
else $pos =$flen — ($num*$appxlen);
$out = _readfile($fp, $pos, $num); // читаем строки до конца файла
// в следующем цикле длина строки будет равна длине средней
// прочитанной строки, умноженной на 2
$appxlen = ($flen - $pos + 1)*$num*2/count($out); //*!*
}
while (count($out)!=$num&&$pos!=0);
fclose($fp);
}
return $out;
}
// вспомогательная функция
// читает файл $fp с позиции $pos и максимум $num строк
function _readfile($fp, $pos, $num) {
fseek($fp, $pos); // позиционируем файл
$imp = array(); // временный массив
while (!feof($fp)) { // читаем файл до конца
$line = chop (fgetsl($fp)); // *!!*
$tmp [] = $line;
}
$j = count($tmp) — $num; // количество лишних строк
if ($pos!=0 && $j = = 0( { // если равно, сколько надо строк,
// чтобы пропустить первую неполную строку, если не хватает строк,
// выводить все
$j++;
```

```
if ($j<0) {
$j = 0;
$xnum = $num — 1;
} else $xnum = $num -1;
// переписать в выводной массив нужные строки
for ($i =0; $i<$xnum && $j<count($tmp); $i++, $j++)
$out[$i] = $tmp[$j];
return $out;
}
endif; //if (!tail_inc) ?>
```

Это не лучший на свете алгоритм, но все же работоспособный. Есть два замечания по поводу этой программы.

! — здесь вычисляется средняя длина строки для следующего цикла. Можно было написать $appxlen = 2, но так будет быстрее.

!! — здесь используется функция `fgetsl()`. Функция чтения строки из файла имеет ограничение на длину считываемой строки. А думать о длинах строк при написании гениальных скриптов не хочется. Как быть? Воспользуйтесь функцией `fgetsl()`, аналогичной `fgets()`, но без неприятного ограничения:

```
<?
Function fgetsl($fp){
While (!feof($fp)&&strchr($att, "\n"))
$out.= fgets($fp, 1000);
return $out;
}
?>
```

Заключение

Изученные в этой главе функции позволяют выполнять различные необходимые операции с файлами, но, по утверждению многих программистов, этих функций недостаточно для решения сложных задач. Владея описанными в этой главе функциями, вы просто сможете свободно ориентироваться в пространстве файловых функций PHP. При возникновении какой-либо задачи, выполнение которой не подчиняется никакой из описанных функций, создавайте свои собственные и применяйте их для решения таких задач. Файловые функции получили большое распространение на практике, так как фактически все операции работы с текстом тре-

буют записи текстовой информации в файл. Например, чат, форум, гостевая книга — все, что находится там в качестве текстовой информации, должно где-то сохраняться, а именно в файлах, созданных для этого. Поэтому главная задача при программировании основных приложений Web-узла состоит в том, чтобы помимо основных особенностей языка изучить и функции работы с файлами.

Главное, что хотелось бы выделить из этой главы:

- при произведении какой-либо операции доступа к содержимому файла необходимо установить так называемый файловый указатель в соответствии с этим файлом;
- после того, как все операции по работе с содержимым файла были проделаны, не забывайте воспользоваться функцией закрытия файла;
- очень важно указывать непосредственно путь к файлу. В случае неверного указания пути возникают ошибки. Следите при этом за правильным наклоном слеша с учетом вашей операционной системы;
- если вы произвели какую-либо операцию с содержимым файла, необходимо помнить о том, что в этот момент указатель переместился на определенное число символов и никак не может находиться в начале файла. Чтобы он был установлен на начало файла, используют функцию rewind ().

Глава 27
Работа с электронной почтой в PHP

Хотя бы раз каждый из вас посещал Web-ресурс, но не только посещал, а еще и производил какие-либо операции, например заполнение той или иной формы, регистрация в качестве полноправного пользователя, ввод сообщения в форум и т. д. Если у вас не просто обычная доска объявлений, а корпоративный сайт, скорее всего появится необходимость вести контроль за вышеперечисленными действиями. Не будем объяснять, для чего это необходимо, просто предположим, что для аудитории коммерческого Web-ресурса будет неприятно платить за то, что он плохо работает или же, еще хуже, вообще не работает.

Помимо всех задач, связанных с отправкой почты, вы также можете контролировать работоспособность вашего ресурса и в случае каких-либо проблем максимально быстро их решить.

Данная глава покажет вам, как можно при помощи простой функции отправки почты производить описанные выше операции. Основная задача, которая стоит перед нами в этой главе, научить вас рационально применять функции работы с электронной почтой, для этого рассмотрим следующие темы:

- функции отправления почты;
- отправление почты помощи дополнительных заголовков;
- отправление составной электронной почты.

27.1. Функция отправления почты

Чтобы передать данные, полученные при работе на Web-ресурсе, используют функцию с подходящим для этого названием — `mail()`.

Она является очень полезной при работе. Самое главное, на что необходимо обращать внимание — на заполнение всех параметров функции. Это прежде всего поможет определить, откуда получено то или иное сообщение и с чем связано само его отправление. Функция `mail()` имеет следующий синтаксис:

```
bool mail(string to, string subject, string message[, string
additional_headers[, string additional_parameters]])
```

Строка string to указывает, кому вы собираетесь отправлять сообщение. Здесь записывается адрес получателя сообщения, например `your_name@domain_name.ru`. После этого идет строка string subject, равносильная строке «Тема» при отправлении обычного письма, т. е. здесь указывается то, о чем будет вестись речь в вашем письме. На месте string message указывается пересылаемое сообщение, например «С днем Рождения!!!». В четвертом параметре — additional_headers можно указать любой атрибут заголовка письма, например `Content-Type: text/html; charset= windows -1251`.

Функция используется в PHP 3 и PHP 4.

Приведем пример применения функции `mail()`:

```
mail("admin135@gfd.on.ca", "Поздравление", "Поздравляю с днем
рождения!" "Content-type:text/html; charset = windows — 1251");
```

Функция позволяет отправлять сообщение «Поздравляю с днем рождения!», в строке «Тема» будет написано «Поздравление», письмо будет отправлено на адрес `admin135@gfd.on.ca`, при этом содержание будет либо текстовым файлом, либо как HTML. Это для того, чтобы получатель не имел каких-либо проблем с прочтением. Проблемы с кодировкой также должны отсутствовать, так как строкой charset = windows - 1251 мы указали необходимую кодировку для письма.

Как правило, задание значений параметрам осуществляется через переменные, именно это и делает язык очень удобным. Покажем, как можно этот же пример записать в другом виде, при этом результат работы будет аналогичным:

```
$to = "admin135@gfd.on.ca";
$subject = "Поздравление";
$message = "Поздравляю с днем рождения!";
$header = "Content-type:text/html; charset = windows — 1251";
```

```
mail ($to, $subject, $message, $header);
```

После задания значений переменным $to, $subject, $message, $header мы просто вставляем их в функцию mail(). Результат работы этого скрипта будет аналогичным предыдущему. Именно такой способ является наиболее рациональным и удобным при программировании.

Сама функция mail() возвращает булевый тип, что позволяет проверить правильность отправки сообщений. При возникновении какой-либо ошибки функция просто возвращает false, в противном случае — true.

В качестве вносимых параметров существует еще и пятый параметр, который мы рассмотрим далее (см. п. 27.2). Если не указывать четвертый $header или пятый параметры, никакой ошибки это не вызовет. Эти параметры используются на усмотрение самих программистов, чтобы просто подробнее описать сообщение, посланное сервером.

27.2. Отправление почты с использованием дополнительных заголовков

Пятый параметр функции mail() является необязательным, т. е. если он не будет указан, это не приведет к какому-либо неправильному выполнению функции. В синтаксисе функции этот параметр указан как string additional_parameters.

Когда установлен пятый параметр, PHP добавит эти данные к вызову функции mail(). Это очень полезно, когда происходит отправление почты при правильной установке Return_Path header. Например:

```
mail("nobody@any_domain_name.com", "the subject", $message,
"From: webmaster@$NAME\n Reply-To: webmaster@$NAME\nX-Mailer:
PHP/". phpversion());
```

При помощи задания пятого параметра можно устанавливать дополнительные параметры командной строки в почтовой функции. В примере, описанном выше, устанавливается правильный параметр Return_Path header для отправки почты. Функция отправки почты будет воспринимать заголовок X-Authentication-Warning при установленном параметре f, так как пользователь Web-сервера не является лицом, которое находится в списке доверительных персон. Чтобы подавить это предупреждение, вам просто следует добавить пользователя Web-сервера в список лиц, которым вы можете доверять. Добавление пользователей, которым вы можете доверять, производится путем редактирования файла конфигурации отправки почты (sendmail).

Параметр additional_parameters был добавлен в функцию mail() в PHP 4.0.5.

Приведем пример к описанному выше:

```
mail("nobody@your_domain_name.com","thesubject", $message,
"From: webmaster@$SERVER_NAME", "-fwebmaster@$SERVERNAME");
```

27.3. Отправление почты нескольким адресатам

Теперь, когда вы полностью разобрались с основными параметрами функции mail(), вы без труда можете позволить себе отправление любого сообщения, необходимого вам или вашим пользователям. А если у вас возникла необходимость отправить некоторое количество писем, используя одну функцию, можно поступить так:

```
1. $to.= "Sasha <sasha57@edu.ou.com>". ", " ;
2. $to.= "Sveta <sveta34@ edu.ou.com >". ", ";
3. $to.= "ronabop@php.net";
4. $subj = "Напоминание о дне рождения в марте";
5. $message.= "День \t\tМесяц \t\tГод\n";
6. $message.= "12-го \t\tМарт\t\t1982\n";
7. $message.= "2-го\t\tМарт \t\t1983\n";
8. $headers.= "From: Birthday Reminder <birthday@php.net>\n";
9. $headers.= "X-Sender: <birthday@php.net>\n";
10. $headers.= "X-Mailer: PHP\n";
11. $headers.= "X-Priority: 1\n";
12. $headers.= "Return-Path: <birthday@php.net>\n";
13. mail($to, $subj, $message, $headers);
```

Первая—третья строки задают адреса получателей сообщения. Четвертая строка описывает тему сообщения, пятая—седьмая — включают содержание послания. В данном случае это даты дней рождения: день, месяц и год. В случае какой-либо ошибки необходимо будет узнать основные параметры сообщения, чтобы сообщить об этом отправителю, поэтому в восьмой строке указывается электронный адрес отправителя сообщения. В нашем случае это birthday@php.net. Далее по отдельности описываются основные параметры: девятый — отправитель, десятый — x-Mailer, т. е. программа, отправившая письмо, одиннадцатый — задается приоритет сообщения, двенадцатый — адрес, куда отправится уведомление об ошибке, если такая произойдет.

После того как все параметры были заданы, мы без каких-либо проблем включаем их в нашу функцию — тринадцатая строка.

СОВЕТ

Если в переменную, указывающую адрес получателя, внести несколько адресов, разделенных пробелами, информация будет разослана по всем указанным адресам. Это открывает простой путь к организации собственной/ни от кого не зависящей системы почтовых рассылок.

27.4. Принципы программирования

После изучения функций работы с электронной почтой у вас появится множество возможностей вести контроль над операциям, совершаемыми пользователями вашего сайта: когда будет происходить то или иное событие, просто посылайте сообщение себе на почтовый ящик. Это избавит вас прежде всего от бесполезной траты времени за контролем операций, выполняемых пользователями, таких, как регистрация, ввод сообщения в форум, гостевую книгу, ошибка открытия файла, организация обратной связи и т. д. Все эти проблемы контроля вы сможете решить при использовании функции mail ().

Главное, на что следует обращать внимание — переменные, например, сообщения, передаваемые на форум, должны соответствовать переменным, которые будут использованы в качестве сообщения посылаемой функции mail (). Другими словами, необходимо четко следить за соответствием имен переменных форме и скрипте. После того как данные будут переданы в функцию mail (), интерпретатор отправит эти данные по адресу, указанному в функции. Вся остальная дополнительная информация является служебной и никакой смысловой нагрузки не несет (имеется в виду именно этот случай).

Если появилась необходимость вести контроль за посетителями сайта, нужно просто создать программу, которая будет это выполнять. Мы будем получать ту информацию о посетителях, которая автоматически доступна в переменных окружения. Это прежде всего Host, т. е. узел, IP (уникальный номер, приписанный к каждому компьютеру, подключенному к Интернету) и время посещения. Полученную информацию будем просто пересылать к себе на электронный ящик. Этот скрипт является весьма полезным, если количество посетителей вашего сайта не очень большое, в противном случае просто можно заполнить свой ящик до отказа.

Таким образом, вы можете вести контроль, например, посещения не только вашей какой-либо конкретной страницы, но и обращения к какой-либо статье или теме на вашей страничке.

Рассмотрим пример:

```
<?php
if (isset ($HTTP_X_FORWARDED_FOR))
{
$host = gethostbyaddr($HTTP_X_FORWARDED_FOR);
}
else
{
$host = gethostbyaddr ($REMOTE_ADDR);
}
```

```
$InterP=getenv('REMOTE_ADDR');
$time=date('d M Y, H:i:s');
$newhast1=gethostbyaddr("$InterP");
$string=(" Дата - $time, Хост - $host, IP-адрес - $ InterP ");
mail(name@your_domain_name.com, "Контроль посещений", $string,
"From:Удаленный ресурс");
?>
```

После выполнения данного скрипта вы сможете получить полный перечень необходимой вам информации о посещении пользователем вашего ресурса. Таким образом можно производить и другие операции. Например, в случае возникновения ошибки при открытий файла также можно воспользоваться функцией mail (), чтобы сообщить вам об ошибке. Например:

```
$file = "new_card.html";
$FilePointer = @fopen($file, "r");
if ( !$FilePointer )
{
@mail(yourlogin@your_domain_name.com, "Произошла ошибка!",
"На вашем ресурсе произошла ошибка открытия файла $file!");
}
```

В этом случае при возникновении какой-либо ошибки открытия файла new_card. html функция автоматически пошлет сообщение на ваш электронный адрес. Знак @ позволяет обойти сообщение об ошибке, т. е. сама ошибка возникнет, но выполнение программы будет продолжаться далее.

Заключение

В данной главе мы рассказали вам о функции mail (), о ее особенностях и методах применения. Необходимо запомнить: почта будет уходить от имени Web-сервера (как пользователя), не забывайте также про четвертый параметр функции mail — $header, в котором можно указывать любой параметр заголовка письма.

Не обязательно задавать все параметры, но следует указывать, от кого пришло сообщение и его содержание. Используя PHP, можно на своем сайте сделать систему получения и отправки электронной почты. Достаточно лишь изучить протоколы POP и ШАР и открыть серверные соединения с почтовым сервером. Функция fsockopen (хост, порт) открывает сетевое соединение, а пользоваться им можно при помощи обычных файлов функции. Не следует открывать соединение к своему Web-серверу и запрашивать свой же скрипт, так как это может вызывать ошибку и много неприятностей.

Глава 28
Операции потокового ввода-вывода

Из этой главы вы узнаете, как манипулировать с данными буфера. В этом случае вы имеете не одно слово или предложение, а целый буфер значений.

Достигнув мастерства, вы сможете адаптировать изученные здесь операции ввода-вывода к вашим собственным типам данных — классам, чтобы делать это также просто, как ввод-вывод базовых типов. Все это можно воплотить, не используя сложных строк формата и не меняя синтаксис для разных типов данных.

Изучим следующие вопросы:

- освобождение буфера вывода;
- включение буфера, пересылка и очистка;
- возврат значений буфера вывода;
- функция работы с кодированными страницами.

28.1. Освобождение буфера вывода

В чем заключается работа с буфером данных? Прежде всего происходит запись данных в буфер. Затем устанавливается функция, которая помогает понять интерпретатору, что процесс считывания информации в буфер закончен. Осталось теперь при желании вывести или же проделать какую-либо другую работу с данными.

Функция, которая освобождает буфер вывода, называется `flush()`. Ее синтаксис:

```
void flush(void);
```

Освобождает буферы вывода PHP и все остальные, используемые PHP (CGI, Web-сервер и т. д.). Это эффективная возможность выдать все накопленное в буферах в браузер пользователя.

ВНИМАНИЕ

Функция flush () не воздействуют на схему буферзации (отправление данных в буфер) Web-сервера или браузера на стороне клиента.

Некоторые серверы, особенно на Win32, буферизуют выход из скрипта перед выдачей результатов в браузер, до тех пор пока скрипт не закончится.

Браузер может буферизовать ввод перед отображением данных. Netscape, например, буферизует текст, пока не достигнет конца строки или начала отметки.

Функция `flush()` используется в PHP 3 и PHP 4.

28.2. Включение буфера, пересылка и очистка

Как и при любой операции, интерпретатору необходимо показать, откуда он будет производить запись данных вывода в буфер. Помимо этого необходимо указать, в каком месте скрипта прекращается запись данных в буфер. Для решения этих задач и были созданы функции:

- `ob_start()`,
- `ob_end_flush()`,
- `ob_end_clean()`.

Для установления буфера ввода необходимо воспользоваться функцией ob_start (), чтобы показать интерпретатору, что именно с этого места будет происходить запись всех данных, выводимых в буфер. Функция ob_start () подчиняется следующему синтаксису:

```
void ob_start([string output_callback])
```

Эта функция устанавливает буферизацию выхода, т. е. после того, как она задана, все данные выхода будут сохраняться в буфере. Во время того, как буферизация выхода является активной, никакой выход не передается от сценария, вместо этого сохраняются данные во внутреннем буфере.

Содержание этого буфера может быть скопировано (передано) в строковую переменную при помощи функции ob_get_contents (). Чтобы вывести все, что сохранено во внутреннем буфере, используют функцию `ob_end_flush()`. В качестве альтернативы можно применить функцию ob_end_clean (), которая отбросит (очистит) буферное содержание.

Необязательная функция `output_callback` может быть как определена, так и неопределена. Эта функция использует строку как параметр и возвращает также строку. Функция будет вызвана совместно с вызовом функции `ob_end_flush ()` или когда буфер вывода находится в браузере в конце запроса. Когда вызывается функция output_callback, содержание буфера вывода принимается как ее параметр, и в последующем, как и ожидается, возвращается новый буфер вывода, который в результате и будет послан браузеру.

Буферы вывода являются наращиваемыми, это значит, что можно производить вызов функции `ob_start()` повторно, в то время как другая функция ob_start () является активной. Главное при повторных вызовах данной функции — убедиться в том, что функция `ob_end_flush ()` вызвана такое же количество раз, сколько и функция ob_start (). Если многократно вызванные функции являются активными, происходит последовательная фильтрация вывода в соответствии с вложенным порядком.

Пример 28.1. Работа с буфером

```
<?php
function call($buff) {
```

```
return($buff);
}
ob_start("call");
?>
<html>
<body>
<h3> Выведен содержащийся буфер,<br>
так как никаких операций в функции call<br>
с ним не произошло .<br>
</body>
</html>
<?php
ob_end_flush();
?>
```

Функцией ob_start("call"); мы показали интерпретатору, что нам необходимо передать все данные буфера в функцию call. После того как они посланы, программа передает управление функции и эти данные буфера обрабатывает. В нашем случае функция пустая, она пропускает через себя имеющиеся данные буфера и возвращает их без изменения. В конце скрипта вызвана функция ob_end_flush(). Она показывает интерпретатору, что после этой строки ввод данных в буфер прекращается, т. е. все, что вводится дальше, уже не будет помещаться в буфер. После того как функция ob_end_flush() обработала данные буфера, она их возвращает. В нашем случае это сделано при помощи операции return ($buff);. В итоге на экран браузера будет выведено:

```
Выведен содержащийся буфер,
так как никаких операций в функции call
с ним не произошло.
```

Браузер обрабатывает теги, поэтому на экране их не будет — будет видна только выводимая фраза.

Функция ob_end_flush() используется, как правило, совместно с ob_start(). Она позволяет ограничивать ввод данных в буфер, а также пересылать буфер, т. е. она не очищает буфер, а позволяет выводить данные, имеющиеся в буфере. Функция имеет следующий синтаксис:

```
void ob_end_flush(void);
```

Эта функция отправляет содержание буфера вывода (если оно есть в наличии) и помимо этого прерывает операции введения данных в буфер. Для обработки данных буфера используют функцию ob_get_contents(), но только в том случае, пока не вызвана функция ob_end_flush(), поскольку доступ к данным буфера будет закрыт после ее применения. Данная функция была применена в примере 28.1.

Если после закрытия ввода данных в буфер нужно его очистить, используют функцию ob_end_clean(), которая удаляет данные буфера. Функция имеет следующий синтаксис:

```
void ob_end_clean (void) ;
```

Если в примере 28.1 заменить функцию ob_end_flush() на ob_end_clean(), то возвращаемыми данными функции call будет пустая строка, так как функция ob_end_clean() до того как переслать данные буфера, просто произведет его очистку. Именно это и делает различным функции.

ВНИМАНИЕ

Функция ob_end_clean() используется только в версии PHP 4.

28.3. Возврат значений буфера вывода

Иногда возникает необходимость получить доступ к буферу, произвести какие-либо операции с ним или проверить наличие соответствующих данных. Для решения таких задач были созданы две функции:

- ob_get_contents(),
- ob_get_length().

Чтобы получить доступ к данным буфера, применяется функция ob_get_contents(). Она возвращает строку данных буфера, которые непосредственно выводятся в окно браузера. Ее синтаксис:

```
string ob_get_contents(void);
```

Функция возвратит содержимое буфера, если буфер является активным, в противном случае возвратит false. Разберем пример:

```
<?php
echo"Строка, расположенная выше функции ob_start(); <br>";
ob_start();
echo"Строка, расположенная выше функции ob_get_contents(), но
ниже ob_start();<br>";
$content = ob_get_contents();
```

```
echo "***1. Строка, после которой происходит вывод
буфера***<br>";
echo $content;
echo"***1. Вывод данных буфера завершен***<br>";
echo"Строка, расположенная ниже функции ob_get_contents(), но
выше ob_end_flush();<br>";
$content2 = ob_get_contents();
echo "***2. Строка, после которой происходит вывод
буфера***<br>";
echo $content2;
echo"***2. Вывод данных буфера завершен***<br>";
ob_end_flush();
echo"Строка, которая расположена ниже функции ob_end_flush();
<br>";
$content3 = ob_get_contents();
echo "***3. Строка, после которой происходит вывод
буфера***<br>";
echo $content3;
echo"***3. Вывод данных буфера завершен***<br>";
?>
```

Наличие данных буфера при вызове функции ob_get_contents () прежде всего зависит от того, где располагается эта функции вызова. Она должна находиться между функциями ob_start () и ob_get_flush() и будет возможна работа с данными блока. За пределами, ограниченными этими двумя функциями, доступ кданным буфера при помощи функции ob_get_contents () будет невозможен. Более подробно вы сможете понять, ознакомившись с результатом программы:

```
Строка, расположенная выше функции ob_start();
Строка, расположенная выше функции ob_get_contents(), но ниже
ob_start();
***1. Строка, после которой происходит вывод буфера***
Строка, расположенная выше функции ob_get_contents(), но ниже
ob_start();
***1. Вывод данных буфера завершен***
Строка, расположенная ниже функции ob_get_contents(), но выше
ob_end_flush();
```

```
***2. Строка, после которой происходит вывод буфера***
Строка, расположенная выше функции ob_get_contents(), но ниже ob_start();
***1. Строка, после которой происходит вывод буфера***
Строка, расположенная выше функции ob_get_contents(), но ниже ob_start();
***1. Вывод данных буфера завершен***
Строка, расположенная ниже функции ob_get_contents(), но выше ob_end_flush();
***2. Вывод данных буфера завершен***
Строка, которая расположена ниже функции ob_end_flush();
***3. Строка после которой происходит вывод буфера***
***3. Вывод данных буфера завершен***
```

Как видно из результата, в первом случае в буфере вывода содержалась только одна строка. После того как мы переместили функцию ob_get_contents(), доступ к данным буфера расширился за счет выполнения большего числа операций, следовательно, и сам буфер расширился (второй случай). В третьем случае мы вообще специально вынесли содержимое буфера за функцию ob_end_flush(), чтобы показать, что в этом месте буфер для функции ob_get_contents() окажется недоступным, т. е. пустым (это последние строчки результата нашего примера).

Если нужно, не обращаясь к данным, получить длину буфера, используйте функцию ob_get_length(). Ее **синтаксис:**

```
string ob_get_length(void);
```

Возвращает длину активного буфера. Если буфер не является активным, функция возвращает false. Рассмотрим пример:

```
<?php
ob_start();
echo "Строка";
$count = ob_get_length();
echo "Имеющийся буфер имеет следующую длину символов:<br>";
echo $count;
ob_end_flush();
?>
```

Как считает длину функция ob_get_length(), видно из результата выполнения скрипта:

```
Строка

Имеющийся буфер имеет следующую длину символов:

6
```

Мы специально ввели одно слово в наш буфер, чтобы вам было легче посчитать символы в слове «строка» (именно это одно слово содержится в нашем буфере). Как видите, результат показал правильное количество символов. Теперь длина нашего буфера при его использовании будет составлять именно такое количество символов — шесть.

Функции ob_get_contents() и ob_get_length() **работают в PHP** 4.0.2 **и выше.**

28.4. Функция работы с кодированными страницами

В этом параграфе рассмотрим функцию ob_gzhandler, которая мало распространена при использовании буферов, но иногда являются незаменимым помощникам.

Функция используется при работе с кодированными данными. Ее синтаксис:

```
string ob_gzhandler(string buffer)
```

Функция ob_gzhandler() предназначена для того, чтобы использовать ее как функцию повторного вызова для функции ob_start(). Это прежде всего необходимо для того, чтобы облегчить передачу gz-кодированных данных в браузер, который поддерживает работу с кодированными страницами. Рассмотрим пример:

```
<?php
ob_start("ob_gzhandler");
?>
<html>
<body>
<p>На этом месте должна быть кодированная страница
</html>
</body>
```

Функция ob_gzhandler() **используется в PHP 4.0.4 и выше.**

Заключение

В данной главе вы подробно ознакомились с функциями работы с буферами, научились не только получать, но и выводить данные, а также определять длину буфера. Все это вы посмотрели на примерах и теперь вы в состоянии самостоятельно

производить действия с буферами. Приведем функции, которые были изучены в этой главе:

- освобождение буфера вывода:

```
flush();
```

- включение буфера, пересылка и очистка:

```
ob_start(),
ob_end_flush(),
ob_end_clean();
```

- возврат значений буфера вывода:

```
ob_get_contents(),
ob_get_length();
```

- функция работы с кодированными страницами:

```
ob_gzhandler ().
```

Глава 29
Функции регулярных выражений и правила их формирования

Регулярные выражения — сочетания символов, позволяющих производить операции со строками (например, перевод строки) на основе заданного шаблона. Функции, которые позволяют поддерживать регулярные выражения, приведены ниже:

```
ereg()
ereg_replace()
eregi()
eregi_replace()
split()
spliti()
```

Все эти функции назначают строку регулярного выражения в качестве первого параметра. PHP использует установленные регулярные выражения POSIX, определенного как POSIX 1003.2.

Рассмотрим простые правила формирования шаблона.

Шаблон составляется из модификаторов, приведеных в табл. 29.1.

Таблица 29.1. Набор модификаторов

Модификатор	Значение
\	Следующий символ является специальным Применяется также совместно с другими символами
\n	Соответствует символу перевода строки
*	Предыдущий символ встречается 0 или более раз
^	Маркер начала строки
^abc	Строка, начинающаяся с «abc»
$	Маркер конца строки
abc$	Строка, заканчивающаяся на «abc»
+	Предыдущий символ встречается более одного раза. Например, шаблону w+ соответствуют строки what, agwt, www. Строка buka не соответствует
?	Предыдущий символ встречается не более одного раза. Например, шаблону w?r соответствуют строки ara, awra
.	Соответствует любому символу, отличному от \n

Остальные модификаторы при необходимости можно изучить на сайте www.php.net.

Руководствуясь этими простейшими модификаторами, можно приступить к рассмотрению функции.

В PHP существует несколько функций для работы с регулярными выражениями: ereg(), ereg_replace(), eregi(), ereg_replacei().

Функции с суффиксом i аналогичны функциям без этого суффикса, но не чувствительны к регистру операндов.

Функция ereg() возвращает true, если заданные соответствия pattern были найдены в string, или false, если не были найдены никакие соответствия или же была вызвана ошибка работы программы.

Синтаксис функции ereg():

```
int ereg(string pattern, string string, array [regs]);
```

В качестве параметра string указывается строка, соответствие которой ищется в регулярном выражении, заданном в параметре pattern.

Если соответствия найдены в указанном регулярном выражении и функция вызывается вместе с третьим аргументом regs, то найденные соответствия будут записа-

ны в качестве элементов массива regs. $regs[1] будет содержать подстроку, начинающуюся с круглой скобки; $regs[2] будет содержать подстроку, начинающуюся со второй позиции, и т. д. $regs [0] будет содержать копию строки.

Если функция ereg () найдет какое-либо соответствие, $regs будет заполнена точно на десять элементов, Даже если было найдено больше или меньше соответствий. Если не было найдено ни одного совпадения $regs не будет изменен при помощи ereg().

Приведем пример:

```
ereg ("abc", $string);
/* функция возвратит true,
если "abc" найдено в строке $string */
ereg ("^abc", $string);
/* функция возвратит true, если "abc"
найдено в начале строки $string */
ereg ("abc$", $string);
/* функция возвратит true, если "abc"
найдено в конце строки $string */
```

Чтобы произвести замену подстроки в строке, используют функцию ereg_replace(). Ее синтаксис:

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

В качестве параметра pattern указывается подстрока, поиск которой осуществляется функцией ereg_replace (). В параметре replacement указывается сочетание символов, на которое будет заменяться найденная подстрока. В параметре string указывается исходная строка, в которой производится поиск. Другими словами, подстрока pattern будет найдена в строке string и заменена на replacement.

Приведем пример:

```
$string = ereg_replace ("$", "<br>", $string);
/* функция позволит встроить тег
<br> в конец строки $string */
$string = ereg_replace ("abc", "cba" $string);
/* функция позволит произвести замену
"abc" на "cba" в строке $string */
$string = ereg_replace ("\n", "", $string);
```

```
/* функция позволит избавиться
от новых строк символов в $string */
```

Приведем пример, позволяющий более детально рассмотреть совместимость работы шаблона и изученных функций.

Возьмем электронный адрес sashamazurkevich@mail.ru. Очевидно, что правдоподобный адрес должен иметь вид слово@слово.слово. В терминах шаблонов произвольный символ обозначается знаком «.» (мы не будем сейчас учитывать тот факт, что в адресах допустимы не все символы). В каждом слове должен быть, по крайней мере, один символ, таким образом, шаблон слова будет иметь вид .+. Вспомним теперь, что «.» — это модификатор, и для явного указания точки (в качестве символа) нужно писать «\.». Таким образом, шаблон будет иметь вид .+@.+\..+.

Проверка будет иметь следующий вид:

```
if (ereg(".+@.+\..+", $email)) {
echo "Адрес правильный";
}
else {
echo "Введите адрес заново";
}
```

После такой проверки можем быть уверены, что e-mail имеет вид слово@слово.слово.

Заключение

Работа с регулярными выражениями является принципиально важной, так как знание и умение владеть этими функциями позволит вам безошибочно работать с преобразованиями строк.

Нами были изучены функции:

```
ereg(),
ereg_replace(),
eregi(),
eregi_replace().
```

Эти функции позволяют осуществлять работу с регулярными выражениями. Функция ereg() позволяет производить поиск указанных соответствий в строке, функция ereg_replace() производит не только поиск, но и замену на необходимую подстроку.

Следующие две функции работают аналогично описанным ранее, единственное отличие их состоит в том, что суфикс i показывает, что функция не чувствительна к регистру.

Для более полного понимания принципов работы данных функций проделайте операции с различными строками и модификаторами самостоятельно на практике.

Глава 30
Функции семафоров и разделяемой памяти

Семафоры («сигнальные функции») — специальные функции, которые применяются для обеспечения эксклюзивного доступа к ресурсам машины или для ограничения числа процессов, которые могут одновременно использовать ресурс. В PHP используются семафоры на основе System V. Поддержка разделяемой памяти также включена. Она может быть использована для обеспечения доступа к глобальным переменным.

ВНИМАНИЕ

Разделяемая память не может осуществить запоминание при одновременном доступе нескольких процессов.

Sem_get

Получение идентификатора семафора.

Синтаксис:

```
int sem_get(int key, int [ max_acquire ], int [ perm ]);
```

Возвращает положительный идентификатор семафора при успехе или false при ошибке.

`Sem_get()` возвращает идентификатор, который может быть использован для доступа к семафору System V с указанным ключом. Семафор создается, если необходимо, используя биты доступа, указанные в perm (по умолчанию 0666). Число процессов, которое может быть зафиксировано семафором одновременно, устанавливается в `max_acquire` (по умолчанию 1). В действительности это значение устанавливается, только если процесс обнаруживает, что он единственный, присоединенный к семафору.

Повторный вызов функции `sem_get()` с тем же ключом вернет другой идентификатор семафора, но оба идентификатора указывают на один и тот же семафор.

Sem_acquire

Фиксирует семафор.

Синтаксис:

```
int sem_acquire(int sem_identifier);
```

Возвращает true при успехе, false при ошибке.

Sem_acquire 0 блокируется (если необходимо) до тех пор, пока семафор сможет быть зафиксирован. Процесс, пытающийся зафиксировать семафор, который уже зафиксирован, будет блокирован навсегда, если фиксация семафора вызовет превышение его значения sem_identifier.

После обработки запроса, любые семафоры, зафиксированные процессом, но не освобожденные вручную, будут освобождены автоматически с выдачей предупреждения.

Sem_release

Освобождает семафор.

Синтаксис:

```
int sem_release (int sem_identifier);
```

Возвращает true **при успехе и** false **при ошибке.**

sem_release() освобождает семафор, если он зафиксирован в данное время вызывающим процессом, иначе выдается предупреждение.

После освобождения семафора функция sem_acquire () может быть вызвана для его рефиксации.

Shm_attach

Создает или открывает разделяемую память.

Синтаксис:

```
int shm_attach(long key, long memsize, long perm);
```

Создает или открывает разделяемую память с указанным ключом и размером памяти.

Shm_detach

Отсоединяет от разделяемой памяти.

Синтаксис:

```
int shm_detach(long id);
```

Отсоединяет от разделяемой памяти с указанным ID, созданным с помощью функции shm_attach(). Помните, что разделяемая память все еще существует в Unix-системе и данные все еще присутствуют.

Shm_put_var

Вставляет или обновляет переменную в разделяемой памяти.

Синтаксис:

```
int shm_put_var(int id, long variable_key, mixed variable);
```

Вставляет или обновляет переменную с указанным `variable_key`. Все типы переменных (`double`, `long`, `string`, `array`) поддерживаются. Функция `serialize()` может быть использована для хранения данных.

Shm_get_var

Считывает переменную с указанным `variable_key`.

Синтаксис:

```
mixed shm_get_var(int id, long variable_key);
```

Считывает переменную с указанным `variable_key`. Переменная все еще присутствует в разделяемой памяти. Функция `unserialize()` может быть использована для декодирования данных.

Shm_remove_var

Удаляет переменную из разделяемой памяти.

Синтаксис:

```
int shm_remove_var(int id, long variable_key);
```

Удаляет переменную с указанным `variable_key` из разделяемой памяти.

Shm_remove

Удаляет разделяемую память.

Синтаксис:

```
int shm_remove(long key);
```

Удаляет разделяемую память из Unix-системы. Все данные будут уничтожены.

Заключение

Из этой главы вы узнали о функциях семафоров и разделяемой памяти. Самое главное, что вам нужно принять во внимание, это то, что семафоры могут применяться для обеспечения эксклюзивного доступа к ресурсам используемой машины или для ограничения числа процессов, которые могут одновременно использовать ресурс.

Глава 31
Сессии в PHP

На разнообразных сайтах, посвященных программированию, в разделах о Web-программировании, в частности в форумах на эту тему, часто задают вопрос: «Что такое сессии в PHP и зачем они нужны?».

В данной главе рассмотрены следующие вопросы;

- понятие сессий в PHP;
- работа с сессиями;
- практическое применение;
- безопасность.

31.1. Понятие сессий в PHP

Когда появились первые версии PHP, программисты столкнулись с такой проблемой, как отсутствие глобальных переменных. Выполнялся некий скрипт, посылал результирующую HTML-страницу клиенту, и все ресурсы, используемые этим скриптом, уничтожались. Рассмотрим пример. Допустим, есть две страницы одного сайта — index.php и page1.php. Вот исходный код этих страниц:

Пример 31.1. Файл index.php

```
<?php
$a = "Test index.php";
?>
<htmlxbody>
<?php
echo $a;
?>
</body></html>
```

Пример 31.2. Файл page1.php

```
<htmlxbody>
<?php
echo $a;
?>
</body></html>
```

Если выполнить эти два скрипта, то на первой странице мы увидим надпись «Test index.php», а вторая страница будет пустой, так как значение переменной $a не передалось на вторую страницу сайта.

Тогда разработчики сайтов стали использовать Cookies для хранения глобальных переменных на стороне клиента. Этот процесс выглядел примерно так: пользователь приходит на главную страницу сайта и производит какие-то действия. Вся информация, которая связана с этим пользователем и может потребоваться на других страницах сайта, будет храниться у него в браузере в виде Cookies. У этого метода есть серьезные минусы, из-за которых от PHP в свое время отвернулось немало программистов. Например, нужно авторизовать пользователя, чтобы разрешить ему доступ к неким закрытым разделам сайта. В таком случае придется оставить на машине пользователя Cookies, который будет служить его идентификатором на сайте. Такой способ очень громоздкий и неудобный, ведь всю информацию, посылаемую пользователю, желательно кодировать, дабы избежать утечки. Еще совсем недавно подделкой Cookies можно было «взломать» не один чат или пробраться в чужую почту. А если у пользователя стоит браузер, который не поддерживает Cookies или они у него просто отключены?

И вот тут на помощь пришли сессии, при использовании которых вся информация хранится не на стороне клиента, а непосредственно на сервере. К тому же работать с сессиями куда проще и удобнее. В браузере клиента хранится лишь уникальный идентификатор номера сессии, либо в форме Cookies, либо в форме переменной в адресной строке браузера. Какой способ использовать для передачи идентификатора сессии между страницами, интерпретатор PHP выбирает сам. И это на 100 % безопасно, так как идентификатор сессии уникален, и подделать его практически невозможно.

31.2. Работа с сессиями

Если при тестировании примеров из главы или ваших собственных появятся ошибки примерно такого содержания:

```
Warning: open(/var/state/php/sess_6f71d1dbb52fa88481e752af7f384db0,
O_RDWR) failed: No such file or directory (2),
```

это значит, что ваш PHP настроен неправильно. Эта проблема решается следующим образом: нужно записать правильный путь (на существующую директорию), в которой будут храниться сессии, в файле php.ini и перезапустить сервер.

Любой скрипт, который будет использовать переменные (данные) из сессий, должен содержать следующую строку:

```
session_start();
```

Эта команда говорит серверу, что данная страница нуждается во всех переменных, которые связаны с пользователем (браузером). Сервер берет эти переменные из файла либо из БД и делает их доступными. Очень важно открыть сессию до того, как какие-либо данные будут посылаться пользователю. На практике это значит, что функцию session_start () желательно вызывать в самом начале страницы, например так:

```
<?php
session_start();
?>
<html>
<head>
</head>
...
```

ВНИМАНИЕ

Если в сессии хранится объект некоторого класса, то описание класса должно быть раньше старта сессии.

После начала сессии можно задавать глобальные переменные. Это элементарно: вызывается функция session_register('var_name'); и переменная $var_name становится доступной на всех страницах, использующих сессию. Модифицируем примеры 31.1 и 31.2.

Пример 31.3. Сессии в PHP. Файл index.php

```
<?php
// открываем сессию
session_start();
// задаем значение переменной
$a = "Меня задали на index.php";
// регистрируем переменную с открытой сессией
// внимание: названия переменных передаются функции
// session_register() без знака $
session_register("a");
?>
<html>
<body>
Все ОК. Сессию загрузили!
Пройдем, посмотрим, что <a href="page1.php>там...</a>
</body>
</html>
```

Пример 31.4. Сессии в PHP. Файл page1.php

```
<?php
// открываем сессию
session_start();
?>
<html>
<body>
<?php
echo $a;
?>
</body>
</html>
```

При последовательном запуске этих файлов первый скрипт (index.php) выдаст следующий результат:

Все OK. Сессию загрузили! Пройдем, посмотрим что там...

А второй (page1.php) вот это:

Меня задали на index.php

Таким образом, переменная $а теперь доступна на всех страницах данного сайта, на которых запущены сессии.

Другие полезные функции для работы с сессиями:

session_unregister (string) — сессия удаляет значение заданной глобальной переменной;

session_destroy (string) — сессия уничтожается (например, если пользователь покинул систему, нажав кнопку «Выход»);

session_set_cookie_params (int lifetime[, string path [, string domain]]) — с помощью этой функции можно установить, как долго будет «жить» сессия, задав unix_timestamp определяющий время «смерти» сессии. По умолчанию сессия существует до тех пор, пока клиент не закроет окно браузера.

31.3. Практическое применение

Теперь обратимся к практическому применению механизма сессий. Рассмотрим пару довольно простых и в то же время полезных примеров.

Авторизация Пользователя

Вопросы по авторизации пользователей с помощью PHP-сессий постоянно задаются в конференциях по Web-программированию. Механизм авторизации

пользователей в системе с помощью сессий довольно хорош с точки зрения безопасности (см. п. 31.4).

Наш пример будет состоять из трех файлов: index.php, authorize.php и secretplace.php. Файл index.php содержит форму, в которую пользователь введет свой логин и пароль. Эта форма передаст данные файлу authorize.php, который в случае успешной авторизации допустит пользователя к файлу secretplace.php, а в противном случае выдаст сообщение об ошибке.

Пример 31.5. Практическое применение. Файл index.php

```
<html>
<head>
<title>Введите пароль</title>
</head>
<body>
<form action="authorize.php" method="post">
Логин:<input type="text" name="user_name"><br>
Пароль:<input type="password" name="user_pass"><br>
<input type="submit" name="Submit">
</form>
</body>
</html>
```

Пример 31.6. Практическое применение. Файл authorize.php

```
<?php
// открываем сессию
session_start();
// проверяем, были ли данные отправлены формой
if($Submit){
// проверяем данные на правильность. В этом случае
// имя пользователя и пароль вписаны прямо в код, но на практике
// целесообразней проверять логин/пароль в базе данных
// и при совпадении давать доступ пользователю
if(($user_name=="cleo")&&($user_pass=="password")){
$logged_user = $user_name;
```

```
// запоминаем имя пользователя
session_register("logged_user");
// и переправляем его на "секретную" страницу
header("Location: secretplace.php");
exit;
}
}
// если что-то было не так, то пользователь получит сообщение об ошибке
?>
<htmlxbody>
Вы ввели неверный пароль!
</body></html>
```

Пример 31.7. Практическое применение. Файл secretplace.php

```
<?php
// открываем сессию
session_start();
/*
просто зайти на эту страницу нельзя. Если
имя пользователя не зарегистрировано, то
перенаправляем его на страницу index.php
для ввода логина и пароля. Тут можно
много чего сделать, например запомнить
IP пользователя, и после третьей попытки доступа
закрыть этому пользователю доступ к файлам
V
if(!isset($logged_user)){
header("Location: index.php");
exit;
}
?>
```

```
<html>
<body>
Здравствуйте, <?php echo $logged_user; ?>, вы на секретной
странице!
</body>
</html>
```

31.4. Безопасность

Итак, мы умеем передавать идентификатор от одной страницы (PHP-скрипта) к другой (до следующего вызова с нашего сайта), а значит, мы можем различать всех посетителей сайта. Идентификатор сессии — это очень большое число (128 битов) и шансов, что его удастся подобрать перебором, практически нет. Поэтому злоумышленнику остаются следующие возможности:

- на компьютере пользователя устанавливают вирус типа «троянский конь», который ворует номера сессий;
- отлавливается трафик между компьютером пользователя и сервером. Конечно, есть защищенный (зашифрованный) протокол SSL, но им пользуются не все;
- незаметно подойти к компьютеру администратора и зафиксировать номер сессии.

Такие ситуации, основанные на том, что кто-то что-то у кого-то украдет, не входят в компетенцию программиста. Об этом должны заботиться администраторы и сами пользователи.

Впрочем, PHP очень часто можно «обмануть». Рассмотрим возможные точки взлома в программе авторизации пользователя.

Файл `authorize.php` — попытка подбора пароля с помощью скрипта.

Файл `secretplace.php` — попытка «обмануть» программу путем вписывания значений переменной $logged_user в адресной строке браузера, например, так: `http://www.yoursite.ru/secretplace.php?logged_user=my_host`

Итак, в нашей программе явно видны две «дыры». Одна маленькая и не особо заметная, а вот вторая — просто огромная, через которую большинство хакеров и делает то, что не надо.

Как устранить «дыру» номер 1? Не будем писать тонны кода по блокировке IP-адреса и т. п., а просто проверим, откуда приходит запрос, а точнее, с какой страницы. Если это будет любая страница нашего сайта, то все нормально, а во всех остальных случаях пускать не будем. Подкорректируем файл authorize.php примера 31.6:

```
<?php
// открываем сессию
```

```
session_start();
// полный путь к корневой директории,
// в которой где расположены скрипты
$SERVER_ROOT = "http://localhost/test1/";
// если пользователь пришел с любой страницы нашего сайта,
// то он наш.
// Переменная $HTTP_REFERER всегда доступна по умолчанию
// и содержит полный адрес ссылающейся страницы
// функция eregi() проверяет, начинается ли адрес ссылающейся
// страницы со значения в переменной $SERVER_ROOT
if(eregi("^$SERVER_ROOT",$HTTP_REFERER)){
// проверям, были ли данные отправлены формой
if($Submit){
// далее все, как раньше
if(($user_name=="cleo"&&($user_pass=="password")){
$logged_user = $user_name;
// запоминаем имя пользователя
session_register("logged_user");
// и переправляем его на "секретную" страницу
header("Location: secretplace.php");
exit;
}
}
}
?>
<html><body>
Вы ввели неверный пароль!
</bodyx/html>
```

Как избавиться от «дыры» номер 2? Предположим, у вас есть сайт, где каждый может зарегистрироваться, чтобы добавлять сообщения в форум. Естественно, в форуме некоторые пользователи (например, администраторы, модераторы) имеют больше возможностей, чем другие. Они, например, могут удалять сообщения других пользователей. Уровень доступа пользователя хранится в сессии, в перемен -

ной $user_status, где $user_status = 10 соответствует полному доступу к системе. Пришедшему на сайт злоумышленнику достаточно зарегистрироваться обычным образом, а потом дописать в адресной строке браузера ?user_status=10. Вот и появился у вас на форуме новый администратор.

В принципе, любую переменную скрипта можно задать через адресную строку, просто дописав после полного адреса к скрипту вопросительный знак и название переменной с ее значением. Поправим код примера 31.7, чтобы избежать этой «дыры»:

```
<?php
// убираем все лишнее из адресной строки
// функция unset () "освобождает" переменную
unset($logged_user);
// открываем сессию
session_start();
// и корректируем "испорченные" переменные.
// Внимание: в этом случае переменная регистрируется не как
// новая, а как уже существующая, потому знак $ не опускается
session_register($logged_user);
if(!isset($logged_user)){
header("Location:index.php");
exit;
}
?>
<html>
<body>
Здравствуйте, <?php echo $logged_user; ?>, вы на секретной
странице!
</body>
</html>
```

Заключение

Сессии — очень удачная особенность языка PHP. Они просты и очень гибки в использовании. Есть еще одна возможность сессий PHP (доступна начиная с версии 4.0.3) — в них можно хранить не только переменные, но и объекты.

С выходом в свет PHP4.1.0 работа с сессиями значительно облегчилась. Все переменные сессий стали доступны из глобального массива $_SESSION['var_name']. Самое приятное в том, что при присвоении какого-либо значения любому полю массива переменная с таким же именем автоматически регистрируется как переменная сессии, например:

```
<?
_SESSION['counter'] = 12;
echo $counter;
?>
```

Эта программа выведет на экран браузера число 12.

Глава 32
Принципы работы с базой данных MySQL

Знание принципов программирования PHP — это далеко не гарантия того, что при работе на PHP в Сети вы будете ощущать себя, как рыба в воде. Все дело в том, что существуют задачи, которые требуют более детального и продуманного изучения языка. Решение данного вопроса требует не только владеть основной частью PHP, но и знать принципы взаимосвязи с объектами извне, такими, как базы данных, разнообразные модули или библиотеки. PHP позволяет получать доступ к широкому перечню баз данных. Мы рассмотрим наиболее распространенную и широко известную базу данных MySQL. Прежде чем начинать обзор принципов работы с базой данных, расскажем, что же представляет из себя MySQL и почему мы выбрали именно ее, а не какую-либо другую:

- MySQL — компактный, многопоточный сервер баз данных. MySQL характеризуется большой скоростью, устойчивостью и легкостью в использовании;
- MySQL был разработан компанией TcX для собственного использования, а именно для быстрой обработки очень больших баз данных. Компания утверждает, что использует MySQL с 1996 года на сервере с более чем 40 базами данных, содержащими 10 000 таблиц, из которых 500 имеют более 7 млн строк;
- MySQL является идеальным решением для малых и средних приложений. Исходные тексты сервера компилируются на множестве платформ. Наиболее полно возможности сервера проявляются на Unix-серверах, которые поддерживают многопоточность, что дает значительный прирост производительности. Под Windows MySQL может запускаться как сервис Windows NT или как обычный процесс на Windows 95/98;

- MySQL-сервер является бесплатным для некоммерческого использования. Иначе необходимо приобретение лицензии, стоимость которой составляет 190 EUR. Это далеко не последняя причина, по которой множество людей устанавливают и изучают эту базу данных.

Что может эта база данных:

- MySQL поддерживает язык запросов SQL в стандарте ANSI 92 и кроме этого имеет множество расширений к этому стандарту, которых нет ни в одной другой СУБД;
- поддерживается неограниченное количество пользователей, одновременно работающих с базой данных;
- количество строк в таблицах может достигать 50 млн;
- быстрое выполнение команд. Возможно, MySQL самый быстрый сервер из существующих;
- простая и эффективная система безопасности.

По словам создателей, именно все эти особенности дали возможность достичь высокого быстродействия сервера MySQL, но их реализация существенно снижает скорость Web-сервера. Эти возможности не являются критичными при создании Web-приложений, что в сочетании с высоким быстродействием и малой ценой позволило серверу MySQL приобрести большую популярность.

В этой главе рассмотрены следующие вопросы:

- установка MySQL;
- функции получения доступа;
- функции открытия и закрытия соединений;
- функции возврата сообщений;
- функция создания БД;
- переход на указанную строку;
- вызов строки результатов БД;
- списки потоков записи на сервере;
- практическая реализация.

32.1. Установка MySQL

Чтобы работа функций MySQL была максимально эффективной, т. е. чтобы не возникали никакие ошибки, необходимо как можно более правильно осуществить установку самой базы данных, совмещенной с PHP. Для этого вам при установке PHP обязательно нужно указать поддержку MySQL, используя опцию `--with-mysql`.

Если не указывать путь к базе данных, PHP автоматически воспользуется встроенными MySQL-библиотеками. В современных версиях данная поддержка существует и является весьма эффективной при работе с базой данных.

Пользователи, выполняющие другие приложения, применяемые MySQL (например, функционирование PHP 3 и PHP4 в качестве Apache-модулей или auth-mysql), всегда должны указывать точный путь к MySQL (например, --with-mysql=/path/to/mysql). С учетом этого можно использовать библиотеки клиента, установленные MySQL, избегая любых конфликтов.

Для более понятного разъяснения опишем установку под Windows NT:

- копируем дистрибутив последней версии MySQL по адресу http://www.mysql.com, распаковываем, запускаем setup.exe;
- после окончания установки в каталоге MySQL будет находиться файл my-example.cnf, который надо скопировать в каталог c:\ под именем my.cnf. При этом копирование можно осуществить либо ссылаясь на файл readme, либо запустив winmysqladmin.exe;
- в опциях setup выбираем закладку my.ini Setup, указываем директорию, в которой находятся поддиректории MySQL (например, /usr/local/mysql);
- выбираем запуск MySQL-сервера как службы NT в опциях mysqld file mysqld-nt;
- нажимаем на «Save Modification», после чего в WINNT-каталоге обнаружите файл my.ini.;
- теперь осталось убедиться, что в списке служб появилась «mysql», и выставить ей атрибут запуска (вручную/автоматически).

На этом установка закончена. После этой процедуры вы можете без страха приступать к работе с MySQL-функциями, не опасаясь за какие бы то ни было ошибки при программировании из-за неэффективной установки.

32.2. Функции получения доступа

В любом из языков программирования существует функции, позволяющие получать, вносить, редактировать те или иные значения или параметры. Для более легкого и систематического изучения функций работы с MySQL они разделены по их принципиальному назначению. В этом параграфе рассмотрены функции, позволяющие получать доступ к базе данных:

- mysql_affected_rows(),
- mysql_db_name(),
- mysql_fetch_field(),
- mysql_fetch_row(),
- mysql_field_flags(),
- mysql_field_name(),
- mysql_field_table(),
- mysql_field_type(),
- mysql_num_fields(),
- mysql_tablename().

Функция mysql_affected_rows() позволяет получить число строк, на которые воздействует последний UPDATE, DELETE или INSERT, т. е. число строк, активных в предыдущей операции MySQL. Функция mysql_affected_rows() имеет следующий синтаксис:

```
int mysql_affected_rows([int link_identifier])
```

Функция возвращает нуль, больше нуля или минус один. Если идентификатор связи link_identifier не определен, функция примет по умолчанию последнюю связь, открытую функцией mysql_connect() (с принципами работы этой функции вы ознакомитесь далее).

Функция mysql_affected_rows () показывает число измененных строк, если возвращаемое значение больше нуля. Если возращается нуль, то никакие записи не соответствовали значениям полей WHERE в UPDATE или DELETE. Когда сам запрос возвратит ошибку, то значение целого число будет равно минус единице. Ошибкой считается, например, попытка добавить двойной первичный ключ в течение одного INSERT.

ВНИМАНИЕ

При использовании UPDATE MySQL не будет модифицировать столбцы, в которых новое значение подобно старому. Данное явление позволяет увидеть, что mysql_affected_rows() не может фактически равняться числу согласованных строк, а только количеству строк, на которые непосредственно воздействовал запрос.

Mysql_affected_rows () не работает с инструкциями SELECT, а только с инструкциями, которые модифицируют записи. Чтобы получить количество строк, возвращенных инструкцией SELECT, используют функцию mysql_num_rows () [int link_identifier] .

Пример 32.1. Получение доступа. Файл test.php

```
<?php
$mydate = mysql_connect("localhost", "username", "password");
mysql_select_db("database", $mydate);
$intres = mysql_query("SELECT * FROM table1", $mydate);
$introw = mysql_affected_rows($mydate);
$num_rows = mysql_num_rows($intres) ;
echo "$num_rows Rows\n";
echo "$introw \n";
?>
```

Опишем принцип работы данного примера. Функция mysql_connect () позволяет установить соединение с сервером MySQL. На данном этапе не стоит сильно обращать внимание на параметры, указанные в данной функции. После соединения с сервером MySQL необходимо соединиться с базой данных, указанной в переменной $mydate, на сервере, указанном под именем database. Для реализации этой задачи используется функция mysql_select_db(). Далее выполняется SQL-запрос, заданный в качестве переменной $mydate к базе данных, указанной как SELECT * FROM table1. Эта операция осуществляется при помощи функции mysql_query().

Функция mysql_affected_rows() **используется В PHP 3 и PHP 4.**

Чтобы получить результат, необходимо воспользоваться функцией mysql_db_name(). Ее синтаксис:

```
int mysql_db_name(int result, int row[, mixed field])
```

Mysql_db_name() возвращает целое число. В качестве параметра result применяется результирующий указатель из вызова функции mysql_list_dbs(). Параметр row является введением в набор результатов.

В случае возникновения ошибки возвращается false. Чтобы определить ее характер, используют функции mysql_errno() и mysql_error().

Пример 32.2. **Получение доступа. Файл index.php**

```
<?php
mysql_connect('localhost', 'username','password');
$list = mysql_list_dbs();
$i = 0;
$cnt = mysql_num_rows ($list);
while ($i < $cnt)
{
echo mysql_db_name($list, $i). "\n";
$i++;
}
?>
```

Функция mysql_db_name () применяется в PHP 3-3.0.6 и PHP 4.

Чтобы получить столбец информации результата и возвратить его в качестве объекта, используют функцию mysql_fetch_field(). Она позволяет возвратить объект, содержащий поле информации (табл. 32.1). Ее синтаксис:

```
object mysql_fetch_field(int result [, int field_offset])
```

Mysql_fetch_field() может использоваться для того, чтобы получить информацию относительно полей, запрошенных в качестве параметра result. Если полевое смещение не определено, то при помощи функции mysql_fetch_field() будет найдено следующее поле, которое до этого все же не было найдено.

Таблица 32.1. Свойства объекта, возвращаемые функцией mysql_fetch-field()

Свойство	Значение
table	Название таблицы, к которой относится столбец
max_length	Максимальная длина столбца
not_null	1, если столбец не может быть нулевым
primary_key	1, если столбец — первичный ключ
unique_key	1, если столбец — уникальный ключ
multiple_key	1, если столбец — не уникальный ключ
numeric	1, если столбец числовой
blob	1, если столбец большой двоичный объект
type	Тип столбца
unsigned	1, если столбец без знака
zerofill	1, если столбец заполнен нулями

Функция mysql_fetch_row() возвращает массив, который соответствует определенной строке или false, если больше нет строк. Синтаксис функции:

```
array mysql_fetch_row(int result)
```

mysql_fetch_row() выбирает одну строку данных, связанную с указанным идентификатором result. Строка возвращается как массив. Каждый столбец сохраняется в матричном смещении, начинающемся в смещении 0.

Последующий запрос к функции mysql_fetch_row() возвратит следующую строку в наборе результатов или false, если больше строк не существует.

Рассмотрим пример:

```
<?php
mysql_connect($host, $user, $password)
or die ("Could not connect");
$result = mysql_db_query("database", "select * from table")
or die ("Query failed");
// получаем данные столбца
```

```
$i = 0;
while ($i < mysql_num_fields($result)) {
echo "Information for column $i:<br>\n";
$meta = mysql_fetch_field($result);
if (!$meta) {
echo "No information available<br>\n";
}
echo "<pre>
blob: $meta->blob
max_length: $meta->max_length
multiple_key: $meta->multiple_key
name: $meta->name
not_null: $meta->not_null
numeric: $meta->numeric
primary_key: $meta->primary_key
table: $meta->table
type: $meta->type
unique_key: $meta->unique_key
unsigned: $meta->unsigned
zerofill: $meta->zerofill
</pre>";
$i++;
}
mysql_free_result($result);
?>
```

Функции mysql_fetch_field() и mysql_fetch_row() работают в PHP 3 и PHP 4.

Функции mysql_field_flags(), mysql_field_name(), mysql_field_table(), mysql_field_type() позволяют осуществлять разнообразные манипуляции с флагами. Функция mysql_field_flags() возвращает флаги указанного поля. Ее синтаксис:

```
string mysql_field_flags(int result, int field_offset)
```

Соответствующий флаг будет возвращен в качестве строки.

Флаги, возвращаемые функцией mysql_field_flags(), могут быть следующие: not_null, primary_key, unique_key, multiple_key, blob, unsigned, zerofill, binary, enum, auto_increment, timestamp.

Чтобы получить название указанного поля, необходимо воспользоваться функцией mysql_field_name(). Функция имеет такой же синтаксис, как и mysql fieldflags().

ВНИМАНИЕ

Параметр field_offset начинается с нуля.

Рассмотрим пример:

```
// таблица пользователя состоит из трех областей:
// lockalhost
// lockalhostl
// lockalhost2
$res = mysql_db_query("users", "select * from users", $link);
echo mysql_field_name($res, 0). "\n";
echo mysql_field_name($res, 2);
```

В результате работы данного примера получим следующий результат:

```
lockalhost
lockalhost2
```

Функция mysql_field_table() позволяет получить имя таблицы. Функция имеет схожий синтаксис с ранее описанными функциями и также возвращает строку.

Для определения типа поля используют функцию mysql_fields_type(). Тип поля может быть следующим: int, real, string, blob и др.

Рассмотрим пример:

```
<?php
mysql_connect("localhost:3306");
mysql_select_db("wisconsin");
$result = mysql_query("SELECT * FROM onek");
$fields = mysql_num_fields($result);
$rows = mysql_num_rows($result);
$i = 0;
$table = mysql_field_table($result, $i);
```

```
echo "Your '".$table."' table has ".$fields." fields and
".$rows." records<br>";
echo "The table has the following fields <br>";
while ($i < $fields) {
$type = mysql_field_type ($result, $i) ;
$name = mysql_field_name ($result, $i) ;
$len = mysql_field_len ($result, $i) ;
$flags = mysql_field_flags ($result, $i) ;
echo $type." ".$name." ".$len." " .$flags."<br>";
$i++;
}
mysql_close();
?>
```

Рассмотренные функции работают в PHP 3 и PHP 4.

Чтобы получить в качестве результата количество полей, используют функцию `mysql_num_fields()`. Ее **синтаксис:**

```
int mysql_num_fields(int result)
```

Функция возвращает целое число.

Функция `mysql_tablename()` позволяет получить имя поля таблицы. Функция имеет следующий синтаксис:

```
string mysql_tablename(int result, int i)
```

Функция `mysql_tablename()` использует указатель result, возвращенный функцией `mysql_list_tables()`, как целочисленный индекс и выводит название таблицы. Функция `mysql_num_rows()` может быть использована, чтобы определить число таблиц в поле, указанном параметром result. Например:

```
<?php
mysql_connect ("localhost: 3306") ;
$result = mysql_list_tables ("wisconsin");
$i = 0;
while ($i < mysql_num_rows ($result)) {
$tb_names [$i] = mysql_tablename ($result, $i) ;
echo $tb_names [$i]. "<br>";
```

```
$i++;
}
?>
```

Функции `mysql_num_fields()` и `mysql_tablename()` **используются в PHP 3 и PHP 4.**

32.3. Функции открытия и закрытия соединений

Перед началом работы с MySQL необходимо установить соединение, позволяющее получить доступ к полям и таблицам базы данных. После необходимо закрыть соединение. Правильно выполнять эти операции помогают следующие функции:

- `mysql_connect()`,
- `mysql_pconnect()`,
- `mysql_close()`.

Чтобы установить соединение с MySQL-сервером, работающим на компьютере, необходимо воспользоваться функцией `mysql_connect()`. Ее синтаксис:

```
int mysql_connect([string hostname [:port] [:/path/to/socket][,
string username [, string password]]])
```

Функция mysql_connect() возвращает MySQL-идентификатор связи при успешном выполнении, в противном случае возвращает нуль, что соответствует наличию ошибки. Значение параметра host может быть сетевым именем или IP-адресом. Он указывается следующим образом: `host:port = 'localhost:3306'`. Username задает логин пользователя MySQL. Password задает пароль для username.

Строка hostname (имя сетевого узла) может также включать номер порта, т. е. `hostname:port`, ИЛИ путь, К локальному узлу (lockalhost), т. е. : /path/to/socket.

Поддержка для `:port` была добавлена в PHP 3.0b4.

Поддержка для : /path/to/socket была добавлена в PHP 3.0.10.

Никакая новая связь не будет установлена, если второй запрос сделан к функции `mysql_connect()` с теми же самыми аргументами. Вместо этого идентификатор связи уже открытого соединения будет возвращен.

Соединение с сервером будет считаться закрытым, как Только выполнение сценария будет закончено, конечно при условии, что не произошло закрытие ранее вызовом функции mysql_close().

ВНИМАНИЕ

Если в функции mysql_connect() параметр hostname не задан, то подразумевается 'localhost'.

Если параметр username не задан, то подразумевается current user. В Windows ODBC текущий пользователь должен быть определен явно. В Unix подразумевается текущий логин.

Если параметр password не задан, то будут проверены только те записи в таблице пользователей, которые не имеют пароля. Это позволяет db-администратору настроить систему привилегий MySQL так, чтобы пользователь получал различные привилегии в зависимости от того, определен пароль или нет.

Рассмотрим пример использования функции mysql_connect

```
<?php
$link = mysql_connect ("localhost", "username", "secret");
print ("Соединение успешно установлено");
mysql_close($link);
?>
```

В начале данного скрипта, чтобы получить доступ к имеющейся базе данных, при помощи функции mysql_connect() устанавливается соединение. Далее выполняем какие-то действия (в нашем случае — это печать) и после этого закрывается установившееся соединение при помощи функции mysql_close().

Функция mysql_connect() работает в PHP 3 и PHP 4.

Функция mysql_pconnect позволяет возвратить идентификатор связи при соединении с работающим сервером MySQL при положительном выполнении операций, в случае ошибки функция возвратит значение false.

Ее синтаксис:

```
int mysql_pconnect([string hostname [:port] [:/path/to/socket] [,
string username[, string password]]])
```

Все параметры и аргументы, указывающиеся в функции mysql_pconnect(), подчиняются тем же правилам, что и для функции mysql_connect(), принципы работы которой были рассмотрены ранее.

Функция mysql_pconnect() имеет принцип работы, аналогичный функции mysql_connect(), однако существует важное отличие, соединение с сервером SQL не будет закрыто при завершении выполнения сценария, существующая связь останется открытой для дальнейшего использования. Функция mysql_close о не будет производить закрытие соединения, которое было установлено при помощи функции mysql_pconnect(). Именно поэтому данный вид соединения называется стойким.

После того как открытое соединение становится ненужным, т. е. все поставленные задачи выполнены, нужно закрыть соединение с MySQL-сервером. Для этого используют функцию mysql_close(). Ее синтаксис:

```
int mysql_close([int link_identifier])
```

Чтобы было ясно, какое именно установившееся соединение необходимо закрыть, в функции mysql_close() в качестве параметра указывается идентификатор связи этого соединения. В случае положительного исхода, т. е. если операция закрытия прошла успешно, функция mysql_close () возвратит true, в противном случае — false.

Функцию mysql_close () можно не использовать при нестойком соединении, так как такие соединения автоматически закрываются в конце выполнения скрипта.

Это является существенным отличием от функции mysql_close() в C++. Функция mysql_close () в C++ закрывает соединение и должна быть вызвана после завершения всех операций, выполняемых через соединение с MySQL. Если это не сделать, поток, созданный функцией mysql_connect(), зависнет до окончания тайм-аута сервера. На сервере, работающем с сильной нагрузкой, это может быстро израсходовать много памяти, хотя нужно очень немного времени центрального процессора.

Поэтому можно сказать, что работа функции mysql_close в PHP является заметно более эффективной, чем в описанном выше случае.

Обратите внимание еще раз, что функция mysql_close () не будет закрывать стойкие связи, созданные при помощи функции mysql_pconnect().

Рассмотрим пример:

```
<?php
$link = mysql_connect ("kraemer", "marliesle", "secret");
print ("Соединениеустановлено успешно") ;
mysql_close ($link);
?>
```

В приведенном примере функция mysql_close () закрывает установленное соединение, при этом в качестве идентификационного параметра используется указатель соединения $link. Вместоэтого указателя могут быть какие-либо другие, при этом будет закрыто именно то соединение с сервером MySQL, указатель которого и был задан в качестве параметра функции mysql_close().

Функция mysql_close () **работает в PHP 3 и PHP 4.**

32.4. Функции возврата сообщений

Функции, которые будут рассмотрены, позволяют осуществлять возврат каких-либо значений, вплоть до вывода ошибки:

- mysql_field_len() ,
- mysql_error () ,
- mysql_errno () .

Функция mysql_field_len () возвращает длину указанного поля. Ее синтаксис:

```
int mysql_field_len(int result, int field_offset)
```

Функция возвращает целое число. Например:

```
<?php
mysql_connect("localhost:3306");
mysql_select_db("wisconsin");
$result = mysql_query("SELECT * FROM onek");
echo "Поле имеет следующую длину" <br>";
$len = mysql_field_len ($result, $i) ;
echo $len." <br>";
mysql_close();
?>
```

В результате выполнения данного скрипта получится длина поля.

Функция `mysql_field_len()` работает в PHP 3 и PHP 4.

Если предыдущая операция при использовании какой-либо функции вызвала ошибку, то, чтобы вывести эту ошибку, необходимо воспользоваться функцией mysql_error(). Ее синтаксис:

```
string mysql_error([int link_identifier])
```

Функция выводит сообщение об ошибке, если ошибку вернула последняя вызванная функция MySQL. В противном случае возвращает пустую строку.

Ошибки, приходящие из базы данных MySQL, впоследствии не проявляются. Чтобы отыскать текст ошибки, необходимо воспользоваться функцией mysql_error(). Обратите внимание, что эта функция только возвращает текст ошибки от недавно выполненной функции MySQL (не включая функции `mysql_error()` и mysql_errno()), поэтому если вы хотите использовать это свойство, убедитесь в том, что вы проверили значение, прежде чем произойдет вызов следующей функции MySQL. Например:

```
<?php
mysql_connect("marliesle");
echo raysql_errno().": ".mysql_error()."<br>";
mysql_select_db("nonexistentdb");
echo mysql_errno().": ".mysql_error()."<br>";
$conn = mysql_query("SELECT * FROM nonexistenttable");
echo mysql_errno().": ".mysql_error()."<br>";
?>
```

Как видно из примера, чтобы обнаружить ошибку и получить текст этой ошибки, необходимо после каждой функции MySQL устанавливать функцию получения ошибки, иначе эта ошибка будет перекрыта другими функциями.

Функция mysql_error() работает в PHP 3 и PHP 4.

Функция mysql_errno() позволяет возвратить численное значение сообщения об ошибках от предыдущей операции MySQL. Его синтаксис:

```
int mysql_errno([int link_identifier])
```

В качестве параметра link_identifier может быть указан идентификатор связи, хотя очень часто данная функция используется без каких-либо параметров. В этом случае будет установлен параметр имеющегося соединения.

Функция mysql_errno() возвращает номер ошибки предшествующей функции MySQL или значение, равное нулю, если ошибка не произошла.

Функция mysql_errno() используется в PHP 3 и PHP 4.

32.5. Функция создания БД

Чтобы воспользоваться такими функциям, необходимо прежде всего иметь доступ не простого пользователя, а привилегированного. В этом случае есть возможность создавать и удалять базы данных при помощи следующих функций:

- mysql_create_db(),
- mysql_drop_db();

Функция mysql_create_db() позволяет создавать базу данных с именем datebase_name на машине, указанной в MySQL. Функция имеет следующий синтаксис:

```
int mysql_create_db(string database_name[, int link_identifier])
```

Функция возвращает целое число, равное нулю, если база данных создана успешно. При возникновении ошибки будет возвращено число, не равное нулю. Сообщение об ошибке можно получить при помощи функции mysql_error(), а ее номер — при помощи функции mysql_errno().

Параметры функции mysql_create_db() задаются следующим образом: datebase — новая база данных на сервере, link_identifier— идентификатор связи, указывающий на новую базу данных.

Рассмотрим пример:

```
<?php
$link = mysql_pconnect("myserver", "utta", "geheim");
if (mysql_create_db ("my_db"))
{
```

```
print ("База данных создана успешно\n");
} else {
printf ("Ошибка при создании базы данных: %s\n", mysql_error
());
}
?>
```

Как видно из примера, идентификатор связи может быть пропущен.

Чтобы удалить базу данных с именем, указанным в datebase_name на сервере, используют функцию mysql_drop_db(). Ее синтаксис:

```
int mysql_drop_db (string database_name [, int link_identifier])
```

Целое число будет равно нулю, если удаление базы данных datebase_name произошло успешно, в противном случае — будет возвращено число, не равное нулю.

Функция mysql_drop_db() используется в PHP 3 и PHP 4.

32.6. Переход на указанную строку

Иногда в ходе работы с базой данных возникает необходимость осуществить разнообразные переходы (со строки на строку, с поля на поле, со столбца на столбец, с адреса на адрес и т. д.). Для этого используется функция mysql_data_seek(), которая позволяет осуществлять переход на указанную строку в наборе результатов запросов. Ее синтаксис:

```
int mysql_data_seek(int result_identifier, int row_number)
```

Функция возвратит true в случае успешного выполнения работы или false — в случае ошибки.

Функция mysql_data_seek() перемещает внутренний указатель строки результата MySQL, связанного с указанным идентификатором (result_identifier), на установленный номер строки (row_number). При следующем запросе к функции mysql_fetch_row() может быть возвращена эта строка.

ВНИМАНИЕ

Значения параметра row_number начинаются с нуля.

Рассмотрим пример:

```
<?php
$link = mysql_pconnect ("kron", "jutta", "geheim")
mysql_select_db ("samp_db")
```

```
$query = "SELECT last_name, first_name FROM friends";
$result = mysql_query ($query)
for ($i = mysql_num_rows ($result) — 1; $i >=0; $i--) (
if (!mysql_data_seek ($result, $i)) {
printf ("Поиск невозможен %d\n", $i) ;
continue;
}
if(!($row = mysql_fetch_object ($result)))
continue;
printf ("%s %s<BR>\n", $row->last_name, $row->first_name);
}
mysql_free_result ($result);
?>
```

Функция mysql_data_seek() **используется в PHP 3 и PHP 4.**

32.7. Вызов строки результатов БД

Рассмотрим, как можно получать строки результатов в качестве ассоциативного массива, числового массива или объекта. Для решения этих задач существуют следующие функции:

- `mysql_fetch_array()`,
- `mysql_fetch_assoc()`,
- `mysql_fetch_object()`.

Функция `mysql_fetch_array()` позволяет вызывать строку результатов в качестве ассоциативного массива, числового массива или обоих. Ее синтаксис:

```
array mysql_fetch_array (int result [, int result_type])
```

Функция возвращает массив, который соответствует выбранной строке, или же `false`, если больше строк не имеется.

Функция `mysql_fetch_array()` является расширенной версией функции `mysql_fetch_row()`. Помимо сохранения данных в числовых индексах результирующего массива, функция позволяет хранить данные в качестве ассоциативного массива, используя имена как индекс этого массива.

Если два или более столбца результата имеют одинаковые имена, то последний столбец будет иметь приоритет по отношению к первому. Чтобы обратиться ко второму столбцу с таким же самым названием, вам необходимо воспользоваться

числовым индексом столбца или же создать условное название для столбца (например, select t1.f1 назвать как foo t2.f1 или как bar from t1, t2).

ВНИМАНИЕ

Функция mysql_fetch_array() работает не медленне, чем функция mysql_fetch_row().

Второй параметр функции mysql_fetch_array() — result_type — необязательный. Он является константой и может принимать следующие значения: MYSQL_ASSOC, MYSQL_NUM и MYSQL_BOTH. Этот параметр был добавлен в PHP 3.0.7.

Рассмотрим пример:

```
<?php
mysql_connect ($host, $user, $password);
$result = mysql_db_query ("database","select user_id, fullname
from table");
while ($row = mysql_fetch_array ($result)) {
echo "user_id: " .$row["user_id"]."<br>\n";
echo "user_id: " .$row[0]."<br>\n";
echo "fullname: " .$row["fullname"]."<br>\n";
echo "fullname: " .$row[1]."<br>\n";
}
mysql_free_result ($result);
?>
```

Функция mysql_fetch_array() работает в PHP 3 и PHP 4.

Функция mysql_fetch_assoc() позволяет вызывать строки результата как ассоциативного массива. Ее синтаксис:

```
array mysql_fetch_assoc (int result)
```

Функция возвращает ассоциативный массив, который соответствует выбранной строке или же false в случае отсутствия строк.

Функция mysql_f etch_assoc() эквивалентна запросу mysql_f etch_array 0 с необязательным вторым параметрам, имеющим значение MYSQL_ASSOC. В этом случае будет возвращаться ассоциативный массив. Таким способом пользовались в ранних версиях PHP. Если нужны числовые индексы такие же, как и ассоциативные , лучше использовать функцию mysql_fetch_array().

Рассмотрим пример:

```
<?php
mysql_connect ($host, $user, $password);
$result = mysql_db_query ("database", "select * from table");
while ($row = mysql_fetch_assoc ($result)) {
echo $row["user_id"];
echo $row["fullname"];
}
mysql_free_result ($result);
?>
```

Функция *mysql_fetch_assoc()* работает в PHP 4–4.0.3.

Существует еще одна функция, позволяющая вызывать строку результата в качестве объекта — *mysql_fetch_object()*. Ее синтаксис:

```
object mysql_fetch_object (int result [, int result_type])
```

Функция возвращает объект со свойствами, которые соответствуют выбранной строке, или `false` в случае отсутствия строк.

Функция `mysql_fetch_object()` подобна функции `mysql_fetch_array()`, однако существует небольшое отличие — вместо матрицы происходит возврат объекта. Это означает, что можно получать доступ к полям при помощи имени поля, а не их смещениями (числа — неправильные названия свойств).

Необязательный параметр `result_type` — константа и может принимать следующие значения: `MYSQL_ASSOC`, `MYSQL_NUM` и `MYSQL_BOTH`. Например:

```
<?php
mysql_connect ($host, $user, $password);
$result = mysql_db_query ("database", "select * from table");
while ($row = mysql_fetch_object ($result)) {
echo $row->user_id;
echo $row->fullname;
}
mysql_free_result ($result);
?>
```

Функция `mysql_fetch_object()` **работает в PHP 3 и PHP 4.**

32.8. Списки потоков записи на сервере

Очень часто программистам необходимо оценить ситуацию и с учетом полученной оценки выдать тот или иной результат либо просто совершить те или иные действия. Под оценкой действующей ситуаций понимается возможность распознать базы данных, полей в этих базах данных и таблиц соответственно на работающем сервере. Эту задачу решают функции, описанные ниже:

- mysql_list_dbs(),
- mysql_list_fields(),
- mysql_list_tables().

Функция mysql_list_dbs() позволяет получить список всех доступных баз данных на MySQL-сервере. Чтобы задать параметры данной функции и получить необходимый результат, достаточно указать только идентификатор связи. Рассмотрим синтаксис функции mysql_list_dbs():

```
int mysql_list_dbs ( [int link_identifier] )
```

Функция mysql_list_dbs() позволяет возвратить указатель результата, содержащий доступные данные на сервере MySQL. Чтобы получить промежуточный результат указателя, необходимо воспользоваться функцией mysql_tablename().

```
$link = mysql_connect ( ' localhost ' , 'myname', 'secret');
$db_list = mysql_list_dbs ($link);
while ($row = mysql_fetch_object ($db_list)) {
echo $row->Database. "\n";
}
```

Результат работы данного скрипта будет следующий:

```
database1
database2
database3
```

Данный код также работает с функцией mysql_fetch_row() и другими ранее упомянутыми функциями.

Функция mysql_list_fields() позволяет получать список полей результата. Ее синтаксис:

```
int mysql_list_fields (string database_name, string table_name
[, int link_identifier])
```

Функция mysql_list_fields() имеет информацию относительно заданного параметра table_name.

Функция возвращает положительное целое число. В случае возникновения ошибки функция возвращает –1. Строка, описывающая ошибку, будет помещена в переменную $phperrmsg даже в случае вызова функции со знаком @ (@mysql()) эта ошибка будет также распечатана.

Рассмотрим пример:

```
$link = mysql_connect('localhost', 'myname', 'secret');
$fields = mysql_list_fields("database1", "table1", $link);
$columns = mysql_num_fields($fields);
for ($i = 0; $i < $columns; $i++) {
echo mysql_field_name($fields, $i) . "\n";;
}
```

Результатом работы данного скрипта будут следующие строки:

```
field1
field2
field3
```

Чтобы получить название таблицы, необходимо воспользоваться функцией mysql_list_table(). Ее синтаксис:

```
string mysql_field_table (int result, int field_offset)
```

Функция возвращает строку с именем таблицы.

Описанные функции: mysql_list_dbs(),mysql_list_fields(),mysql_list_table() работают в PHP 3 и PHP 4.

32.9. Практическая реализация

Рассмотрим работу изученных функций на примере. Сначала создадим базу и внесем туда определенные данные, необходимые для дальнейшей работы. Для этого входим в командную строку MySQL и выполняем команды:

```
mysql > CREATE DATABASE mydatebase;
mysql> CREATE TABLE students
( id tinyint(4) DEFAULT '0' NOT NULL AUTO_INCREMENT,
first varchar(20), last varchar(20), address varchar(255),
position varchar(50), PRIMARY KEY (id), UNIQUE id (id));
INSERT INTO students VALUES
(1,'Саша','Мазуркевич','ул Маяковского','Староста группы');
```

```
INSERT INTO students VALUES
(2,'Игорь','Полещук','ул. Пушкина','Профорг');
INSERT INTO students VALUES
(3,'Виталий','Ивончик','ул. Высокая','Гений группы');
```

В результате будет создана база данных mydatebase с таблицей students, в которую будут вставлены три записи с данными о студентах.

Теперь выведем эти данные из базы данных. Для этого создадим файл PHP, который будет содержать следующий скрипт:

```
<html>
<body>
<?php
$date = mysql_connect("localhost", "root");
mysql_select_db("mydatebase", $date);
$my_result = mysql_query("SELECT * FROM students", $date);
printf("Имя: %s<br>\n", mysql_result($my_result, 0, "first"));
printf("Фамилия: %s<br>\n", mysql_result($my_result, 0, "last"));
printf("Улица: %s<br>\n", mysql_result($my_result, 0, "address"));
printf("Должность: %s<br>\n", mysql_result($my_result, 0, "position"));
mysql_close($date);
?>
</body>
</html>
```

Данный скрипт позволяет получать данные, содержащиеся в базе. Рассмотрим последовательность операций, выполняемых в скрипте. Прежде чем получить доступ к базе данных, необходимо произвести соединение с указанными параметрами. Для этого используется функция mysql_connect(), которая создает связь с сервером баз данных MySQL. В качестве параметров указано имя узла localhost, на котором находится база данных, имя пользователя — root, под которым мы будем с ней работать. Очень часто в качестве третьего параметра функции mysql_connect() задают пароль соединения. В нашем случае он не был указан.

В результате выполнения данной функции получим некое значение, которое будет присвоено переменной $date. Эту переменную называют идентификатором соединения. С учетом полученного значения можно сделать вывод о результате соединения с сервером MySQL.

Функция mysql_select_dbl() позволяет выбирать базу данных, с которой необходимо осуществить работу, в частности получить данные и вывести в браузер. В качестве параметров указываем само имя необходимой базы данных и идентификатор соединения, установленный при помощи предыдущей функции.

В результате выполнения функции mysql_select_db () получаем значение true или false. Если соединение с базой данных прошло успешно — true, если нет — false. Чтобы наша программа-страница работала лучше, можно проанализировать возвращаемое значение, и если оно будет false, вывести сообщение об ошибке.

После того как все необходимые операции соединения с базой данных выполнены, можно получить необходимые данные. Для этого служит функция mysql_query(). В качестве первого параметра передаем текст запроса, а в качестве второго — идентификатор, полученный от выполнения функции mysql_connect ().

Результаты выполнения функции mysql_query() — записи, удовлетворяющие нашему запросу — помещаем в переменную $my_result.

И наконец, с помощью функции mysql_result () извлекаем из результатов выполнения нашего запроса (т. е. переменной $my_result), первый ряд-запись (который имеет порядковый номер 0), и значение определенного поля (по его имени). Перебирая друг за другом записи от 0 до 2, извлечем все записи, которые хранятся в нашей маленькой базе данных.

Вот так легко можно работать с базой данных в PHP.

После того как мы научились извлекать данные из базы, попробуем осуществить противоположную операцию, т. е. внести данные в имеющуюся базу данных. Решая эту задачу в PHP, у вас не возникнет ни малейшей проблемы. Для этого создадим простую форму:

```
<html>
<body>
<form method="post" action="<?php echo $PHP_SELF?>">
Имя: <input type ="Text" name = "first"><br>
Фамилия: <input type="Text" name ="last"><br>
Улица: <input type = "Text" name = "address"><br>
Должность: <input type="Text" name="position"><br>
<input type = "Submit" name="submit" value="Ввод информации">
</form>
<body>
<html>
```

Обратите внимание, мы опять используем переменную $PHP_SELF. Как уже говорилось, PHP-код можно как угодно включать в обычный HTML. Также обратите

внимание, что название каждого элемента формы совпадает с названием поля в базе данных. Это не обязательно, но очень удобно, чтобы в дальнейшем не запутаться в том, какая переменная какому полю в базе данных соответствует.

Помимо этого мы присвоили имя кнопке «Submit». Это сделано, чтобы в коде затем проверить, есть ли переменная $submit. Таким образом, когда страница будет вызываться, можно узнать, вызывается ли она в первый или во второй раз.

Следует еще раз отметить, что вовсе не обязательно писать код так, чтобы страница снова и снова вызывала саму себя. Программу можно разделить на две, три и более страниц, если угодно. Но чаще бывает удобнее, когда вся программа находится в одном файле.

Теперь введем данным в базу:

```
<html>
<body>
if (submit) {
$date = mysql_connect ("localhost", "root");
mysql_select_db ("mydatebase", $date) ;
$sql = "INSERT INTO students (first, last, address, position)
VALUES
('$first', '$last', '$address', '$position')";
$result = mysql_query ($sql) ;
echo "Спасибо! Ваша информация введена\n";
}else {
?>
<form method="post" action="<?php echo $PHP_SELF?>">
Имя: <input type ="Text" name = "first"><br>
Фамилия: <input type="Text" name ="last"><br>
Улица: <input type = "Text" name = "address"><br>
Должность: <input type="Text" name="position"><br>
<input type = "Submit" name="submit" value="Ввод информации">
</form>
<?php
}
?>
</body>
</html>
```

Заключение

Чтобы систематизировать информацию, изученную при работе с базами данных, самостоятельно напишите какой-нибудь скрипт работы с базой данных, по возможности не заглядывая в книгу. После этого проанализируйте работу данного примера и поэкспериментируйте.

Только когда вы все проделаете самостоятельно, вы сможете овладеть принципами программирования баз данных.

Глава 33
Практическое применение PHP

Рассмотрим несколько несложных примеров скриптов, которые помогут вам лучше понять концепцию языка PHP и научиться реализовывать свои идеи.

В этой главе рассмотрены следующие вопросы:

- гостевая книга;
- оптимизатор кода HTML.

33.1. Гостевая книга

На большинстве сайтов, если не сказать почти на всех, присутствуют так называемые гостевые книги. И если вы, сделав свой сайт, тоже решили усовершенствовать его таким полезным скриптом, то сейчас мы поможем вам понять принцип создания такой «книги».

Самое главное в гостевой книге — это определиться, какие поля для заполнения вы предложите посетителям сайта. В большинстве гостевых книг это «Имя», «E-mail», «URL» домашней странички и, конечно, само сообщение (или какой-либо комментарий). Создадим форму для гостевой книги (рис. 33.1).

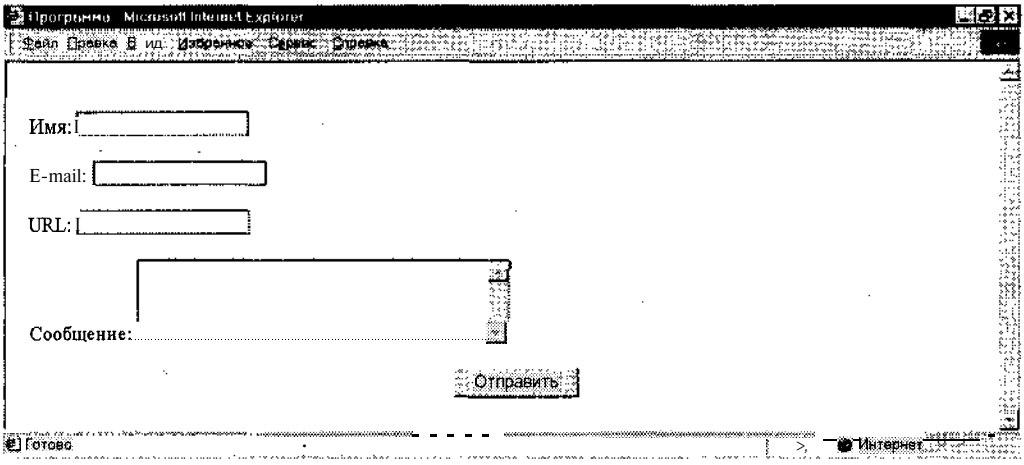


Рис. 33.1. Полученная форма

```
<form name="form1" method="post" action="guestbook.php"><br>
<font face="Times New Roman, Times, serif" size="3">Имя:
<input type="text" name="name"><br>
<br>E-mail: <input type="text" name="email"><br>
<br>URL: <input type="text" name="url"><br>
<br>Сообщение: <textarea name="msg" cols="35" rows="4"
wrap="VIRTUAL"></textarea></font><br>
<br><p align="center"><input type="submit"
name="Submit" value="Отправить"></p></form>
```

Можно сделать эту форму в таблице, чтобы улучшить ее внешний вид, добавить другие поля и т. д., но тогда придется подредактировать код скрипта.

Описывать полностью структуру формы не будем, подробное описание всех составляющих формы можно прочесть в предыдущих главах, а сразу перейдем к рассмотрению PHP-скрипта.

Итак, мы создали страницу с формой и назвали ее, например, guestbook.htm. На этой же странице будут выводиться новые сообщения. Скрипт будет отвечать за чтение данных из формы, их обработку, очистку от HTML-тегов и запись в файл с сообщениями (назовем его guest.txt). Новичку в PHP, на первый взгляд, все это может показаться очень сложным, но стоит приглядеться, и все становится на свои места. Через 15 минут вы сами сможете писать свои собственные гостевые книги.

```
/* мета-тег возвращает нас на страницу с гостевой книгой */
<meta http-equiv="refresh" content="0;URL=guestbook.htm">
/* здесь начинается скрипт PHP */
<?php
/* определяем файл, в который будут записываться новые сообщения */
$files = "guest.txt";
/* если посетитель не захотел оставлять свой E-mail, напишем
"нет" */
if ( $email=="" ) {$email="нет";}
/* здесь начинается раздел скрипта, убирающий кавычки < и >,
чтобы теги HTML не отобразились в гостевой книге */
$msg=eregi_replace("<","",$msg);
$msg=eregi_replace(">","",$msg);
```

```
$email=eregi_replace("<","",$email);
$email=eregi_replace(">","",$email);
$name=eregi_replace("<","",$name);
$name=eregi_replace(">","",$name);
/* кавычки можно написать как &lt; и &gt;, это мы тоже предусмотрим */
$msg=eregi_replace("&lt;","",$msg);
$msg=eregi_replace("&gt;","",$msg);
$email=eregi_replace("&lt;","",$email);
$email=eregi_replace("&gt;","",$email);
$name=eregi_replace("&lt;","",$name);
$name=eregi_replace("&gt;","",$name);
/* обрезаем каждую переменную на случай, если кто-то захочет побаловаться */
$msg=substr($msg,0,499);
$email=substr($email,0,39);
$name=substr($name,0,39);
/* эта часть кода работает только при заполненных полях "Имя" и "Сообщение" */
/* она как раз и отвечает за запись данных в файл */
if ($msg != "" && $name != "")
{
$time = Date("M d");
/* создаем переменную, которая будет записана в файл guest.txt */
$zapis = "\n<hr><br><b>$time<br><br>
От:</b> $name<br>
<b>E-mail: </b><a href=\"mailto: $email \">$email</a><br>
<b>Сообщение: </b>$msg<br><br>";
/* открываем файл с указанным в начале скрипта названием и записываем в него переменную $zapis */
$fp = fopen($files, "a+");
$fw = fwrite($fp, $zapis);
```

```
fclose($fp); }
?>
```

Вот и все. Сложно? На первый взгляд, может быть. Несколько практических применений — и все придет в норму.

Последняя наша задача совсем проста в своей реализации. Необходимо вставить в нужном месте файла `guestbook.htm` скрипт, который будет выводить на экран записи из файла `guest.txt`. Этот скрипт будет выглядеть так:

```
<?php include("guest.txt") ?>
```

Теперь все готово для проверки работоспособности скрипта. Вы можете загрузить все это на сервер (если ваша страница в Интернете) или проверить работу скрипта на локальном сервере. Если вы все сделали правильно, то все будет работать. Если что-то не получилось, что-то не работает, просмотрите главу еще раз и найдите ошибку.

33.2. Оптимизатор кода HTML

В наше время, когда большинство пользователей Интернета работают в нем через dial-up с помощью модема, остро стоит вопрос о скорости загрузки сайта. Очень долгая загрузка, естественно, не доставляет никому удовольствия. Конечно, основной способ уменьшить объем странички — использовать меньше графики, аппетов и т. д. Этим должны заниматься сами Web-мастеры. Есть и другой способ — оптимизировать код HTML, т. е. убрать лишние пробелы, переносы строк, заменить длинные теги аналогичными, но короткими. Если при этом сайт состоит, допустим, из ста страниц, то каждую редактировать долго и нудно. Лучше доверить эту работу скрипту.

Сначала нужно решить, что должен делать наш скрипт:

- убирать лишние пробелы,
- убирать все переводы строк,
- заменять некоторые длинные теги короткими.

Вы усмехнетесь и скажете, что этого мало и загрузка страничек едва ли ускорится. Но когда вы увидите, как скрипт уменьшает объем странички, сделанной в визуальном редакторе примерно на 30 %, вы убедитесь в своей неправоте.

Рассмотрим скрипт, который облегчит жизнь посетителям вашей странички.

```
/* тут начинается скрипт */
<?php
/* запускать скрипт следует с параметром files, равным имени
файла, подлежащего оптимизации */
/* открываем файл с именем, указанным в параметре files, для
чтения */
```

```
$fp = fopen($files, "r");
/* читаем данные из файла в массив $soo */
$soo = fread($fp, filesize( $files ));
/* убираем все переносы строки */
$soo = str_replace("\n","", $soo);
/* заменяем все пробелы более одного подряд одним пробелом */
$soo = str_replace("  ","", $soo);
$soo = str_replace("  ","", $soo);
$soo = str_replace("  ","", $soo);
$soo = str_replace("  ","", $soo);
$soo = str_replace("  ","", $soo);
$soo = str_replace("  ","", $soo);
$soo = str_replace("  ","", $soo);
$soo = str_replace("  ","", $soo);
$soo = str_replace("  ","", $soo);
/* надеемся, более десяти пробелов подряд нам не встретится */
/* теперь убираем все пробелы между кавычками > и < */
$soo = str_replace("> <","><", $soo);
$soo = str_replace("> <","><", $soo);
$soo = str_replace("> <","><", $soo);
$soo = str_replace("> <","><", $soo);
$soo = str_replace("> <","><", $soo);
$soo = str_replace("> <","><", $soo);
$soo = str_replace("> <","><", $soo);
$soo = str_replace("> <","><", $soo);
$soo = str_replace("> <","><", $soo);
$soo = str_replace("> <","><", $soo);
/* убираем все пробелы перед и после кавычек (думаем, больше 5
пробелов не будет)*/
$soo = str_replace("< ","<", $soo);
```

```
$soo = str_replace("< ","<", $soo);
$soo = str_replace("< ","<", $soo);
$soo = str_replace("< ","<", $soo);
$soo = str_replace("< ","<", $soo);
$soo = str_replace(" >",">", $soo);
$soo = str_replace(" >",">", $soo);
$soo = str_replace(" >",">", $soo);
$soo = str_replace(" >",">", $soo);
$soo = str_replace(" >",">", $soo);
/* а теперь заменим длинные теги короткими, аналогичными по сути */
$soo = str_replace("<address>", "<i>", $soo);
$soo = str_replace("<strong>","<b>", $soo);
$soo = str_replace("<em>","<i>", $soo);
$soo = str_replace("<strike>","<s>", $soo);
$soo = str_replace("<blockquote>", "<ul>", $soo);
$soo = str_replace("<var>","<i>", $soo);
$soo = str_replace("<cite>","<i>", $soo);
$soo = str_replace("<code>","<tt>", $soo);
$soo = str_replace("<kbd>","<tt>", $soo);
$soo = str_replace("<samp>","<tt>", $soo);
/* закрываем файл */
fclose($fp);
/* теперь открываем этот же файл для записи */
$fp = fopen($files, "w");
/* заменяем старый код оптимизированным из массива $soo */
$fw = fwrite($fp, $soo);
/* закрываем этот файл (теперь уже окончательно) */
fclose($fp);
/* конец скрипта */
?>
```

СОВЕТ

Для удобства можно сделать страничку с формой (одно поле текста с именем files и кнопка «Submit») и action, равным имени скрипта-оптимизатора, где нужно указать имя файла и нажать на кнопку. В этом случае нужно будет сделать мета-тег в скрипте, возвращающий вас на страничку с формой:

```
<meta http-equiv="refresh" content="0;URL=Ваша_страничка_с_формой">
```

Заключение

Теперь вы умеете писать скрипты на PHP. Фантазируйте, модифицируйте, обязательно придумывайте что-то свое. Ведь этот язык, на самом деле, дает очень широкий простор для вашей мысли. Реализовать можно очень много интересных вещей. И гостевая книга или счетчик — это капля в море, по сравнению с перспективой ваших идей в области PHP. Успехов вам в дальнейших разработках.

Приложения

Приложение А
Список опций конфигураций

Данные опции, поддерживаемые скриптами PHP4 configure, используются при компиляции в Unix-подобной среде.

Опции для баз данных

`--with-db`

Включает поддержку старого xDBM (не рекомендуется).

`--enable-dba=shared`

Строит DBA как совместно используемый (shared) модуль.

`--with-gdbm[=DIR]`

Включает поддержку GDBM.

`--with-ndbm[=DIR]`

Включает поддержку NDBM.

`--with-db2[=DIR]`

Включает поддержку Berkeley DB2.

`--with-db3[=DIR]`

Включает поддержку Berkeley DB3.

`--with-dbm[=DIR]`

Включает поддержку DBM.

`--with-cdb[=DIR]`

Включает поддержку CDB.

`--enable-dbase`

Дает возможность использовать связанную библиотеку dbase.

```
--with-dbplus
```

Включает поддержку dbplus.

```
--enable-dbx
```

Разрешает dbx.

```
--with-fbsql[=DIR]
```

Включает поддержку FrontBase. DIR — это базовая директория FrontBase.

```
--enable-filepro
```

Разрешает поддержку связанного read-only file Pro.

```
--with-fribidi[=DIR]
```

Включает поддержку fribidi (требуется FriBidi >=0.1.12). DIR — это директория инсталляции fribidi — default /usr/local/.

```
--with-informix[=DIR]
```

Включает поддержку Informix. DIR — это базовая директория инсталляции Informix, по умолчанию не определена.

```
--with-ingres[=DIR]
```

Включает поддержку Ingres II. DIR — это базовая директория Ingres (по умолчанию /II/ingres).

```
--with-interbase[=DIR]
```

Включает поддержку InterBase. DIR — это базовая директория инсталляции InterBase, по умолчанию это /usr/interbase.

```
--with-msql[=DIR]
```

Включает поддержку mSQL. DIR — это базовая директория инсталляции mSQL, по умолчанию это /usr/local/Hughes.

```
--with-mysql[=DIR]
```

Включает поддержку MySQL. DIR — это базовая директория MySQL. Если не специфицирована, используется связанная библиотека MySQL.

```
--with-oci8[=DIR]
```

Включает поддержку Oracle-oci8. DIR — по умолчанию ORACLE_HOME.

```
--with-adabas[=DIR]
```

Включает поддержку Adabas D. DIR — это базовая директория инсталляции Adabas, по умолчанию это /usr/local.

`–with-sapdb[=DIR]`

Включает поддержку SAP DB. DIR — это базовая директория инсталляции SAP DB, по умолчанию — `/usr/local`.

`–with-solid[=DIR]`

Включает поддержку Solid. DIR — это базовая директория инсталляции Solid, по умолчанию — `/usr/local/solid`.

`–with-ibm-db2[=DIR]`

Включает поддержку IBM DB2. `DIR` — это базовая директория инсталляции DB2, по умолчанию — `/home/db2inst1/sqllib`.

`–with-empress[=DIR]`

Включает поддержку Empress. `DIR` — это базовая директория инсталляции Empress, по умолчанию — `$EMPRESSPATH`. Начиная с PHP 4, эта опция поддерживает только Empress Version 8.60 и выше.

`–with-empress-bcs[=DIR]`

Включает поддержку Empress Local Access. `DIR` — это базовая директория инсталляции Empress, по умолчанию — `$EMPRESSPATH`. Начиная с PHP 4, эта опция поддерживает только Empress Version 8.60 и выше.

`–with-birdstep[=DIR]`

Включает поддержку Birdstep. `DIR` — это базовая директория инсталляции Birdstep, по умолчанию это `/usr/local/birdstep`.

`–with-custom-odbc[=DIR]`

Включает поддержку определенной пользователем ODBC. DIR — это базовая директория инсталляции ODBC, по умолчанию это /usr/local. Убедитесь, что определена `CUSTOM_ODBC_LIBS` и что файлы `odbc.h` имеются в ваших include dirs. Например, вы должны определить следующее для Sybase SQL Anywhere 5.5.00 на QNX, прежде чем запускать скрипт **конфигурирования:** `CPPFLAGS="-DODBC_QNX -DSQLANY_BUG" LDFLAGS=-lunix CUSTOM_ODBC_LIBS="-ldblib -lodbc"`.

`–with-iodbc[=DIR]`

Включает поддержку iODBC. `DIR` — это базовая директория инсталляции iODBC. по умолчанию — /usr/local.

`–with-esoob[=DIR]`

Включает поддержку Easysoft OOB. DIR — это базовая директория инсталляции OOB, по умолчанию — `/usr/local/easysoft/oob/client`.

```
--with-unixODBC[=DIR]
```

Включает поддержку unixODBC. DIR — это базовая директория инсталляции unixODBC, по умолчанию это `/usr/local`.

```
--with-openlink[=DIR]
```

Включает поддержку OpenLink ODBC. DIR — это базовая директория инсталляции OpenLink, по умолчанию это /usr/local. То же самое, что и для iODBC.

```
--with-dbmaker[=DIR]
```

Включает поддержку DBMaker. DIR — это базовая директория инсталляции DBMaker, по умолчанию это там, где установлена последняя версия DBMaker (как `/home/dbmaker/3.6`).

```
--with-oracle[=DIR]
```

Включает поддержку Oracle-oci7. DIR — по умолчанию `ORACLE_HOME`.

```
--with-ovrimos[=DIR]
```

Включает поддержку Ovrimos SQL Server. DIR — это директория инсталляции Ovrimos' libsqlcli.

```
--with-pgsql[=DIR]
```

Включает поддержку PostgreSQL. DIR — это базовая директория инсталляции PostgreSQL, по умолчанию это `/usr/local/pgsql`.

```
--with-sybase[=DIR]
```

Включает поддержку Sybase-DB. `DIR` — это домашняя директория Sybase, по умолчанию это /home/sybase.

```
--with-sybase-ct[=DIR]
```

Включает поддержку Sybase-CT. `DIR` — это домашняя директория Sybase, по умолчанию это /home/sybase.

Опции для графики

```
-with-gd[=DIR]
```

Включает поддержку GD (DIR — это директория инсталляции GD).

```
--enable-gd-native-ttf
```

GD: Включает строковую функцию TrueType в gd.

```
--with-jpeg-dir=DIR
```

GD: Устанавливает путь к префиксу установки libjpeg.

`--with-png-dir=DIR`

GD: Устанавливает путь к префиксу установки libpng.

`--with-xpm-dir=DIR`

GD: Устанавливает путь к префиксу установки libXpm.

`--with-freetype-dir=DIR`

GD: Устанавливает путь к префиксу установки freetype2.

`--with-ttf[=DIR]`

GD: Включает поддержку FreeType 1.x.

`-with-t1lib[=DIR]`

GD: Включает поддержку T1lib.

`--with-cpdflib[=DIR]`

Включает поддержку cpdflib (требует cpdflib >= 2). DIR — это директория инсталляции cpdfllib, по умолчанию это `/usr`.

`--with-jpeg-dir[=DIR]`

jpeg dir для cpdflib 2.x.

`--with-tiff-dir[=DIR]`

tiff dir для cpdflib 2.x.

`--with-pdflib[=DIR]`

Включает поддержку PDFlib. DIR — это базовая директория инсталляции pdflib, по умолчанию это `/usr/local`.

`--with-jpeg-dir[=DIR]`

PDFLIB: определяет директорию инсталляции libjpeg.

`--with-png-dir[=DIR]`

PDFLIB: определяет директорию инсталляции libpng.

`--with-tiff-dir[=DIR]`

PDFLIB: определяет директорию инсталляции libtiff.

`--with-swf[=DIR]`

Включает поддержку swf.

`--with-ming[=DIR]`

Включает поддержку ming.

Другие опции

`--enable-force-cgi-redirect`

Включает проверку безопасности (security) для внутренних перенаправлений сервера. Вы должны использовать это, если запускаете CGI-версию PHP с Apache.

`--enable-discard-path`

Если включена, то исполнимый файл PHP, установленный как CGI-программы, может быть безопасно размещен вне Web-дерева, и никто не сможет обойти `.htaccess`.

-with-fastcgi=SRCDIR

Строит PHP как FastCGI-приложение.

`--enable-debug`

Компиляция с символами отладки.

`--with-layout=TYPE`

Устанавливает расположение инсталлированных файлов. `TYPE` — это PHP (по умолчанию) или GNU.

-with-pear=DIR

Устанавливает PEAR в DIR (по умолчанию `PREFIX/lib/php`).

`--without-pear`

Не устанавливать PEAR.

`--with-openssl[=DIR]`

Включает поддержку OpenSSL (требует OpenSSL >= 0.9.5).

`--enable-sigchild`

Включить в собственный PHP-обработчик SIGCHLD.

`--disable-rpath`

Отключить передачу путей поиска дополнительной runtime-библиотеки.

`--enable-libgcc`

Включить явную компоновку относительно libgcc.

`--enable-dmalloc`

Включить dmalloc.

`--enable-php-streams`

Включает поддержку экспериментальных PHP-потоков. Не используйте, пока не протестировали код!

-with-zlib-dir=<DIR>

Определяет местоположение директории инсталяции zlib.

```
--with-zlib[=DIR]
```

Включает поддержку zlib (требует zlib >= 1.0.9). DIR — это директория инсталяции zlib.

```
--with-aspell[=DIR]
```

Включает поддержку aspell.

```
--enable-bcmath
```

Разрешает использовать математические функции в стиле bc.

```
--with-bz2[=DIR]
```

Включает поддержку BZip2.

```
--enable-calendar
```

Включает поддержку конвертации календарей.

```
--with-ccvs[=DIR]
```

Включает поддержку CCVS.

```
--with-crack[=DIR]
```

Включает поддержку crack.

```
--enable-ctype
```

Включает поддержку ctype.

```
--with-curl[=DIR]
```

Включает поддержку **CURL.**

```
--with-cybercash[=DIR]
```

Включает поддержку CyberCash. DIR — это директория инсталляции CyberCash MCK.

```
--with-cybermut[=DIR]
```

Включает поддержку CyberMut (система интернет-рассчетов, предлагаемая французским банком Credit Mutuel).

```
--with-cyrus
```

Включает поддержку cyrus IMAP.

--enable-exif

Включает поддержку exif.

-with-fdftk[=DIR]

Включает поддержку fdftk.

--enable-ftp

Включает поддержку FTP.

--with-gettext[=DIR]

Включает поддержку GNU gettext. DIR — это директория инсталляции gettext, по умолчанию это /usr/local.

--with-gmp

Включает поддержку gmp.

--with-hyperwave

Включает поддержку Hyperwave.

--with-icap[=DIR]

Включает поддержку ЮАР.

--with-iconv[=DIR]

Включает поддержку iconv.

--with-imap[=DIR]

Включает поддержку IMAP. DIR — это префикс инсталляции c-client.

--with-kerberos[=DIR]

ШАР: Включает поддержку Kerberos. DIR — это директория инсталляции Kerberos.

--with-imap-ssl[=DIR]

IMAP: Включает поддержку SSL. DIR — это директория инсталляции OpenSSL.

--with-ircg-config

Путь к скрипту ircg-config.

--with-ircg

Включает поддержку ircg.

--with-java[=DIR]

Включает поддержку Java. DIR — это базовая директория инсталляции JDK. Это расширение может быть построено только как shared dl.

`–with-ldap[=DIR]`

Включает поддержку LDAP. `DIR` — это базовая директория инсталляции LDAP.

`–enable-mailparse`

Включает поддержку mailparse.

`–enable-mbstring`

Включает поддержку мультибайтных строк.

`–enable-mbstr-enc-trans`

Включает поддержку перевода японской кодировки.

`–with-mcal[=DIR]`

Включает поддержку MCAL.

`–with-mcrypt[=DIR]`

Включает поддержку mcrypt. DIR — это директория инсталляции mcrypt.

`–with-mhash[=DIR]`

Включает поддержку mhash. `DIR` — это директория инсталляции mhash.

`–with-mnogosearch[=DIR]`

Включает поддержку mnoGoSearch. `DIR` — это базовая директория инсталляции mnoGoSearch, по умолчанию это `/usr/local/mnogosearch`.

`–with-muscat[=DIR]`

Включает поддержку muscat.

`–with-ncurses`

Включает поддержку ncurses.

`–enable-pcntl`

Включает экспериментальную поддержку pcntl (только CGI!).

`–without-pcre-regex`

Не включать поддержку Perl Compatible Regular Expressions. Используйте `with-pcre-regex=DIR` для специфицирования DIR, где размещены включаемые и библиотечные файлы PCRE, если не используется связанная библиотека.

`–with-pfpro[=DIR]`

Включает поддержку Verisign Payflow Pro.

```
--disable-posix
```

Отключает POSIX-подобные функции.

```
--with-pspell[=DIR]
```

Включает поддержку PSPELL.

```
--with-qtdom
```

Включает поддержку QtDOM (требует Qt >= 2.2.0).

```
--with-libedit[=DIR]
```

Включает поддержку замещения libedit readline.

```
--with-readline[=DIR]
```

Включает поддержку readline. DIR — это директория инсталляции readline.

```
--with-recode[=DIR]
```

Включает поддержку recode. DIR — это директория инсталляции recode.

```
--with-satellite[=DIR]
```

Включает поддержку CORBA через Satellite (эксперимент). DIR — это базовая директория инсталляции ORBit.

```
--with-mm[=DIR]
```

Включает поддержку mm для хранения сессий.

```
--enable-trans-sid
```

Включает прозрачное распространение session id.

```
--disable-session
```

Отключает поддержку сессии.

```
--enable-shmop
```

Включает поддержку shmop.

```
--with-snmp[=DIR]
```

Включает поддержку SNMP. DIR — это базовая директория инсталляции SNMP, по умолчанию поиски ведется в обычных местах размещения установки snmp.

```
--enable-ucd-snmp-hack
```

Включает пометку UCD SNMP.

```
--enable-sockets
```

Включает поддержку сокетов.

-with-regex=TYPE

Тип библиотеки regex: system, apache, php.

--with-system-regex

Использовать системную библиотеку regex (не рекомендуется).

--enable-sysvsem

Включает поддержку System V semaphore.

--enable-sysvshm

Включает поддержку совместно используемой памяти System V.

--with-vpopmail[=DIR]

Включает поддержку vpopmail.

--with-tsrm-pthreads

Использовать потоки POSIX (по умолчанию).

--enable-shared[=PKGS]

Построить совместно используемые (shared) библиотеки [default=yes].

--enable-static[=PKGS]

Построить статические библиотеки [default=yes].

--enable-fast-install[=PKGS]

Ортимизировать для быстрой инсталляции [default=yes].

--with-gnu-ld

Принять, что компилятор C использует GNU ld [default=no].

--disable-libtool-lock

Исключить блокирование (может прервать параллельные построения).

--with-pic

Попытаться использовать только объекты PIC/non-PIC [default=use both].

--with-yaz[=DIR]

Включает поддержку YAZ (ANSI/NISO Z39.50). DIR — это директория инсталляции YAZ bin.

--enable-memory-limit

Компилировать с поддержкой ограничения памяти.

--disable-url-fopen-wrapper

Запретить функции fopen открывать URL.

--enable-versioning

Экспортировать только необходимые символы. См. также файл INSTALL.

PHP-ОПЦИИ

--enable-maintainer-mode

Включает make-правила и зависимости, не используемые (и иногда путающие) в произвольном инсталляторе.

-with-config-file-path=PATH

Устанавливает путь для поиска php.ini, по умолчанию это PREFIX/lib.

--enable-safe-mode

Включает safe mode по умолчанию.

--with-exec-dir[=DIR]

Разрешает экзешники только в DIR, где safe mode включен, по умолчанию это /usr/local/php/bin.

--enable-magic-quotes

Включает "магические" кавычки по умолчанию.

--disable-short-tags

Отключает сокращенную форму <? начального тега по умолчанию.

Опции сервера

--with-aolserver=DIR

Специфицирует путь к установленному AOLserver.

--with-apxs[=FILE]

Строит shared Apache-модуль. FILE — это необязательный pathname к утилите Apache apxs; по умолчанию это apxs.

--with-apache[=DIR]

Строит Apache-модуль. DIR — это директория верхнего уровня построения Apache, по умолчанию это /usr/local/apache.

--with-mod_charset

Включает таблицы переноса для mod_charset (Rus Apache).

`--with-apxs2[=FILE]`

Строит shared Apache 2.0-модуль. FILE — это необязательный pathname к утилите Apache apxs; по умолчанию это apxs.

`--with-fhttpd[=DIR]`

Строит fhttpd-модуль. `DIR` — это директория исходников fhttpd, по умолчанию это `/usr/local/src/fhttpd`.

`--with-isapi=DIR`

Строит PHP как ISAPI-модуль для использования с Zeus.

`--with-nsapi=DIR`

Специфицирует путь к установленному Netscape Server.

-with-phttpd=DIR

Информации пока нет.

`--with-pi3web=DIR`

Строит PHP как модель для использования с Pi3Web.

`--with-roxen=DIR`

Строит PHP как Pike-модуль. DIR — это базовая директория Roxen, обычно это `/usr/local/roxen/server`.

`--enable-roxen-zts`

Строит Roxen-модуль с использованием Zend Thread Safety.

`--with-servlet[=DIR]`

Включает поддержку сервлетов. `DIR` — это базовая директория инсталляции JSDK. Для prereqs этого SAPI расширение java обязано быть построено как shared dl.

-with-thttpd=SRCDIR

Строит PHP как thttpd-модуль.

`--with-tux=MODULEDIR`

Строит PHP как TUX-модуль (только Linux).

XML-опции

-with-dom[=DIR]

Включает поддержку DOM (требует libxml >= 2.4.2). DIR — это директория инсталляции libxml, по умолчанию это /usr.

```
--disable-xml
```

Отключает поддержку XML с использованием связанной expat lib.

```
--with-expat-dir=DIR
```

XML: директория инсталяции external libexpat.

```
--with-xmlrpc[=DIR]
```

Включает поддержку XMLRPC-EPI.

```
--enable-wddx
```

Включает поддержку WDDX.

Приложение B
Зарезервированные слова PHP

Как и все языки программирования, PHP имеет свой перечень зарезервированных слов. Они не могут применяться в скриптах в качестве имен переменных, констант или других функций.

And	E_ERROR	PHP_VERSION
Break	E_WARNING	require ()
Case	Extends	require_once ()
Class	false	return
Continue	For	static
Default	Foreach	switch
Do	Function	this
Else	If	true
Elseif	include()	Var
Empty ()	include_once ()	Xor
Endfor	global	virtual ()
Endif	list()	while
Endswitch	new	__FILE_
Endwhile	not	__LINE_
E_ALL	or	
E_PARSE	PHP_OS	

Приложение C
Сообщения об ошибках

Чтобы сообщить об ошибках внутренней функции, нужно произвести вызов функции php3_error(). Для этого потребуется указать по крайней мере два параметра. Первый — уровень ошибки, второй — в форме строки сообщение об ошибке (как при стандартном вызове функции printf()). Любые следующие аргументы указываются только для того, чтобы создать нужный формат строки сообщения об ошибке. Уровни ошибок бывают следующие.

E_NOTICE

Примечания не печатаются по умолчанию, а указывают на то, что сценарий (наш скрипт) столкнулся с чем-то, что могло указывать ошибку, но могло также случиться и при нормальном выполнении скрипта. Например: попробуйте обратиться к значению переменной, которая не была определена, или вызывать функцию stat() для файла, который не существует.

E_ERROR

Ошибки также не печатаются по умолчанию, и выполнение сценария приостанавливается после функциональных возвращений. В этом случае указываются ошибки, которые нельзя восстановить, например проблемы распределения памяти.

E_PARSE

Ошибки синтаксического анализа генерируются только синтаксическим анализатором.

E_CORE_ERROR

Подобно E_ERROR, кроме этого сгенерирован ядром PHP. Функции не должны генерировать этот тип ошибки.

E_CORE_WARNING

Подобно E_WARNING, кроме этого сгенерирован ядром PHP. Функции не должны генерировать этот тип ошибки.

E_COMPILE_ERROR

Подобно E_ERROR, кроме этого сгенерирован при помощи Zend Scripting Engine. Функции не должны генерировать этот тип ошибки.

E_COMPILE_WARNING

Похожа на E_WARNING, кроме этого сгенерирована при помощи Zend Scripting Engine. Функции не должны генерировать этот тип ошибки.

E_USER_ERROR

Подобно E_ERROR, кроме этого сгенерирован в коде PHP при помощи функции trigger_error. Функции не должны генерировать этот тип ошибки.

E_USER_WARNING

Подобно E_WARNING, помимо этого сгенерирован, используя функцию trigger_error. Функции не должны генерировать этот тип ошибки.

E_USER_NOTICE

Подобно E_NOTICE, кроме этого сгенерирован при помощи функции trigger_error. Функции не должны генерировать этот тип ошибки.

E_WARNING

Обозначает условие, при котором PHP известно, что что-то неверно, но будет выполнено в любом случае.

Приложение D
Таблицы кодировки символов

Таблица кодов ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Кодовая таблица Windows(CP-1251)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		1	2	3	4	5	6	7	t	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255